Genetic Algorithms Applied to Floorplan Area Optimization Problem

Duygu Sap

August 2, 2008

Istanbul Technical University, Faculty of Science and Letters, Mathematical Engineering Department, Maslak, TR-34469 Istanbul, TURKEY.

Thesis for BSc in Mathematical Engineering
Advisor: Ali Ercengiz

Abstract: As the genetic algorithms have been perceived to be compatible for solving optimization problems, several techniques using genetic algorithms have been applied to a wide range of optimization problems including floorplan area optimization problem. In this study, the representations of the floorplan area designs are set forth with their effectiveness, a detailed analysis of the structure of genetic algorithms is made and the applications of genetic algorithms to the floorplan area optimization problem are discussed comparatively. Thus, the development of the floorplan area representations and the evolution of the genetic algorithms applied to the floorplan area optimization problem are observed.

Özet: Genetik algoritmaların optimizasyon problemlerinin çözümünde kullanılabilecek uygun yöntemler olduğunun belirlenmesi ile birlikte pek cok optimizasyon probleminin çözümünde olduğu gibi yerleşim planı alanı optimizasyonu probleminin çözümünde de genetik algoritmaların kullanıldığı çeşitli tekniklere başvurulmuştur. Bu çalışmada, yerleşim planı alanı tasarımlarının gösterilimleri etkinlik dereceleri ile birlikte verilmiş, genetik algoritma yapısının ayrıntılı bir analizi yapılmış ve yerleşim planı alanı optimizasyonu problemine yapılan genetik algoritma uygulamaları karşılaştırmalı olarak verilmiştir. Böylece, yerleşim planı alanı gösterilimlerinin gelişimi ve yerleşim planı alanı optimizasyonu problemine uygulanan genetik algoritmaların evrimi gözlemlenmiştir.

Contents

1. Introduction
2. Floorplan Area Representations
3. Genetic Algorithms
   (a) Biological Background
   (b) Genetic Algorithms
   (c) Some Important Terms
   (d) Applications of Genetic Algorithms
4. Genetic Algorithms Applied to the Floorplan Area Optimization Problem
5. Conclusion
6. References

1 Introduction

Floorplanning is an essential placement problem in VLSI (Very Large Scale Integration) physical design as it helps to determine a compact placement of a given set of circuit modules on the minimum bounding rectangle of a floor plan.

Very Large Scale Integration (VLSI) is the creation process of integrated circuits by integrating thousands of circuits made up of transistors into a single chip. In electronics, an integrated circuit called a chip is a small electronic circuit which has been manufactured in a thin substrate of the surface of a semiconductor material.

Integrated circuits are widely used in electronic equipments these days and have changed the electronics world radically. Integrating large numbers of tiny transistors into a small chip was an immense improvement over the manual assembly of circuits using discrete electronic components. As a result, the VLSI started during the development of complex semiconductor and communication technologies in the 1970s. As the number of complex functions required in data processing and telecommunications devices increases, the necessity of integrating these functions in a small system increase as well. This results in the augmentation of the effectiveness of the floorplanning to the VLSI design[12].

The basic principle of floorplanning is the determination of the relative positions of the modules and the selection of the best implementations for all the modules according to an evaluation function in order to minimize the chip area. Additionally, in some of the related works, floorplan design problem objects to minimize the total length of the wires connecting the terminals on the modules' boundaries to reduce the high fabrication costs and the complexity of the design.

Geometric relationships among the modules affect the module operations and the complexity of the design process. Thus,the floorplan representation structure has crucial effects on the flexibility, efficiency and effectiveness of the optimization problem. There are mainly two types of representations of floorplans: slicing floorplans and non-slicing floorplans. A slicing floorplan is the floorplan obtained by the recursive cuts of a rectangle in horizontal or vertical directions. A non-slicing floorplan is the floorplan which does not satisfy this condition[4] (See Figure-1). Slicing floorplans are represented by binary trees. Non-slicing floorplans are represented by Sequence Pair (SP), O-tree, and Corner Block List(CBL) techniques.

Generally, non-sliceable representations' solution space is larger than that of sliceable representations but it is shown that the solutions of sliceable representations are similar to the solutions of non-sliceable representations; furthermore they are preferable in solving large-size design problems as it is more advantageous to use a representation with a small solution space in such cases[3].

Moreover, genetic algorithms is another tool used in large-scale floorplan

design problems. In this paper, genetic algorithms using different kinds of genetic operators and referring to distinctive evolutionary methods are analyzed in solving the floorplan area optimization problem.

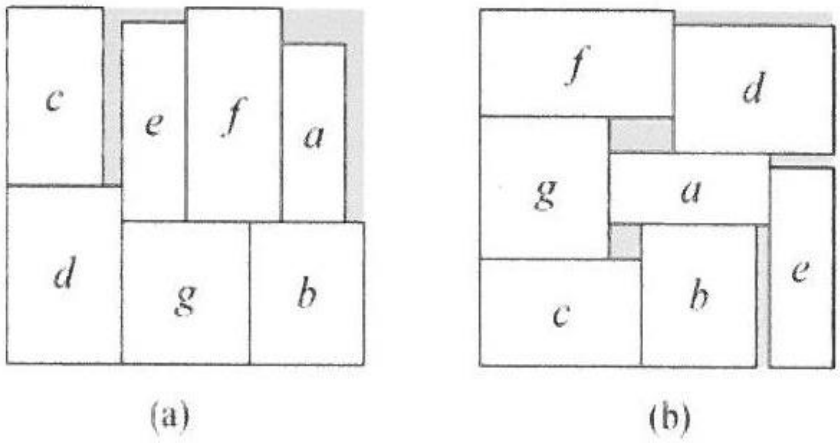


Figure-1: a) a sliceable structure b) a non-sliceable structure

2 Floorplan Representations

In representing slicing floorplans, it is possible to use slicing trees, which has the modules as its leaves and the operators representing the cuts as its internal nodes (See Figure-2).

Additionally, the slicing floorplan can be represented by a Polish expression through the post-order traversal of the slicing tree. Thus, for a slicing floorplan consisting of n modules, the Polish expression will be $(2n-1)$ long as the operators are also added in accordance with the places they occupy in the slicing tree[3]. By evaluating this expression using the stack structure one can obtain the dimensions of the bounding rectangle as follows:

- While the expression is scanned from left to right, the element composed of the width and the height of a module are pushed to the top of the stack whenever the module name is encountered.
- When $*$ operator is seen, both the top and next to-top elements in the stack are popped out and a new element having their total width as its width and maximal height as its height is added to the stack.
- When $+$ operator is seen, both the top and next to-top elements in the stack are popped out and a new element having their total height as its height and the maximal width as its width is added to the stack.

Eventually, only one rectangle i.e. the bounding rectangle is left in the stack[3].

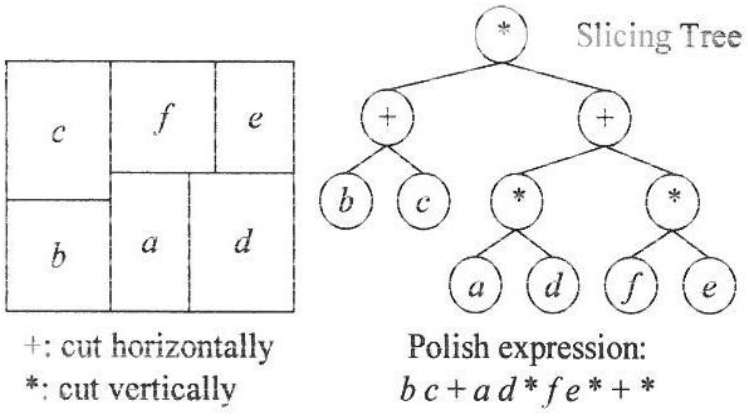


Figure-2: A floorplan with its corresponding slicing tree and Polish expression

In representing non-slicing floorplans, the SP (Sequence Pair), B* tree, O-tree and CBL (Corner Block List) are used.

In SP representations, pairs of module name sequences $(R+, R-)$ are used in representing a placement. These sequences are obtained by applying a method called Gridding to the placement. Every sequence pair corresponds to a feasible placement(See Figure-3). Under the left-of and below constraints, horizontal-constraint and vertical-constraint graphs are constructed and eventually the dimensions of the chip are determined by the longest path calculations in horizontal and vertical dimensions. Thus one of the optimal solutions is obtained [9].

- V: source s, sink t and m vertices are labeled with module names
- E: (s, x), (x, t) for each module x and (x, x') iff x is left of x'
- Vertex-weight: 0 for s and t, width of the module for any other module

The vertical constraint graph $G_V(V, E)$ based on the below constraint is also constructed in a similar way by using the heights of the modules(See Figure-4).

The longest path lengths between the source and the sink in these graphs provide the width and the height of the chip [9].

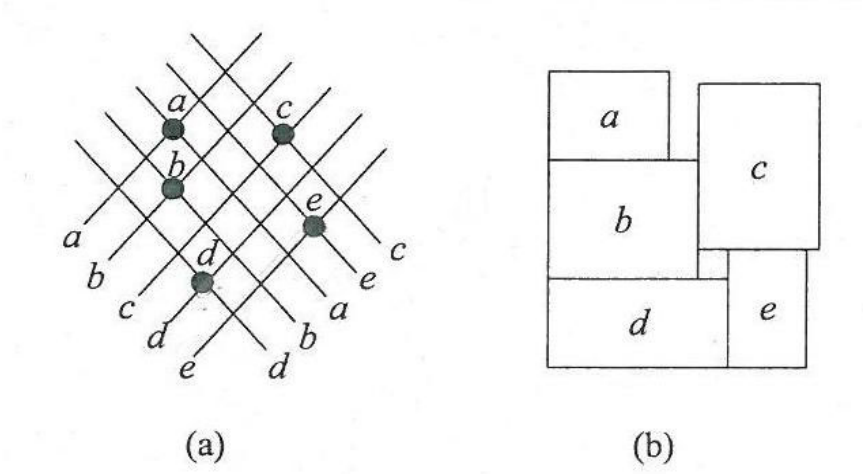


Figure-3: a) oblique grid for R+ ={a b c d e} and R- ={d b a e c}
b) resultant packing

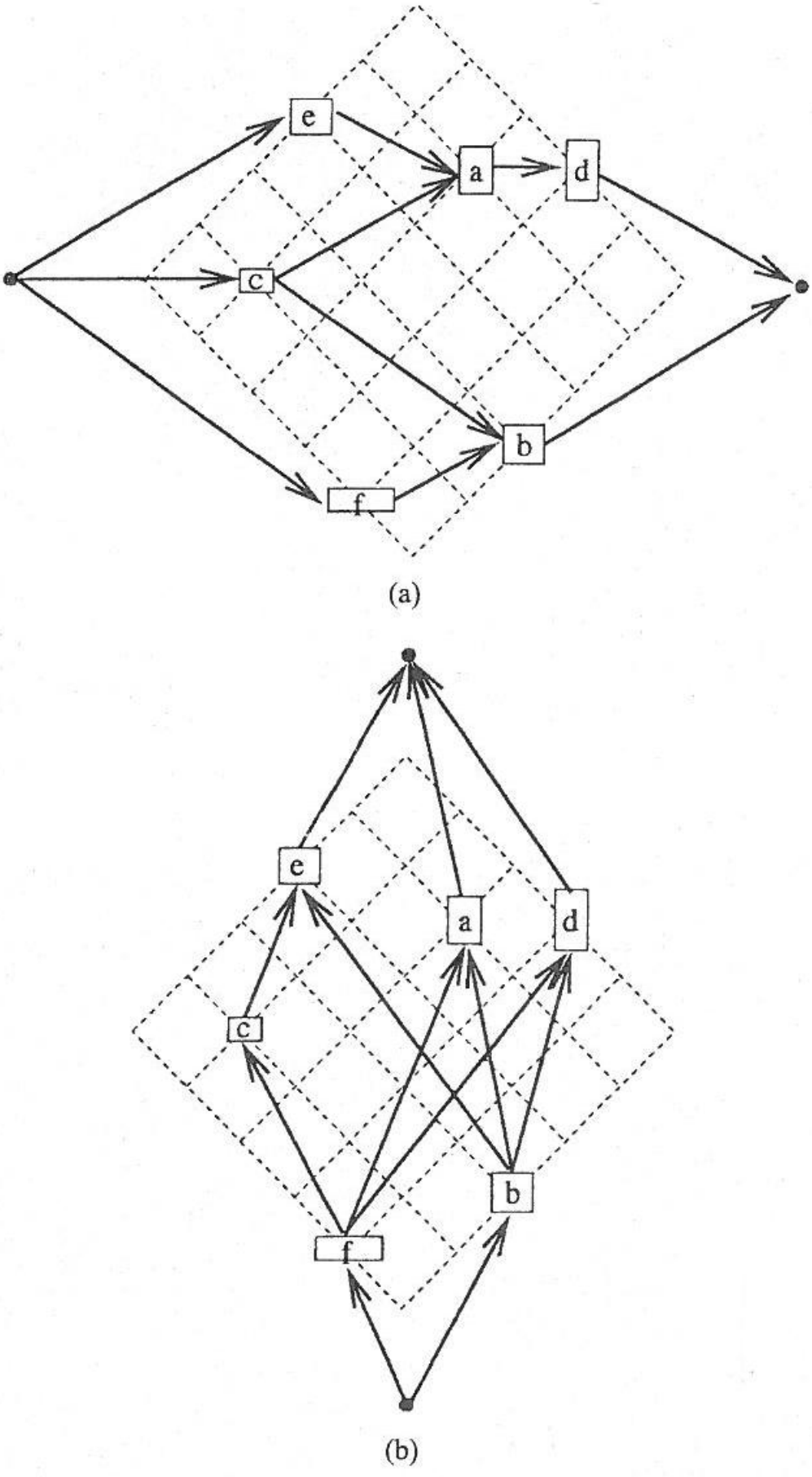


Figure-4: Constraint graphs. a) GH (above), b) GV (below)

In B* tree representations, the root of a B* tree is demonstrated by the module on the bottom-left corner of the placement and the rest of the tree is constructed through the DFS (Depth First Search) method. Starting from the root, firstly the left subtree and then the right subtree are constructed. Let R_i denote the modules on the right hand side and adjacent to the module b_i. The left child of the node n_i corresponds to the lowest unvisited module in R_i and the right child of n_i corresponds to the module above and adjacent to b_i with its x coordinate equal to b_i's and y coordinate less than the y coordinate of the top boundary of the module on the left hand side and adjacent to b_i if there exists such (See Figure-5). B*tree preserves the mutual relations between the modules. When the node n_j is the left child of the node n_i, module b_j should be located on the right hand side and adjacent to b_i with its x coordinate $x_j = x_i + w$ and when the node n_j is the right child of the node n_i, module b_j should be located above and adjacent to b_i with its x coordinate equal to the x coordinate of the module b_i[2].

O-tree structure differs from B*tree structure with its irregularity. Each node of an O-tree can have arbitrary number of branches(See Figure-6). This irregular structure of the O-tree makes the deviations from the optimal solution possible during the perturbation procedure including insertion and deletion operations used in the floorplan algorithm. This drawback of the O-tree representation circumscribes the quality of the resultant floorplan design . Another important property of the O-tree structure is that it can only represent an permissible placement which prevents modules from shifting left or down with other modules being fixed such that all the modules are compacted at the left and bottom boundary of the chip.

When the O-tree and B* tree representations are compared, B* tree representations are determined to perform 4.5 times faster than O-tree representations, which have been known as the fastest for the non-slicing floor plans, consuming 60 percent less memory and obtaining better results[2].

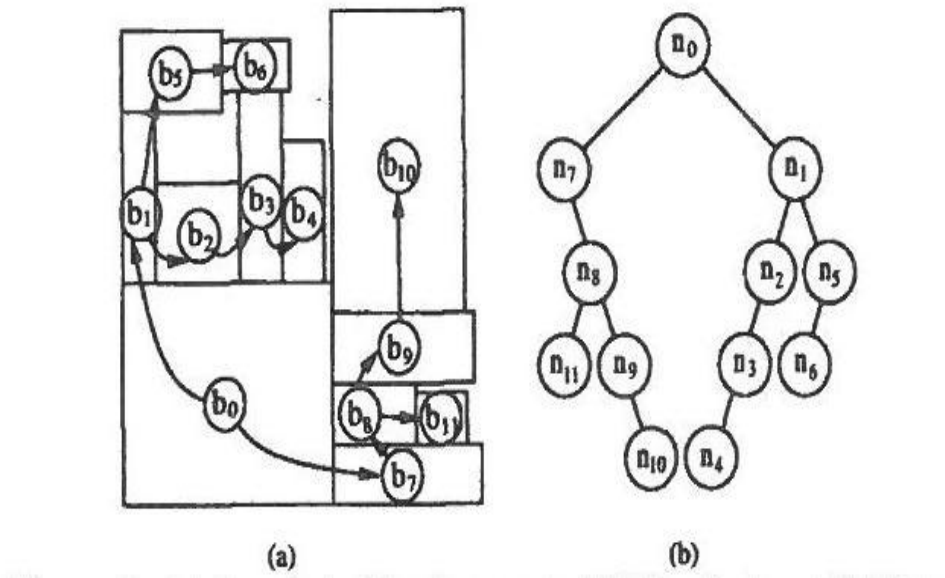


Figure-5: a) an admissible placement b) B* tree representing the placement

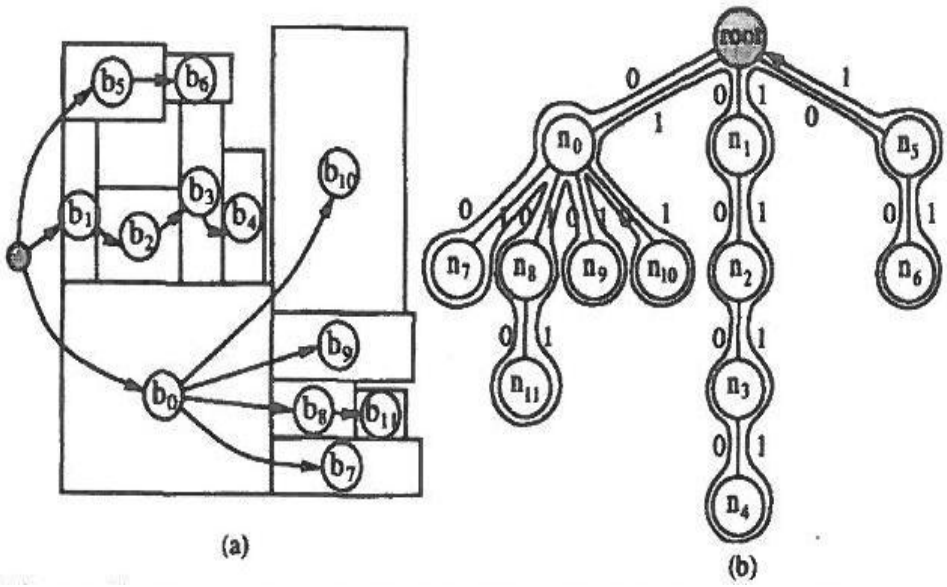


Figure-6: a) an admissible placement b) O-tree representing the placement

Another representation is Corner Block List (CBL) which is constructed from the record of recursive corner block deletions. For each deletion, a record of the deleted block name, block orientation and the number of the T-junctions is kept. A T-junction is a composition of a crossing and a non-crossing segment which has one end point joining it to the crossing segment's interval. At the end of the deletions, the data related with these three items concatenated in a reverse order and a sequence S of block names, a list L of orientations and a list T of T-junction information are obtained and this three tuple (S, L, T) is called the corner block. It should be noted that at the last block's (n_{th}) deletion, when there's only one block left in the placement, the orientation and the T-junction information of that block are not involved in the L and T lists. See Figure-7 for the formation of the Corner Block List $(dbaec, 0010, 001010)$

The number of 1s in the T list corresponds to the number of attached T-junctions and 0s are used to separate each string of 1s from the record of the next block deletion. Also, the 1s in the L list represent the horizontal orientations and the 0s represent the vertical orientations. [3].

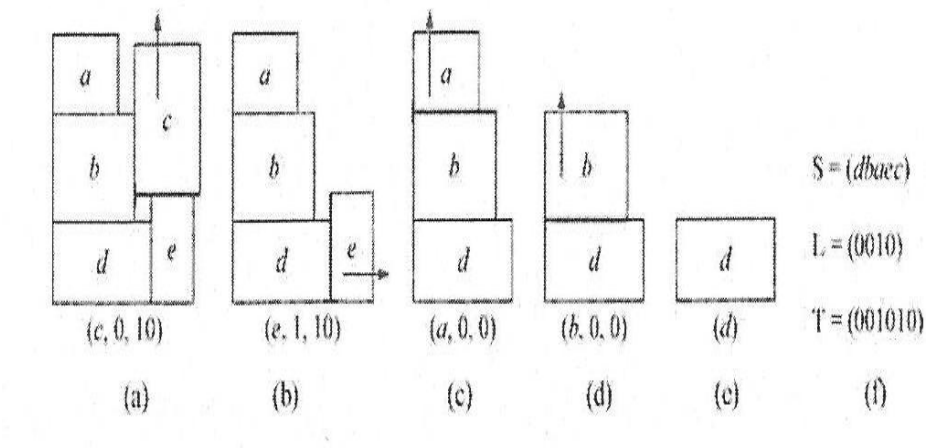


Figure-7: From (a) to (e) deletion of a packing (f) the corresponding CBL =(S, L, T)

The numerical results about the complexity and the performance of the above methods are given as follows: The upper bound of the number of possible combinations for Polish expression is $O(n!2^{3n-3}/n^{1.5})$ and the time required for the transformation of the representation to a physical packing is $O(n)$. Those values are also valid for the corner block list (CBL).The combinations for the sequence pair (SP) is $O(n!^2)$ and the transformation time is $O(nlglgn)$. The combinations and the transformation time for B* tree and O-tree are $O(n!2^{2n-2}/n^{1.5})$ and $O(n)$ respectively. Although these values are the same for B*trees and O-tree, one should keep in mind that an O-tree is irregular and has unpredictable number of branches that causes higher operation complexity and/or high encoding cost[3].

Comment

Non-slicing floor plan problems are much harder than slicing floor plan problems, which generally have solutions not optimal but can be handily represented and manipulated, so in this study the slicing floor plan techniques are going to be analyzed and evaluated[3].

3 Genetic Algorithms

3.1 Biological Background

3.1.1 Chromosomes, DNA and Genes

In the nuclei of every cell of living organisms there exist structures called chromosomes which supply the genetic information that is inherited from the ancestors to the offspring. Chromosomes are made up of DNA strands and proteins, which wrap these strands tightly[14].

DNA is a long polymer composed of units called nucleotides. Nucleotides are the organic compounds each of which consists of a nitrogenous base, a five carbon sugar (deoxyribose for DNA) and a phosphate group. DNA usually appears as a pair of strands associated in the shape of a double helix(See Figure-8). Each strand contains units of inheritance called genes[1].

Genes are located on particular positions called locus on the DNA strands. They are the functional blocks of DNA. They encode the genetic information needed in carrying out the process of building the proteins used in performing body functions and indicating several traits like eye color. A gene encodes the genetic information as a sequence of the four different nucleotides named after the bases they contain, A (Adenine), T (Thymine), C (Cytosine) and G (Guanine)(See Figure-9). These coding sequences - in other words the alternative forms of a gene are called alleles[14].

The complete set of chromosomes in a cell of an organism is called the organism's genome. Genotype is the certain set of genes in a genome. Under the influence of the environmental conditions, genotype determines the physical and mental properties of an organism known as the phenotype[8]

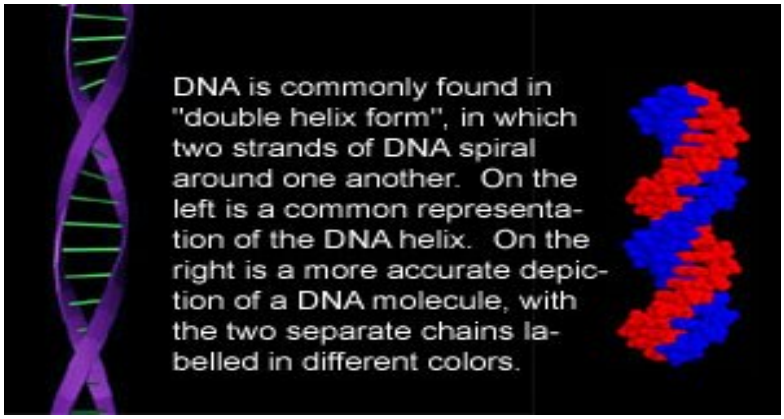


Figure-8: The double helix. Retrieved May 8, 2008 from http://www.globalchange.umich.edu/globalchange1/current/lectures/selection/selection.html

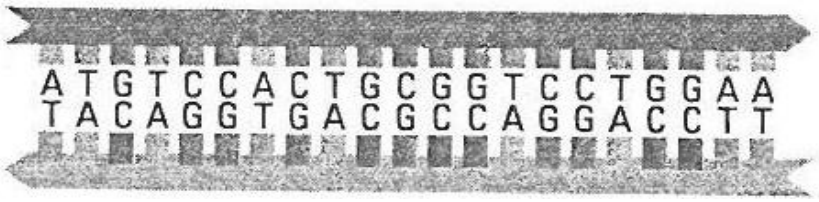


Figure-9: A nucleotide sequence in a DNA molecule

3.1.2 Crossover and Mutation

Organisms inherit traits from their parents in the form of genes. Both of the parents supply a group of chromosomes to their offspring. When the chromosomes pair up, they exchange sections of their DNA. Thus, the offspring don't carry exactly the same genetic information with their siblings and parents. This process is called the crossover.

In addition to the genetic diversity obtained through crossover, variations can also be caused by mutations. Mutations are the heritable changes of the genetic material that may be caused by the erratic copying of the genetic material during the cell division, exposure to the ultraviolet or ionizing radiation, chemical mutagens etc[10].

3.1.3 Natural Selection

Natural selection is the process that enables the opportune heritable traits to be passed to the successive generations of a population more commonly than the inopportune heritable traits due to the differential reproduction of genotypes. This process is mainly active on the observable characteristics of the organisms, phenotypes. The individuals with convenient phenotypes are more likely to survive and pass their traits to the successive generations by reproduction than the other individuals. Thus, the frequency of the genetic basis of the phenotype i.e. the genotype increases over the following generations. In the long run, this process may cause adaptations of a species to a specific environment and a new species may emerge eventually[6].

3.2 Genetic Algorithms

Genetic algorithms were proposed by John Holland in 1962 as a special class of evolutionary algorithms and developed by Holland and his colleagues at the University of Michigan in the 1960s and 1970s. Different from the other evolutionary algorithms designed for solving explicit problems, the aim of the genetic algorithms was to improve the comprehension of natural adaptation process and to design artificial systems similar to the natural systems[8].

Traditionally, the algorithm starts with the generation of a random popula-

tion. Every individual of the population represents a solution to the problem. During each generation, the fitness of each individual is computed by a fitness function defined in the first place. Then the genetic operators, crossover and mutation, are applied to the individuals to evolve the population. Usually the algorithm continues until reaching the maximum number of generations declared or an adequate fitness level settled for the population.

Thus, the three basic aspects in using genetic algorithms are:

- Definition of the genetic representation
- Definition of the fitness function
- Definition of the genetic operators

3.2.1 Genetic representations

In order to use the algorithm effectively, one should choose the encoding of the solutions cautiously. Strings of bits, arrays, trees, lists and sequences can be used as representations, but it should not be ignored that the choice of a representation can affect the performance and the cost of the algorithm in a noticeable way.

Here are some examples of different genetic encodings:

- Binary encoding

  Binary encoding is the most classical form of encoding. According to binary encoding, every gene has two alleles: 1 and 0. For instance, if the knapsack problem that objects to maximize the value of items that can be put in the knapsack, which has a certain capacity, is considered, the bits in the encoding represent the status indicating the presence of the relevant items in the knapsack[7].

  Chromosome A: 11010010110010
  Chromosome B: 10111100101110

- Permutation encoding

  Permutation encoding can be used in solving a well-known problem called the Traveling Salesman Problem (TSP). TSP objects to minimize the amount of distance taken in traveling all the given cities. It is used as a model problem for several problems such as the pen movement of a plotter, drilling of printed circuit boards (PCB), real-world routing of school buses, airlines, delivery trucks and postal carriers. Moreover, TSPs have been used in the study of biomolecular pathways, routing a computer networks' parallel processing, cryptography, determining the order

of exposures required in X-ray crystallography and in adjusting routes searching for forest fires. According to this encoding, every gene has 7 alleles each of which represents a city to be traveled[7].

Chromosome A: 2 5 6 1 3 4 7
Chromosome B: 7 6 5 1 3 2 4

3.2.2 Fitness function

The fitness function evaluates the quality of each solution and helps to find the best solution through evolution. Fitness function determines how often a solution will be selected for crossover.

According to a well-known selection method called fitness-proportionate selection, the expected number of times an individual is selected for crossover is $\frac{its\ fitness}{average\ fitness\ of\ the\ population}$ [8].

Roulette wheel method described in Section-3.3 is also based on the fitness-proportionate selection method.

3.2.3 Genetic operators

Genetic operators play a very essential role in finding the best individual of the population step by step. Crossover and mutation are the principal genetic operators.

The objective of the crossover is to inherit the good properties from the ancestors to the offspring and the objective of the mutation is to create diversity in the population and to prevent the algorithm being captured by a local optimum.

Here are some examples of several crossover and mutation techniques improved according to different types of genetic representations[7]:

- When binary strings are used in representing the solutions, the crossover is done through the division of the parent strings by one or more selected crossover points and the mutation is done through the inversion of the selected numbers as shown below.
  - *Single-point crossover*

$$\mathbf{10011}011 + 11011\mathbf{101} = \mathbf{10011101}$$

  - *Two-point crossover*

$$\mathbf{10}001\mathbf{001} + 11\mathbf{011}111 = \mathbf{10011001}$$

- *Mutation*

$$11100110 \mapsto 00111011$$

- When permutations are used in representing the solutions, the crossover is done through the selection of single points in permutations and the mutation is done through the inversion of the selected numbers as shown below.
  - *Crossover*

$$(435768) + (453786) = (435786)$$

  - *Mutation*

$$(345687) \mapsto (365487)$$

3.3 Some Important Terms

- **Population size**

  Population size is a significant parameter of a genetic algorithm. It corresponds to the number of individuals generated in a generation[3].

- **Crossover probability**

  Crossover probability conveys how often the crossover will take place. If this probability is 0 percent, then the new generation will be the exact copy of the old one when the mutation is discarded [3].

- **Mutation probability**

  Mutation probability conveys how often the chromosomes will be mutated.

  Note that the mutation probability is often kept smaller than the crossover probability in order to prevent the genetic algorithm from becoming a random search engine[3].

- **Elitism**

  Elitism is the preservation of the individual with the best fitness in a generation to keep the individual of high quality.[7].

- **Roulette Wheel Selection**

  Roulette Wheel selection is a technique commonly used for selecting the chromosomes that will cross over. It is based on the idea of selecting the individuals with high fitness more often than the others.

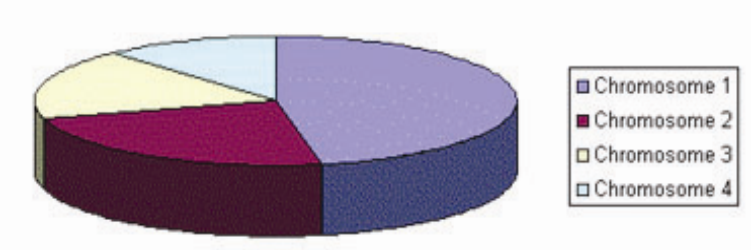


Figure-10: Roulette Wheel Selection. Retrieved May 8, 2008 from http://www.obitko.com/tutorials/genetic-algorithms/index.php

In the figure above, every chromosome shares a slice increasing in size proportionally to its fitness value. According to the roulette wheel selection, the chromosome with the biggest share is more likely to be chosen[8].

Assume that a marble is being thrown onto the disc to select a share. Then it is clear that the chromosome with a bigger share (higher fitness) is more likely to be selected.

3.4 Applications of GA

Genetic algorithms are applied to a great number of scientific and engineering problems. Here are some of the well-known application areas of GAs.

- *Optimization:* Genetic algorithms are widely used in numerical optimization and combinatorial optimization. Traveling salesman problem, job-scheduling problem, circuit design optimization problems are some of the combinatorial optimization problems genetic algorithms are applied to[8].
- *Automatic Programming:* Genetic algorithms are used in the evolution of computer programs for certain tasks and the design of some computational structures such as cellular automata[8].
- *Machine and robot learning:* Genetic algorithms are used in a plenty number of machine-learning applications and protein structure prediction. Furthermore, they are used in the design of neural networks, in the evolution of rules for learning classifier systems or symbolic production systems, and also in the design and controlling of robots[8].
- *Immune system models:* Genetic algorithms are used in modeling various aspects of the natural immune system such as the somatic mutation throughout the lifetime of an individual and the discovery of multi-gene families in the evolution period[8].
- E*conomics:* Genetic algorithms are used in modeling innovation processes, evolvement of bidding strategies and the emergence of economic markets[8].
- *Ecological models:* Genetic algorithms are used in modeling ecological phenomena such as biological arms races, host-parasite co-evolutions and resource flow in ecologies[8].
- *Population genetics models:* Genetic algorithms are used in the studies related with population genetics. For example, the answer to the question asking for the necessary conditions for the evolutionary viability of a gene for recombination is searched for by using genetic algorithms[8].
- *Interactions between evolution and learning:* Genetic algorithms are used in the study of the interaction between the individual learning and species evolution[8].
- *Social systems:* Genetic algorithms are used in the studies of several evolutionary aspects of social systems, such as the evolution of cooperation and communication in multi-agent systems[8].

An outline of a genetic algorithm

- *Step-1* Generate a random population of n chromosomes
- *Step-2* Compute the fitness(x) for each chromosome x
- *Step-3* Repeat the following for creating a new population
  - Select two parent chromosomes from the population depending on their fitness
  - Cross over the parents to form an offspring with a crossover probability p_c
  - Mutate the offspring with a mutation probability p_m
  - Add the new offspring to the newly generated population
- *Step-4* Replace the old population with the newly generated population
- *Step-5* Check the end condition.
  - If the condition is satisfied, stop the search and return the individual with the best fitness value.
  - If the condition is not satisfied, go to *Step-2*

The complete set of generations is called a run. One or more chromosomes with high fitness are generated at each run[8].

4 Genetic Algorithms Applied to the Floorplan Area Optimization Problem

As mentioned in the first section, the objective of the floorplan area optimization problem is to find the minimum area covering a given set of rectangular modules avoiding the possible overlaps.

We will mainly focus on the case where each solution of the problem i.e. each bounding rectangle is represented by a Polish expression. Thus, chromosomes are encoded by Polish expressions.

Although defining a chromosome as a Polish expression is a quite common technique, genetic operator definitions have been evolving to improve the quality of the solutions and accelerate the search process.

Here is a definition of some crossover operators used by Cohoon et al(1991) and Tazawa et al(1996).

Crossover operators

- *Operator-1*

  Let $P1$ and $P2$ be the chromosomes to be mated to produce the new chromosome $O1$. This operator copies the operands from $P1$ to $O1$ and the operators from $P2$ to the available positions in $O1$ in their original order. This can be illustrated as follows:

$$P1 : ab * dc + e * + fg * *hi + *j+$$

$$O1 : ab + dc + e * *fg + *hi + *j*$$

$$P2 : jhbc + d + ae * f * g + *i + **$$

- *Operator-2*

  This operator works similarly to the *Operator-1*. It copies the operands from $P2$ to $O2$ and the operators from $P1$ to the available positions in $O2$ in their original order. This can be illustrated as follows:

$$P1 : ab * dc + e * + fg * *hi + *j+$$

$$O2 : jhbc * d + ae * f + g * *i + *+$$

$$P2 : jhbc + d + ae * f * g + *i + **$$

- *Operator-3*

  This operator works slightly different from the preceding ones. Firstly, it copies the operators from $P1$ to $O3$ and then picks up a substring of

$P1$ and copies the operands of the substring to the corresponding places in $O3$. Finally, it copies the operands not included in $O3$ from $P2$ to the appropriate places in $O3$ keeping their order unchanged. This can be illustrated as follows:

$$P1 : ab * \texttt{dc+e*} + fg * *hi + *j+$$
$$O3 : jh * dc + e * +ba * *fg + *i+$$
$$P2 : jhbc + d + ae * f * g + *i + **$$

Comment

Random use of these operators prevents the offspring from inheriting the ancestors' good properties. Thus, good solutions cannot be found efficiently.

Another crossover operator which is more effective in finding good solutions more hastily is defined by Chen et al (2006). This operator is based on the idea of extracting and then growing the good subtrees from the parent chromosomes. Good subtrees are the area efficient subplacements.

Area efficiency is determined by the fitness function. The fitness of a chromosome P is:

$$fitness(P) = \frac{Area_R - \sum_{B_i \epsilon R, 1<=i<=n} Area_{B_i}}{Area_R}$$

where $Area_R$ is the area of the placement corresponding to P and $Area_{B_i}$ is the area of the i_{th} module.

The fitness of a placement is the measure of the ratio of the dead area to the area of the placement. Thus, the fitness value is inversely proportional to the area of the placement.

Consequently, a good subtree is defined as the subtree that has a fitness value smaller than or equal to a predefined threshold value V_{TH} .

Crossover operator

Let $P1$ and $P2$ be two chromosomes and $n(P1)$ and $n(P2)$ be the number of the good subtrees they have respectively.

Firstly, the good subtrees of the chromosome with higher $n()$ values are extracted and carried to the Good subtree Pool, GsP. Then the good subtrees of the other chromosome that don't have the modules already put into GsP are carried to GsP. After that, the modules which are not contained in any of the good subtrees in GsP are put into GsP under the name of degenerated subtrees denoted by Gsd.

After the determination of the good subtrees and construction of the GsP, good subtree growing period begins. There are three stages in this period. In the first stage, two degenerated subtrees are merged vertically or horizontally to form an enlarged subtree and the fitness of the new subtree is calculated by using one of the fitness functions below in accordance with the orientation of mergence.

- If the degenerated subtrees Gsd(i) and Gsd(j) with the dimensions (w_i, h_i) and (w_j, h_j) respectively are merged vertically to generate the subtree S . The fitness of the newly generated subtree S is:

$$f_v(S) = \frac{(h_i + h_j)max(w_i, w_j) - (h_i w_i + h_j w_j)}{(h_i + h_j)max(w_i, w_j)}$$

- If the degenerated subtrees Gsd(i) and Gsd(j) with the dimensions (w_i, h_i) and (w_j, h_j) respectively are merged horizontally to generate the subtree S . The fitness of the newly generated subtree S is:

$$f_h(S) = \frac{(w_i + w_j)max(h_i, h_j) - (h_i w_i + h_j w_j)}{(w_i + w_j)max((h_i, h_j)}$$

In the second and third stages, the same procedure is applied to the good subtree-degenerated subtree and good subtree-good subtree duples respectively. Consequently, the newly generated subtree is evaluated as follows:

- If the fitness value is smaller than the threshold value, then the generated subtree is put into the GsP replacing the generating subtrees.
- If the fitness value is larger than the threshold value, then the other orientation of mergence is applied to the subtrees and the new fitness value is calculated.
  - If the new fitness value is smaller than the threshold value, then the generated subtree is put into the GsP replacing the generating subtrees.
  - If it is still larger than the threshold value, then the generated subtree is discarded.

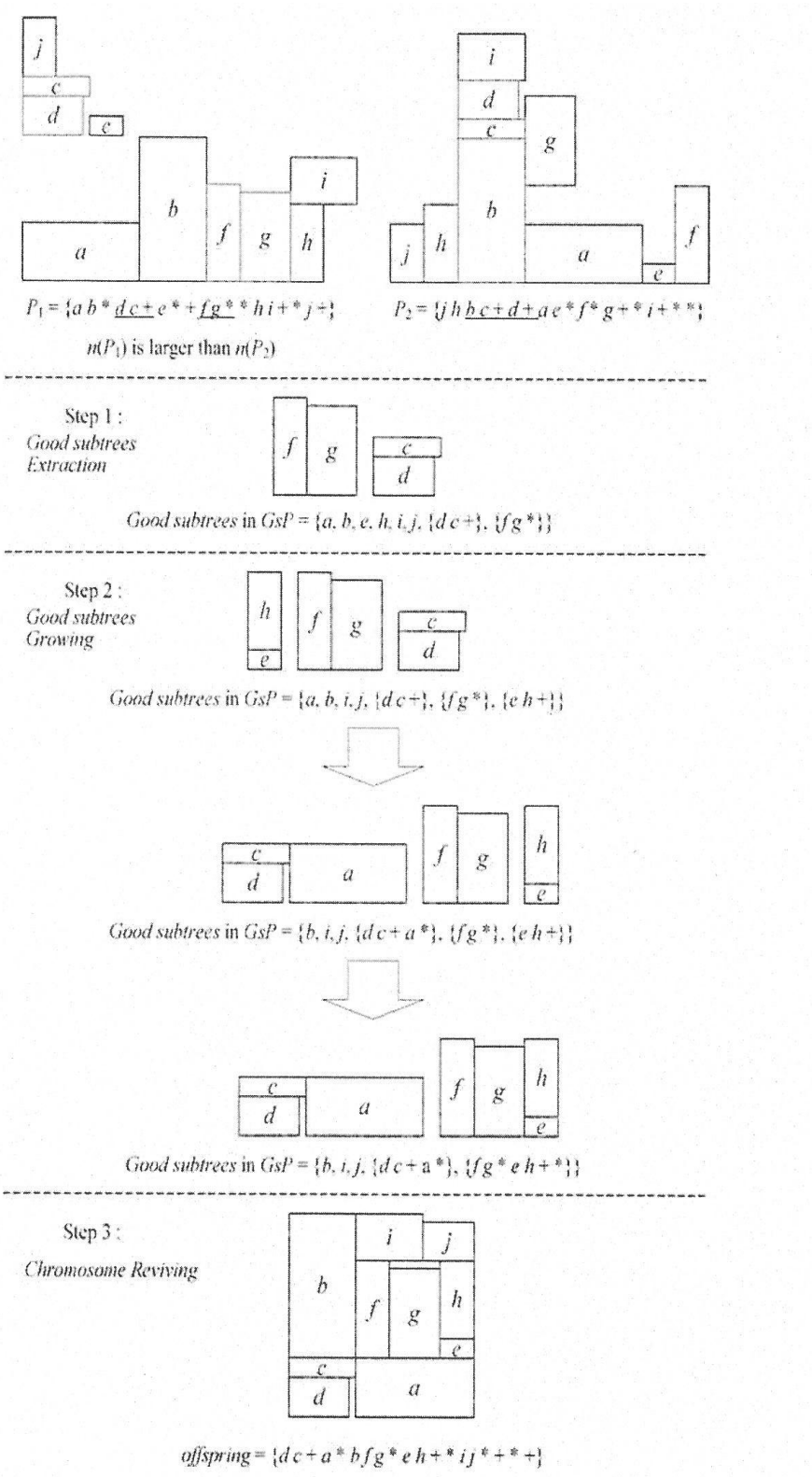
During this period, subtrees are allowed to rotate to increase the rate of mergence. Subtree rotation is performed in two steps. In the first step, the relational operators are mutually converted. In the second step, two subtrees associated with two sibling subtrees rooted at the relational operator + are swapped.

The good subtree growing period is followed by the chromosome reviving period. In this period, the threshold value V_{TH} is gradually increased and for

each new V_{TH} value good subtree growing period is repeated until a complete chromosome is generated in the GsP.

Once the complete chromosome is generated, its fitness value is compared with the fitness values of $P1$ and $P2$. If its fitness value is better than the fitness value of $P1$ (or $P2$), it replaces $P1$ (or $P2$); otherwise it is neglected.

Here's an application of this crossover technique:



Comment

By extracting and growing good subtrees, the good properties of the ancestors are inherited to the offspring. Thus, the deficiency of the preceding crossover techniques is covered.

Also note that the final chromosome is always feasible as the crossover operator always generates feasible chromosomes when applied to two feasible chromosomes. Thus, the newly obtained placement is a feasible solution to this problem. As a result, there is no need to use any repairing techniques to provide feasibility.

In addition to the crossover operations, some mutation operations are applied randomly in order to avoid the local optima.

Mutation

The mutation operators mentioned by Chen et al (2006) are as follows.

- *Operator-1* (complement)

  The relational operators in the Polish expression are mutually exchanged.

- *Operator-2* (exchange)

  A pair of subexpressions is exchanged.

- *Operator-3* (rotate)

  A module is randomly rotated as described in the good subtree growing step.

Here's the overall procedure of the proposed genetic algorithm implemented in C++ programming language on a PC with P4 1.8 GHz CPU by Chen et al (2006). It's applied to MCNC (Microelectronics Center of North Carolina) benchmarks and the results are compared with the ones obtained by using B* tree (Cheng et al., 2000), CBL (Hong et al., 2000), SP (Nakaya et al., 2000; Chrzanowska et al., 2002) methods (See Table-I).

`Genetic algorithm I`

Input: A set of modules $m_1, m_2, .., m_n$

Output: An area-efficient placement

Constraint: No modules are duplicate and overlapping

1.Randomly generate a set of chromosomes;

2.Evaluate each chromosome in the population;

3.Initialize the threshold value V_{TH};

4.**WHILE** (**NOT** satisfy stop condition)

begin

5. **FOR** i **IN** 1 **TO** R_C*popsize **LOOP**

6. Randomly choose P_1 and P_2 from the population;

7. Crossover($P1$,$P2$);

end FOR LOOP

8.**FOR** i **IN** 1 **TO** R_M*popsize **LOOP**

9.Randomly choose P from the population;

10.Mutation(P);

end FOR LOOP

11. **IF** (period)**THEN** Increase V_{TH} by a little value $\triangle$;

end

Table-I

	Non-Sliceable								Sliceable	
Engine	Simulated Annealing				EA		GA		GA	
Pres.	B*-tree (Chang et al., 2000)		CBL (Hong et al., 2000)		SP (Chrzanowska et al., 2002)		SP (Nakaya et al., 2000)		PE (ours)	
	Time (sec)	Area (mm^2)	Time (sec)	Area (mm^2)	Time (sec)	Area (mm^2)	Time (sec)	Area (mm^2)	Time (sec)	Area (mm^2)
aptes	7	46.92	NA	NA	NA	NA	NA	NA	<1 (<1)	46.92 (46.92)
xerox	25	19.83	30	20.96	NA	NA	NA	NA	<1 (<1)	20.36 (20.36)
hp	55	8.95	NA	NA	NA	NA	NA	NA	<1 (<1)	9.17 (9.15)
ami33	3417	1.27	36	1.2	57.24	1.21	1118	1.21	13 (7.43)	1.23 (1.19)
ami49	4752	36.8	65	38.5?	169.21	36.92	1978	37.14	37.03 (19)	36.82 (36.40)
Test66	NA	NA	86	2.38	NA	NA	NA	NA	63 (55.93)	2.41 (2.35)
Test99	NA	NA	110	3.67	NA	NA	NA	NA	95 (63.13)	3.68 (3.55)
Test198	NA	NA	165	7.51	NA	NA	NA	NA	99.16 (49)	7.77 (7.43)

Comment

As a result, it is concluded that a sliceable representation can be more advantageous than the non-sliceable representation based techniques (B* tree, CBL) when it is used with a well-designed search engine.

In another genetic algorithm applied to the floorplan area optimization problem (Rebaudengo and Reorda, 1996), some new genetic operators called heuristic operators are introduced along with the crossover and mutation operators used in the algorithm constructed by Cheng et al (2006) so as to improve the effectiveness of the method.

Here are some of the basic properties of this algorithm:

- Each population consists of P different individuals. I number of new individuals are produced at each generation. In order to avoid population homogeneity, maximum number of individuals having the same amount of area is fixed and every individual is different than the others in the same generation.
- Each individual is related with a list of implementations ordered for the ascending values of the aspect ratio i.e. $width/height$.
- Each individual is represented by a string of n genes where n denotes the number of the modules constituting the floorplan and every gene in the string represents the implementation selected as the implementation of the related module from its list of implementations.
- The fitness function is obtained from the floorplan area through linearization. The individual with the maximum area is assigned the maximum fitness value S and the individual with the minimum area is assigned the minimum fitness value 1. The other individuals' fitness values change within the range $(1, S)$. S is known as the selection pressure as its increase results in the ascension of the fitness values and consequently this decreases the probability of the selection of the worst individuals.

The representative code of the overall algorithm is as follows.

`Genetic algorithm II`

P_0= create_initial_population();

compute_initial_fitness();

$i = 0$;

while($stopping\_condition() \neq TRUE$)

{

$A = P_i$;

for $j = 0$ to I

{ /* new element generation */

op= select_an_operator();

s_j^i=apply_operator(op, P_i);

$A = A$ U s_j^i;

$j = j + 1$;

}

compute_fitness(A);

P_{i+1} = (the P best individuals ϵA);

$i = i + 1$;

}

Crossover operator

Crossover operator used in this algorithm selects the individuals by using the roulette wheel selection technique so that the individuals with high selection probability are chosen. Then, two numbers i and j in the range $(0, n)$ satisfying $i < j$ are randomly selected as the crossover points. After that, the new individual is generated by using the implementations of X from 0 to i and from $j + 1$ to n and the implementations of Y from $i + 1$ to j (See Figure-11).

Mutation Operator

Mutation operator aiming to explore the obscure regions of the search space randomly selects an individual. Then, the mutation takes place in three ways:

- By changing the value of a selected gene (See Figure-12(a))
- By generating an integer k and then randomly changing the genes $k, 2k, 3k, ..$ (See Figure-12(b))
- By choosing the next implementation in the implementation list for each gene. This corresponds to the rotation of all the modules (See Figure-12(c))

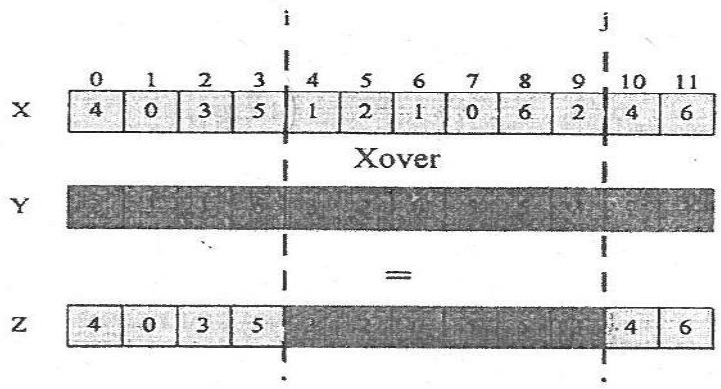
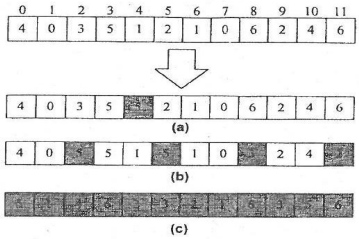


Figure-11: Two cuts crossover

Figure-12: Mutation operators a) concerning a single gene b) several genes and c) rotation



Heuristic Operators

Heuristic operators operate on randomly chosen individuals to improve their fitness values. There are two heuristic operators defined in obtaining the hybrid genetic algorithm connecting the heuristic operators to the genetic approach.

- *Operator-1* (HO_1):

  This operator evaluates all the implementations for a randomly selected module and replaces the current implementation by the implementation most suitable to the minimum floorplan area (See Figure-13).

- *Operator-2* (HO_2):

  This operator works on the horizontal and vertical adjacency graphs. When a non-slicing floorplan is considered, the horizontal adjacency graph is the graph where the vertices correspond to the vertical edges of the floorplan and the arcs correspond to the implementations for the basic modules, and the vertical adjacency graph is the graph where the vertices correspond the horizontal edges of the floorplan and the arcs correspond to the implementations for the basic modules (See Figure-14).

  A floorplan's height and width are assessed by the lengths of the longest paths in the vertical and horizontal adjacency graphs respectively. These paths can be defined as vertical and horizontal critical paths. HO_2 selects all the blocks in either of these paths leaving the common blocks of the paths out. Then in order to shorten the path, the implementation of each of these blocks is replaced by the preceding implementation in its implementation list if the blocks belong to the horizontal critical path and by the following implementation in its implementation list if the blocks belong to the vertical critical path. If the critical path is unique in the relevant direction and the size of the floorplan doesn't change in the other direction after this process then the floorplan area will be decreased (See Figure-15).

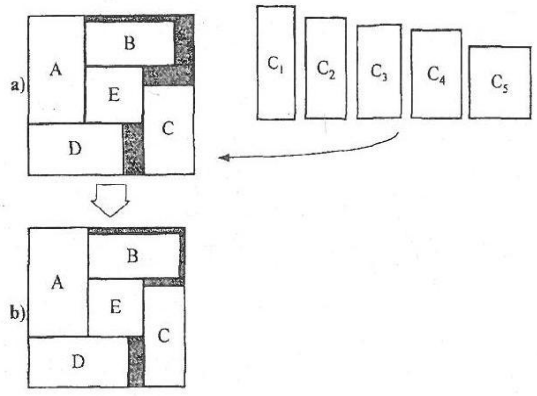


Figure-13: A floorplan. a) before and b) after the application of HO1

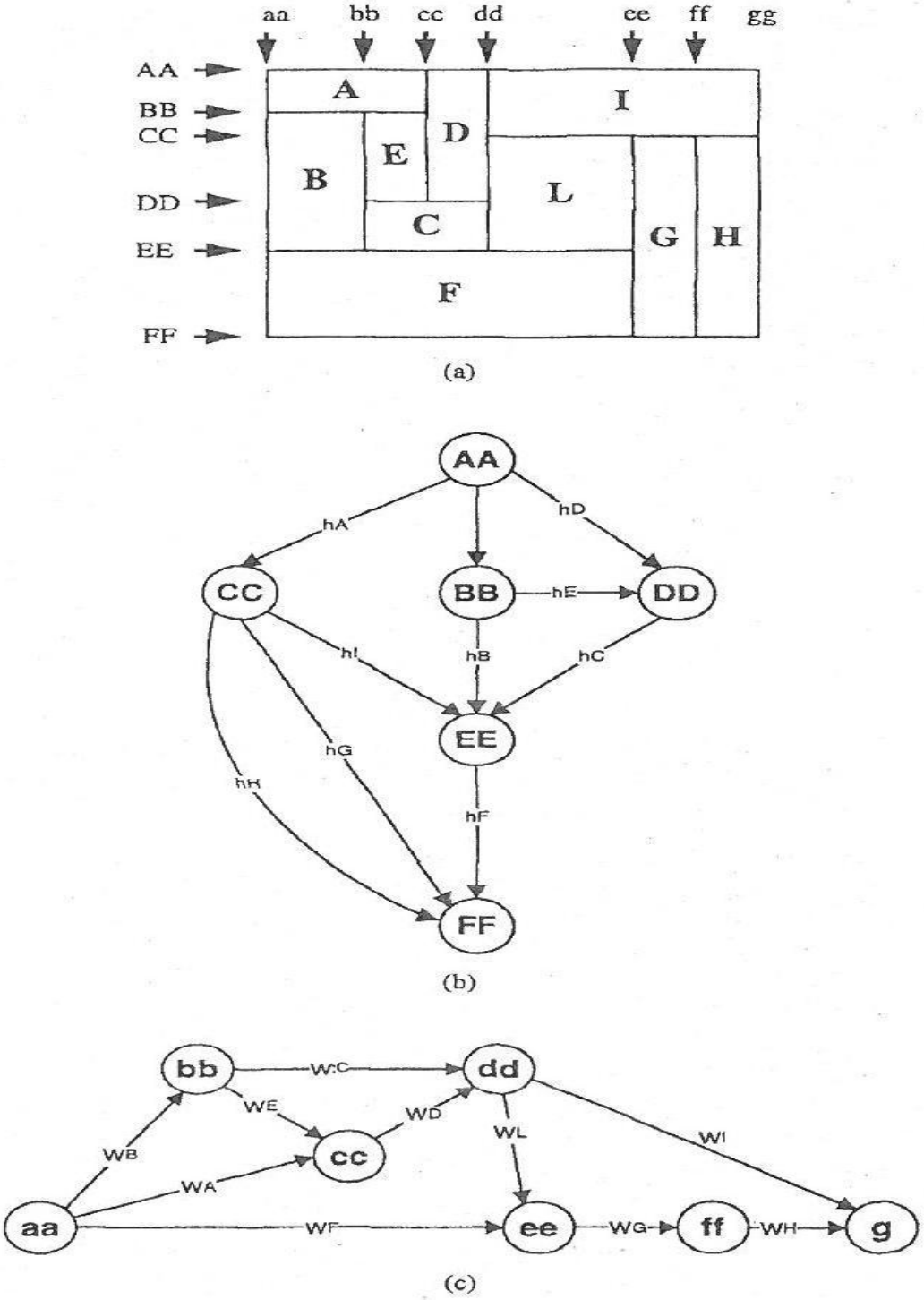


Figure-14: a) a floorplan b) its vertical and c)horizontal adjacency graphs

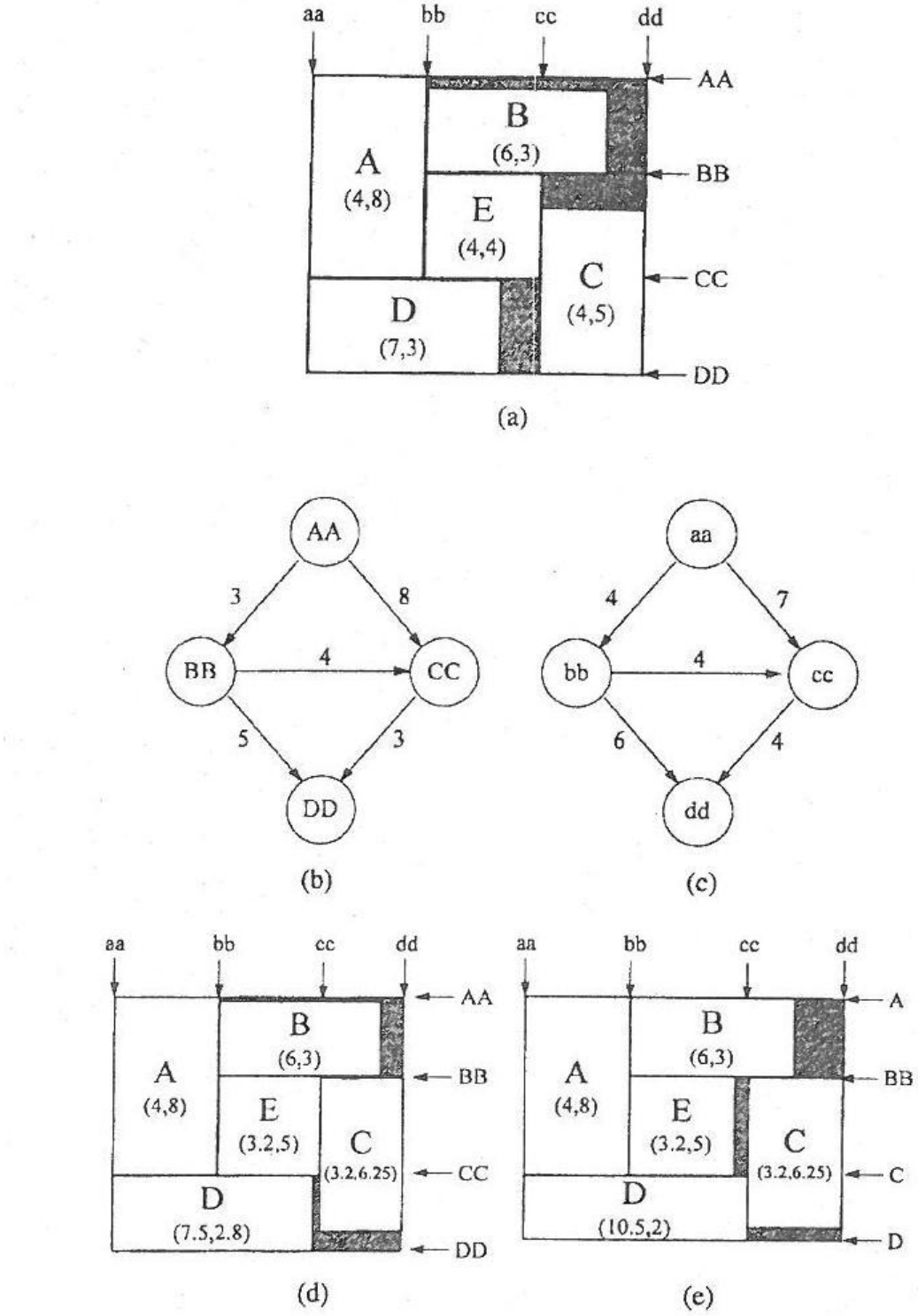


Figure-15: HO2 applied to a floorplan. a) a floorplan, b) vertical adjacency graph, c) horizontal adjacency graph, d) better result and e) worse result obtained by the application of HO2

In Figure-15, the vertical critical path is consist of the modules A and D and the horizontal critical path is consist of the modules A, E and C. Thus, the length of the horizontal critical path is 12 and the length of the vertical critical path is 11 and the resulting floorplan area is 132. HO_2 replaces the current implementation for the modules E and C by the preceding ones in their implementation lists and for the module D by the following one in its implementation list. The length of the new vertical critical path is 10.8 and the length of the new horizontal critical path is 10.4, thus the floorplan area is decreased to 112.32 (See Figure-15(d)).

On the other hand, the new implementation for a block reducing the critical path's length in one direction can transform a path in the other direction into a critical path. In Figure-15(e), it's assumed that the module D's following implementation's dimensions are 10.5 and 2. Although the length of the vertical path is reduced to 10, a new horizontal critical path consisting of the modules D and C is formed and the resulting floorplan area is increased to 137.

This algorithm called GALLO (Genetic Algorithm for Floorplan Area Optimization) is applied to 6 medium-large size benchmarks, namely $P40$, $P49$, $P120$, $P245$, $P1225$ and $P5000$. A maximum aspect ratio (w/h) of T and m implementations with aspect ratios varying between T and $1/T$ are used for all the benchmarks except $P120$. For $P120$, T is taken as 2 and 30 implementations are used. In order to evaluate the effectiveness of the algorithm, the stopping condition of each experiment is defined to be reaching a steady state i.e. the state when the best solution remains unchanged for a fixed number of generations. Each experiment is repeated for 10 times and the average of the results are computed to avoid the randomness[11].

Table-II

	Basic Algorithm			Improved Algorithm		
Example	Worst	Average	Best	Worst	Average	Best
P40	682	677	674	674	674	674
P49	3,233	3,233	3,233	3,233	3,233	3,233
P120	9,654	7,872	7,216	7,621	7,450	7,175
P245	4,691	4,689	4,687	4,688	4,688	4,687
P1225	41,726	41,716	41,710	41,711	41,711	41,710
P5000	410,710	406,740	404,270	398,475	398,441	398,382

(a)

	Basic Algorithm			Improved Algorithm		
Example	Worst	Average	Best	Worst	Average	Best
P40	14,000	10,600	6,000	8,000	5,600	4,000
P49	8,000	7,000	6,000	6,000	4,400	4,000
P120	24,000	15,600	6,000	14,000	10,400	8,000
P245	14,000	13,000	10,500	2,500	2,050	1,500
P1225	43,500	38,100	33,000	7,500	3,750	3,000
P5000	410,000	356,500	330,000	30,000	17,250	15,000

(b)

	Basic Algorithm			Improved Algorithm		
Example	Worst	Average	Best	Worst	Average	Best
P40	3.5	2.65	2.0	3.0	2.1	1.4
P49	3.2	2.4	2.1	2.8	2.3	2.0
P120	15.0	9.4	3.7	10.2	7.57	5.8
P245	24.4	22.6	18.7	7.7	6.2	5.1
P1225	232.8	219.6	191.4	101.1	50.5	39.1
P5000	7,530	6,314	5,810	1,329	757	656

(c)

Table-III

	Area	Generations	CPU Time [s]
No heur. op.	41,716	38,100	219
HO1	41,714	13,500	72
HO2	41,713	9,200	60
HO1 + HO2	41,711	3,750	50

Comment

Consequently, it is ascertained that the heuristic operators accelerate the search and enable achieving the optimal solution at a low computational cost. Additionally, more stable results have been obtained by the improved algorithm (See Table-II).

For a further comprehension of the benefits of the heuristic operators; a detailed analysis of P1225 is made (See Table-III)[11].

As seen in the above tables, the cooperation of HO_1 and HO_2 resulted in a slight improvement of the floorplan area in a reduced number of generations and CPU time.

On the other hand, heuristic operators are not determined to be accelerating factors throughout the whole run. They are completely ineffective at the final phases. As the local optimum implementation of each module has been already found, the contribution of HO_1 is hindered. Additionally, the great number of critical paths and blocks belonging to both the horizontal and vertical critical paths impedes the beneficial use of HO_2.

Therefore, for an efficient use of these operators an activation probability control mechanism has been developed. This mechanism is based on activating the operator with the higher activation probability. Activation probabilities are initially computed due to the costs of the operators. The cost of HO_1 is the number of times it computes the evaluation function for each module i.e. for n modules each of which possesses m implementations the cost of HO_1 is $n * m$. On the other hand, HO_2's cost is assessed by its complexity. HO_2 makes a longest path search twice in the graph so its complexity is $O(n^2)$.

Moreover, as stated before, the effectiveness of the operators are not fixed throughout a run, so a monitoring system is used to observe the changes in their effectiveness. As the activation probability of the most effective operator in a generation is increased, the activation probability of the other operator is decreased. The amount of change is inversely proportional to the operator's cost.

Different from the preceding algorithms, sometimes wirelength minimization is also taken into account in floorplan design problems as mentioned in the Introduction section. A genetic algorithm constructed for such a floorplan design problem is named GAPE which is inspired by an evolutional idea called punctuated equilibria and is aiming to use large scale, distributed memory, message-passing and parallel processing systems[5].

Punctuated equilibria is established upon two principles: allopatric speciation and stasis. Allopatric speciation is based on the evolution of a new species after the differentiation of a small set of members of a species to a new environment and statis claims that after reaching the equilibria in an environment the

genome of the species does not change remarkably[5].

According to this method, each module is represented by (A_i, l_i, u_i) where A_i is the area of the module and l_i and u_i are the lower and upper bounds on the valid aspect ratio respectively. A floorplan consisting of m modules and bounded by a rectangle R is divided into non-overlapping rectangular regions by horizontal and vertical cuts. Each region is denoted by r_i and should be large enough to comprise the relevant module i.e. $A_i <= x_i * y_i$ where x_i is the width and y_i is the height of the region r_i. The objective function contains two components: a total area component and a weighted wirelength component. The evaluation of a floorplan is expressed in its score which is computed as follows:

$$Score = \sum_{i=1}^{m} x_i y_i + \lambda \sum_{i,j=1}^{m} c_{ij} d_{ij}$$

where d_{ij} denotes the distance between the centers of the modules r_i and r_j , c_{ij} denotes the associated cost value of each connection between modules, which is equal to 0 if there's no connection between the relevant modules, and λ denotes the relative significance of the total area and the wirelength. It is assumed that $\lambda = 1$ by Cohoon et al. The best solution of the floorplan design problem is defined as the one with the minimum score[5].

Punctuated equilibria exhibit an efficient way of generating a new species which is drawing an old species to a new environment. According to the allopatric speciation, a continuous evolution is obtained by introducing the stabilized species to different environments. GAPE is related to the punctuated equilibria in the following way:

Each processor is considered as a disjoint environment and after a fixed number of generations some qualified species emerge. Then, as a result of a catastrophe some adjacent environments unite and form new environments. The amount of change occurring during the catastrophe period is controlled by the S factor in the structure given below.

Here's the genetic algorithm each processor uses:

initialize

for E iterations `do`

`parfor` each processor i `do`

run GA for G generations

`endfor`

`parfor` each processor i `do`

for each neighbour j of i `do`

send a set of solutions, S_{ij}, from i to j

`endfor`

`endfor`

`parfor` each processor i `do`

select an n element subpopulation

`endfor`

`endfor`

Crossover Operators

Let $P1$ and $P2$ be the parents reproducing the offspring O.

- CO_1: The operands from $P1$ and the operators from $P2$ are copied to the offspring O(See Figure-16).
- CO_2: The operators from $P1$ and the operands from $P2$ are copied to the offspring O(See Figure-17).
- CO_3: Firstly the operators from $P1$ are copied to O. Then a subtree $s1 = (b1, e1)$ of $P1$, which is the substring beginning with the module $b1$ and ending with the modules $e1$, is selected and copied to O. Finally the remaining operands from $P2$ are copied to O(See Figure-18).
- CO_4: Differs from the other ones by producing two offspring, $O1$ and $O2$. Firstly, substrings $s1 = (b1, e1)$ and $s2 = (b2, e2)$ are selected from $P1$ and $P2$ respectively. It should be confirmed that the lengths of these strings are the same. Then, for the reproduction of $O1$, operators are copied from $P1$ and the values of $P2(b2 + i)$ are assigned to $O1(b1 + i)$ for $i(0, e1 - b1)$. Finally, the operands of $P1$ are assigned to $O1(i)$ for $i < b1$ or $i > e1$.

The offspring $O2$ is reproduced in the same fashion by replacing $P1$ by $P2$ and $P2$ by $P1$(See Figure-19).

The crossover operators are selected randomly in the experiments made by using GAPE as it is confirmed that best average scores and best found scores are obtained this way.

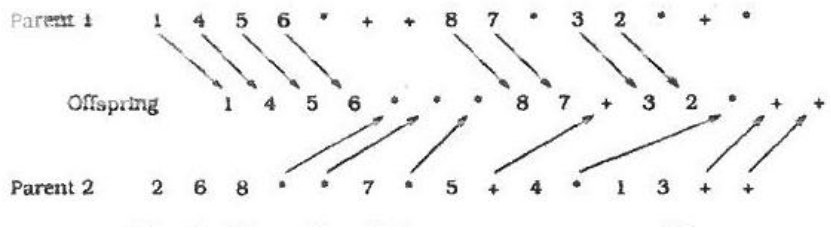


Figure-16

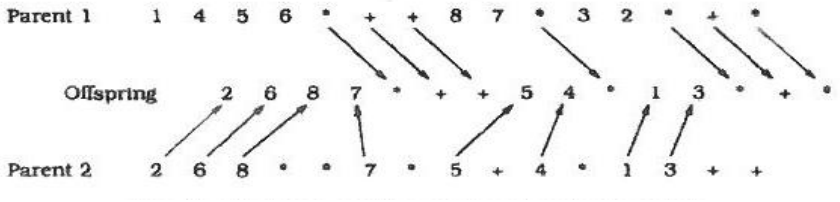


Figure-17

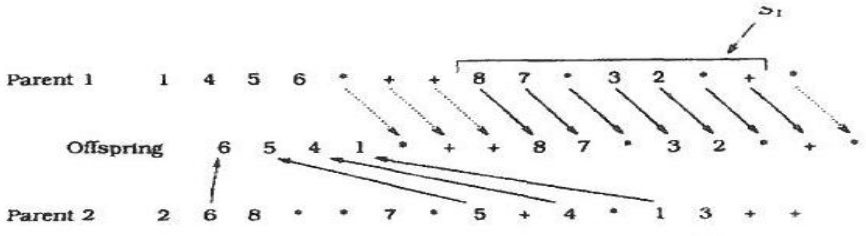
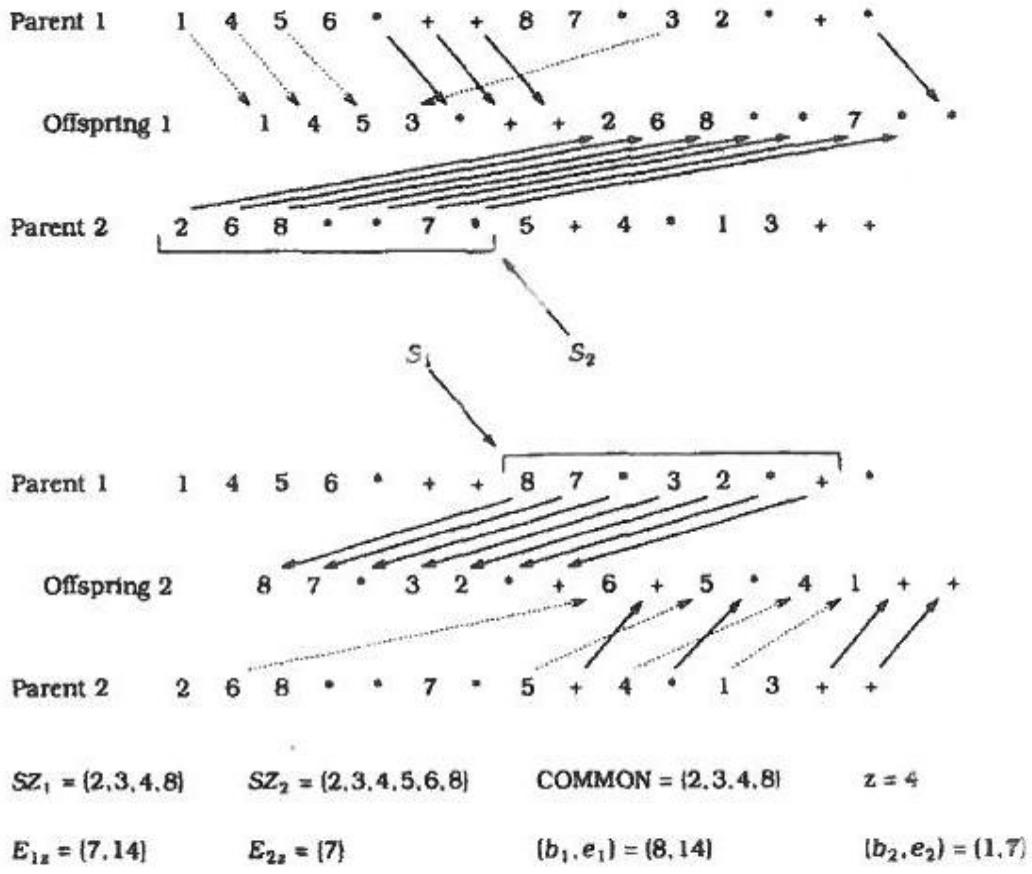


Figure-18



$SZ_1 = \{2,3,4,8\}$ $SZ_2 = \{2,3,4,5,6,8\}$ COMMON = {2,3,4,8} z = 4

$E_{1z} = \{7,14\}$ $E_{2z} = \{7\}$ $(b_1, e_1) = (8,14)$ $(b_2, e_2) = (1,7)$

Figure-19

Mutation

Mutation is done by the modification of a single individual locally. The methods of mutation are: swapping two adjacent operators, switching a sequence of adjacent operators and swapping an operator with a neighboring operand.

Comment

This algorithm is compared to a commonly used optimization technique SA (Simulated Annealing) and its performance is deduced to be better regarding the solutions' average computational cost and best-found solutions[5].

5 Conclusion

In this study, floorplan area design representations are defined and their effects on the performance of the optimization problem are discussed. The imitations of the natural evolutionary systems to the artificial systems are put forward by means of the genetic algorithm techniques applied to the floorplan optimization problem. The process of finding qualified solutions with low computational costs is analyzed and the development of the genetic algorithms applied to the floorplan area optimization problem is observed.

Consequently, it is deduced that natural evolutionary systems set a significant model for solving optimization problems. By the application of the evolution theory to the real-world problems, effective results can be obtained. Although the idea seems to be easily recognizable, the methods needs to be elaborative in order to reach satisfying solutions in reasonable CPU times.

On the other hand, the formulation i.e. the encoding is one of the most important steps in genetic algorithms. One should be cautious in determining the encoding that fits the relevant problem to avoid working in a complex environment with enormous runtime.

References

[1] *An Introduction to DNA and Chromosomes*. Retrieved May 8, 2008 from

http://hopes.stanford.edu/basics/dna/b1.html

[2] Chang, Y.C., Chang,Y.W., Wu,G.M., Wu, S.W.,2000. "B*-Trees: A new Representation for Non-Slicing Floorplans". In *Proceedings of the 37th ACM/IEEE Design Automation Conference*, pp. 458-461, Los Angeles, California, United States. Retrieved June 10, 2008 from

http://citeseer.ist.psu.edu/cache/papers/cs/15370/http:zSzzSzwww.cis.nctu.edu.twzSz~ywchangzSzPaperszSzbtree2k.pdf/chang00btrees.pdf

[3] Chen D.S., Lin C.T., Wang Y.W., 2006. "A Robust Genetic Algorithm for Rectangle Packing Problem". Retrieved April 21, 2008 from

http://www.springerlink.com/content/v19v037823654t08/

[4] Chen D.S., Lin C.T., Wang Y.W., 2002. "An Efficient Genetic Algorithm for Slicing Floorplan Area Optimization". In *Proceedings of International Symposium on Circuits and Systems*, p. 879, Phoenix-Scottsdale, AZ, USA. Retrieved May 8, 2008 from

http://www.springerlink.com/content/v19v037823654t08/

[5] Cohoon, J. P., Hegde, S. U., Martin, W. N.,Richards, D. S., 1991. "Distributed Genetic Algorithms for the Floorplan Design Problem". *IEEE Transactions on Computer-Aided Design*, vol.10, no.4, pp.483-489. Retrieved June 18, 2008 from

http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel1/43/2519/00075631.pdf?isnumber=2519&prod=JNL&arnumber=75631&arSt=483&ared=492&arAuthor=Cohoon%2C+J.P.%3B+Hegde%2C+S.U.%3B+Martin%2C+W.N.%3B+Richards%2C+D.S.

[6] *Evolution and Natural Selection*, 2005. Retrieved May 8, 2008 from

http://www.globalchange.umich.edu/globalchange1/current/lectures/selection/selection.html

[7] Obitko, M., 1998. *Genetic Algorithms*. Retrieved May 8, 2008 from

http://www.obitko.com/tutorials/genetic-algorithms/index.php

[8] Mitchell, M., 1996. *An introduction to Genetic Algorithms*. London:The MIT Press

[9] Murata, H., Fujiyoshi, K.,Nakatake, S., Kajitani, Y., 1996. "VLSI module placement based on rectangle-packing by the sequence-pair". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no.12, pp.1519-1522. Retrieved June 10, 2008 from

`http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel1/11994/00552084.pdf?temp=x`

[10] *Mutation*. Retrieved May 8, 2008 from

`http://en.wikipedia.org/wiki/Mutation`

[11] Rebaudengo, M., Reorda, M. S., 1996. "GALLO: A Genetic Algorithm for Floorplan Area Optimization". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol.15, no.8, pp.944-948 Retrieved June 10, 2008 from

http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel1/43/11089/00511573.pdf?arnumber=511573

[12] *Very-large-scale integration*. Retrieved May 8, 2008 from

`http://en.wikipedia.org/wiki/Very-large-scale_integration`

[13] Wall, M., (n.d). *Introduction to Genetic Algorithms*. Retrieved May 8, 2008 from

`http://lancet.mit.edu/~mbwall/presentations/IntroToGAs/index.html`

[14] *What is a chromosome?* Genetic Science Learning Center. Retrieved May 8, 2008 from

`http://learn.genetics.utah.edu/units/basics/tour/chromosome.swf`