New Metaheuristic Algorithms for Non-Slicing VLSI Floorplanning

by

Wali Mohammad Abdullah

MASTER OF SCIENCE IN INFORMATION AND COMMUNICATION TECHNOLOGY

Institute of Information and Communication Technology

BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY

2014

The thesis titled "**NEW METAHEURISTIC ALGORITHMS FOR NON-SLICING VLSI FLOORPLANNING**" submitted by Wali Mohammad Abdullah, Roll No.: 0411312005 P, Session: April/2011 has been accepted as satisfactory in fulfillment of the requirement for the degree of M. Sc. in ICT on 15th October, 2014.

Board of Examiners

1. _____
   Dr. M. Sohel Rahman — **Chairman**
   Professor
   Department of CSE, BUET, Dhaka.
   (Supervisor)

2. _____
   Dr. Md. Liakot Ali — **Member (Ex-Officio)**
   Professor and Director
   IICT, BUET, Dhaka.

3. _____
   Dr. Md. Liakot Ali — **Member**
   Professor and Director
   IICT, BUET, Dhaka.

4. _____
   Dr. Mohammad Nurul Huda — **Member (External)**
   Professor
   Department of CSE
   United International University (UIU), Dhaka.

CANDIDATE'S DECLARATION

It is hereby declared that neither this thesis paper nor any part thereof has been submitted anywhere else for the award of any degree, diploma or other qualifications.

_____-

Wali Mohammad Abdullah

DEDICATED TO PROPHET HAZRAT MUHAMMAD (SWM.)

Acknowledgement

Foremost, I am thankful to Almighty Allah for his blessings for the successful completion of my thesis. I would like to express my heartiest gratitude, profound indebtedness and deep respect to my supervisor, Dr. M. Sohel Rahman, Professor, Dept. of CSE, BUET, Dhaka, Bangladesh, for his constant supervision, affectionate guidance and great encouragement and motivation. His keen interest on the topic and valuable advices throughout the study was of great help in completing thesis.

I am especially grateful to Institute of Information and Communication Technology (IICT) of Bangladesh University of Engineering and Technology (BUET) for providing their all out support during the thesis work. My sincere thanks goes to IICT Office staffs for providing logistic support to me to successfully complete the thesis work.

Finally, I would like to thank my family: my parents Dr. Md Mohiuddin Abdullah and Afsir Begum, my wife Sharmin Islam, my sister, my brother and all of those who supported me for their appreciable assistance, patience and suggestions during the course of my thesis.

-Wali Mohammad Abdullah

Table of Contents

Abstract

A classical problem in VLSI floorplanning is to arrange a set of rectangular modules on a rectangular chip area ensuring that no two modules overlap with each other which provide an optimum measure of performance. In our thesis we incorporate different metaheuristic algorithms to solve the VLSI floorplanning problems which produce different results according to the applied algorithms. While applying these algorithms we incorporate the use of an effective method that is Clonal Selection Algorithm (CSA) with mutation operator to explore the search space. Then these algorithms also use a number of single state methods separately, to exploit the search space. This incorporation of CSA instead of other methods like crossover is a novel work in this area. Experimental results show that our metaheuristic algorithms provide better result than those algorithms which use other metaheuristic algorithms. In our thesis, we use the existing popular benchmarks with a view to comparing our results with the previous research works. Hence, our algorithms produce optimal or nearly optimal solutions for all the tested benchmark problems. Moreover, we develop visualization software, incorporating all our algorithms, along with graphical output of floorplans, which can show different comparative results on basis of statistical test and Kruskal-Wallis test for different applied algorithms.

Chapter 1
Introduction

1.1 Overview

Very Large Scale Integration (VLSI) is a method by which the functionality of many different types of electronic components are put into a small space or chip. Before the invention of VLSI technology, there were small and effective transistors to design circuits. During that time, the possibilities of constructing advanced circuits arose. But the main problem was the size of the circuit. If the components of a complex circuit were too large or the wires interconnecting the components were too long, the electric signals couldnot travel fast enough through the circuit. This slow transmission of electric signals made the circuit too slow as well. Due to this reason, a complex circuit, like a computer, could not be constructed with earlier technologies. Thus, the invention of VLSI technology provides us the opportunity to have the same functionality in less area/ volume. With the invention of VLSI method the size, cost and current consumption of the device are reduced significantly and the speed of operation increases as well [3, 39, 37, 54]. Floorplanning is important for VLSI design, because it takes the layout information into account at early stage of the design process. Floorplanning provides an overview by considering the layout at early stages and deliver an early feedback suggesting valuable architectural modification. In a top-down design strategy as well as in the stepwise refinement strategy this floorplanning concept fits very well. An example of VLSI flloorplan is given in Figure 1.1.

There are a number of floorplanning concepts. These are *slicing* and *non-slicing*. A slicing floorplan is a floorplan with the property that a composite cell's subcells are obtained by a horizontal or vertical bisection of the composite cell. Slicing floorplans can be represented by a *slicing tree*. In a slicing tree, all cells (except for the top level cell) have a *parent*, and all composite cells have *children*. Not all floorplans are slicing. The floorplans without the slicing property are called non-slicing floorplan.

Early in the design process, a floorplan is designed by using the estimation of the sizes of the blocks and the number of wires between those blocks. The area required for wiring is estimated during floorplanning. This initial floorplan serves as a budget for the design of actual

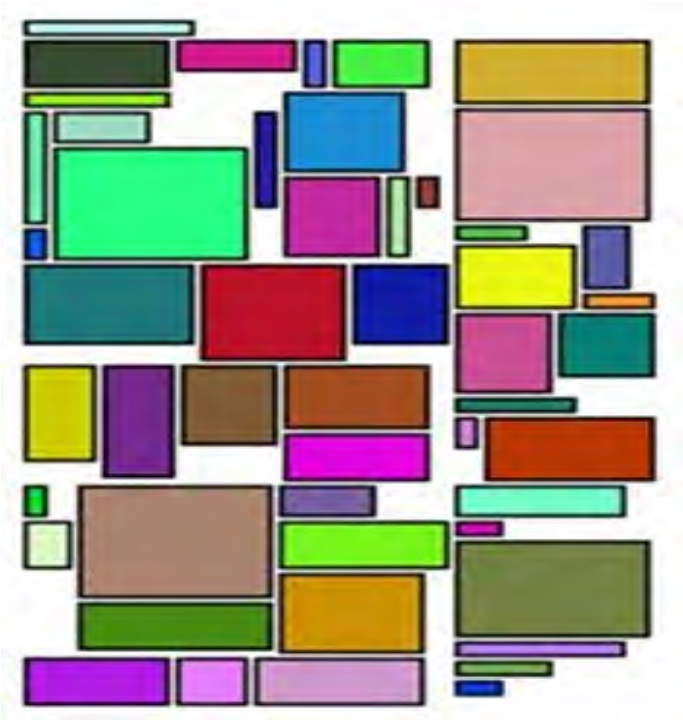


Figure 1.1: An example of VLSI floorplan

floorplan. After the implementation, if the sizes of components or of wires are significantly differed from those in the initial floorplan, then the floorplan needs to be rethought. Having a budget for blocks and wires, encourages the designers to live within their allocated areas.

The design of the initial floorplan defines the interface requirements for the blocks. Once those blocks are designed, the chip layout can be assembled from the blocks. Blocks may need to be modified due to errors in estimating the properties of the blocks during floorplanning.

In floorplanning, we are given the information of a set of modules, including their areas and interconnection. Our goal is to plan their positions on a chip to minimize the total chip area and interconnect cost. In this thesis, we address this floorplanning problem with placement constraint, i.e., besides the module information, we are also given some constraints in placement between the modules where our aim is to plan their positions on a chip such that all the placement constraints can be satisfied and the area as well as the interconnect cost are minimized.

The floorplanning problem has been tackled using various approaches: constructive approaches, iterative approaches, knowledge approaches, and mathematical programming ap-

proaches [53, 47, 52, 49, 26]. Constructive approaches start from a seed module and then select one or several modules to add to the partial floorplan. This procedure continues until all modules have been selected. Typical constructive approaches are cluster growth, partitioning and slicing, connectivity clustering, and rectangular dualization [53]. Iterative approaches search for an improved floorplan by making local changes until a feasible floorplan is gained or no more improvements can be obtained. Among different iterative approaches the popular iterative techniques are simulated annealing, force directed interchange/ relaxation and genetic algorithms [47, 52]. Apart from these, knowledge based approaches implement a knowledge expert system. Fuzzy-based floorplanning is one of the examples of knowledge based approaches. Mathematical programming approaches use linear programming, the branch-and-bound algorithm, convex programming, or nonlinear programming [49, 26].

For the VLSI floorplanning problem, the existing search methods fall into two categories, namely: 1) "local search" and 2) "global search". Generally, the local search methods are efficient. However, they may not be able to produce quality solutions as they can get stuck into local optima. Genetic Algorithm (GA) is a widely used global search method for the VLSI floorplanning problem. GAs have been successfully applied to solve slicing VLSI floorplanning problems [15, 51, 62]. For non-slicing VLSI floorplanning as well, a GA has been presented in [33].

1.2 Motivation

The placement of a set of rectangular circuit modules on a chip (minimize the total area and the total interconnecting wire length) is known as floorplan design which is one of the most important stages in the physical design of VLSI circuits. While placing circuit modules (or macro cells) many of the modules are not yet fully designed themselves and frequently have some flexibility in their shape. For example a circuit module made up from 12 identical components may have them placed in one row of 12 components, 2 rows of 6 components, 3 rows of 4 components etc., ordering the floorplan designer a range of possible shapes for that module.

Metaheuristic algorithms are used to find answers to problems when we donot know what

the optimal solution looks like, we dont know how to go about finding it in a principled way, we have very little heuristic information to go on, and brute-force search is out of the question because the space is too large. But if we are given a candidate solution to the problem, then we can test it and assess how good it is. That is, we know a good one when we see it. For example, in our problem, we have to find out a floorplan for VLSI design. We have idea to assess its quality, but we dont know about optimal result. So, we have used metaheuristic techniques to solve this problem. In this thesis we are considering non-slicing VLSI floorplanning, because slicing floorplan may not produce optimal result. Non-slicing VLSI floorplanning is NP-hard problem. So finding an algorithm which solves this problem with minimized floorplan area is a challenge.

The purpose of our thesis is to present a number of metaheuristic algorithms which appear to be very effective at non-slicing floorplans and also to minimize the total area of the floorplan. Again, there are no such software by which one can visualize the floorplan. So, after implementing our algorithms we also developed a visualization software.

1.3 Objectives

The main objectives of this thesis are:

(i) To devise a number of novel metaheuristic Algorithms for the non-slicing VLSI floorplanning problem.

(ii) To conduct extensive experiments to compare our results with other algorithms.

(iii) To conduct standard statistical tests to verify the significance of our results.

(iv) To incorporate the capability of handling mixed signal blocks.

(v) To develop a visualization software for VLSI floorplanning.

1.4 Organization of the thesis

In Chapter 2, Literature Review of our problem is discussed. The early studies on a Genetic Algorithm, Artificial Immune System (AIS) were basically on the theoretical issues. Here we have presented recent researches on AIS and Clonal Selection Algorithm.

We have discussed some preliminary concepts related to our work in Chapter 3. The concept of metaheuristics, its algorithm, representation of problems and different techniques to solve these problems are discussed. Formal definition of our problem is also discussed in this chapter. Data structure, floorplan representation, floorplan structure and their necessity in our problem is discussed here.

Chapter 4 illustrates our algorithm with a number of variants. Besides these variants we also implemented some local search techniques. All the variants and implementation of our algorithm are presented in this chapter.

Chapter 5 presents the experimental results. Section 5.1 describes some existing benchmarks which are used by us as well as by other researchers in the literature. It also illustrates the experiment data as well as environment of our research and examine the experimental results. Finally the comparison between different algorithms are presented.

We have presented the visualization software, which we made as a part of our thesis work to visualize the floorplans in Chapter 6. Here we also demonstrate all the features of our software.

Finally, Chapter 7 concludes our thesis.

Chapter 2
Literature Riview

Very Large Scale Integration (VLSI) is the process of creating an integrated circuit by combining thousands of transistors into a single chip. VLSI began in the 1970s when complex semiconductor and communication technologies were being developed. Now a days VLSI lets IC makers add lots of circuits into one chip. Floorplanning considers function, timing, size, shape for different conditions including space allocation, signal routing, power supply routing, and clock distribution. There are two different types of modules for VLSI floorplanning. These are hard module and soft module [13]. The hard module's shape is fixed, and is denoted as (W,H), where W is the width and H is the height of the module. Area is again fixed in case of soft module, but the ratio of $width/height$ is included in a given range. It can be denoted as (S,L,U), where S represents the area, L and U the power and upper boundary of the $width/height$ ratio.

Recently, clonal selection theory in the immune system has received the attention of researchers and given them inspiration to create algorithms that evolve candidate solutions by means of selection, cloning, and mutation procedures. Moreover, diversity in the population is enabled by means of the receptor editing process. The Clonal Selection Algorithm (CSA) in its canonical form and its various versions are used to solve different types of problems and are reported to perform better compared with other heuristics (i.e., genetic algorithms, neural networks, etc.) in some cases, such as function optimization and pattern recognition.

Since we will be using some metaheuristic techniques, namely, Hill-Climbing, Steepest Ascent Hill-Climbing, Random Restart Hill-Climbing with Clonal Selection Algorithm, here we briefly give a literature review on these techniques.

2.1 VLSI floorplanning

Floorplans are either slicing or non-slicing. A slicing floorplan can be obtained by repetitively cutting the floorplan horizontally or vertically, whereas a non-slicing floorplan cannot. To solve floorplan problems researchers choose either of these two structures. Lin et al. has merged the properties of encoding schemes of slicing trees and the evolutional mechanism

of genetic algorithms [41]. Lai and Wong have also used slicing floorplan in their research and presented that slicing tree could be a complete floorplan representation [40].

On the other hand, Ma et al. has presented a new efficient topology representation for non-slicing floorplan [45]. Chen et al. has proposed PSO-based intelligent decision algorithm considering non-slicing floorplan [13]. Tang and Yao have presented a memetic algorithm for VLSI floorplanning where they have considered non-slicing floorplan [56].

Our main motivation for using a non-slicing floorplan approach is to get better results than using slicing representations, and it is commonly belived. The area optimization problem for non-slicing floorplans is NP-Hard [56].

According to the data structure of VLSI floorplan, there are two types of modules in floorplanning. These are hard and soft modules. Hard module is the module whose shape is fixed. On the other hand, the shape of the module can be varied as long as the aspect ratio is within the given range with fixed area for the soft module. Tang and Yao have used hard-module for their research in [56]. On the other hand, Chang et al. used soft modules for their research [12] and Valenzuela and Wang also used the same [62].

In VLSI floorplanning, we hardly ever face a problem which is flexible in shape. For these reasons, we are motivated to use hard module in this thesis.

2.2 Related Work

Artificial Immune System (AIS) algorithms imitate the principles of immune systems. AIS is a biologically-inspired computation paradigm that emerged in recent years. Research in AIS has become popular in the late 1990s. Dasgupta published one of the early studies on AIS with theories, models, simulations, and applications [18]. Here, he gave emphasis on the natural immune system because of its powerful information processing capabilities. The immune system is a highly parallel system from an information processing perspective. It provides an excellent model of adaptive processes operating at the local level and of useful behavior emerging at the global level. Moreover, it uses learning, memory, and associative retrieval to solve recognition and classification tasks.

Other pioneering studies have reviewed the general applications of the AIS as well as ad-

dressed open issues to make the AIS a real world problem solving technique [22, 24, 10, 70]. Their work was based on immunological principles, new computational techniques, aiming not only at a better understanding of the system, but also at solving engineering problems. In their report, they discussed the main strategies used by the immune system to problem solving, and introduced the concept of immune engineering. The immune engineering made use of immunological concepts in order to create tools for solving demanding machine-learning problems using information extracted from the problems themselves. There were also some detail description of the development of several immune engineering algorithms.

Dasgupta and Nino have compared the basic principles of evolutionary algorithms (EAs) for optimization, Analytical Neural Networks (ANN) for classification, and AIS [21]. The related information have been provided using chromosome strings for EAs, connection strings for ANN, and component concentration/ network for AIS.

De Castro and Timmis have given details of the main models in AIS such as clonal selection, immune networks, and negative selection theories [10]. Timmis et al. has presented a simple example for each of the main types of AIS algorithms and then give the details of the theoretical analysis [60]. They stated that biological models of the natural immune system, in particular the theories of clonal selection, immune networks and negative selection, have provided the inspiration for AIS algorithms. Moreover, such algorithms have been successfully employed in a wide variety of different application areas. However, despite these practical successes, until recently there has been a dearth of theory to justify their use. In their paper, the existing theoretical work on AIS was reviewed. After the presentation of a simple example of each of the three main types of AIS algorithm (that is, clonal selection, immune network and negative selection algorithms respectively), details of the theoretical analysis for each of these types were given.

The immune system based models and algorithms are used to solve different types of problems. Hart and Timmis have stated that the recent applications of AIS include the areas, such as computer security, numerical function optimization, combinatorial optimization (i.e., scheduling), learning, bioinformatics, image processing, robotics (i.e., control and navigation), adaptive control systems, virus detection, and web mining [34].

Since the aim of this thesis is not outlining all the developments within AIS, the interested readers may be referred to recent review papers and books for further detailed information

which are described in detail in [29, 19, 58, 20, 21, 67]. We can provide a short summary from these papers, which is given below.

Garrett's survey has tracked the development of AIS and then attempted to make an assessment of its usefulness, defined in terms of 'distinctiveness' and 'effectiveness'. In his work, the standard types of AIS were examined, such as, negative selection, clonal selection and immune networks as well as a new breed of AIS. Dasagupta stated some advancement in AIS and proved that most of the AIS models emphasize designing artifacts. Like other biologically-inspired techniques, such as artificial neural networks, genetic algorithms, and cellular automata, AISs also try to extract ideas in order to develop computational tools for solving different problems. The AIS is emerging as an active and attractive, field involving models, techniques and applications of greater diversity.

Wong and Lau have noted the future directions of AIS algorithms as follows: exploration of novel immune theories, analyzing distinct advantages of AIS compared to other EAs, integration of AIS with other techniques, and applications of AIS algorithms to solve industrial problems [67]. Their aim was to review and outlook on the latest development of AIS-based algorithms in the recent decade. An analysis of the AIS applications is also discussed.

Timmis has stated that a large part of AIS works have been based on the clonal selection theory [58]. Therefore, their paper aimed to provide up-to-date information, specifically for CSA, as well as future research directions.

Hofmeyr and Forrest have discussed that clonal selection can be considered as genetic algorithm (GA) without crossover [35]. An artificial immune system (ARTIS) was described in their work. This work has incorporated many properties of natural immune systems, including diversity, distributed computation, error tolerance, dynamic learning and adaptation, and self monitoring. ARTIS is a general framework for a distributed adaptive system and could, in principle, be applied to many domains. In their paper, ARTIS was applied to computer security in the form of a network intrusion detection system called LISYS. LISYS was described and shown to be effective at detecting intrusions, while maintaining low false positive rates. Finally, similarities and differences between ARTIS and Holland's classifier systems were discussed.

However, De Castro and Von Zuben have stated that GA cannot cover all features of clonal selection such as affinity proportional reproduction and hyper-mutation [23]. De Castro and

Von Zuben have reviewed CSA with engineering applications and provided the details of binary character recognition, multi-modal optimization, and travelling salesperson problems (TSP) applications [24].

Brownlee has provided a detailed summary of CSA and proposed the algorithm taxonomy, a standardized nomenclature and a general model of such algorithms [9]. In addition, he discussed the similarities and differences between CSA and Evolutionary Computation (EC) as well as general research trends for CSA. Their paper provided a summary of a new field of clonal selection algorithms. Finally, the field was compared and contrasted to the field of evolutionary computation, and general research trends were discussed in their paper.

Hart and Timmis have discussed that AIS algorithms are still worth studying [34]. Recently, researchers have applied AIS in a wide spectrum of challenging problems. Furthermore, the number of studies where CSA is used with other meta-heuristics in a hybrid form is increasing. From this literature review we can have a short review of clonal selection algorithms which revealed at least three interesting findings. The first being the clear and recent popularity in the application of clonal selection algorithm derivatives, particularly from eastern (Chinese) works. The volume of works suggests the potential ease of implementation of the approach and the variety of application (primary engineering optimization and model refinement) suggests the potential generality of the approach. The second interesting trend is the hybridization of the algorithm into other algorithms and systems. In addition to using CSA as the general iterative adaptive element in other artificial immune system algorithms, the CSA principle has been grafted into a variety of other algorithms including particle swarm optimization (PSO), gene expression programming (GEP), evolutionary algorithms and various other adaptive methods. The third interesting finding is the clear similarity of the clonal selection principle with Darwin's evolution principle, and the resultant strong similarity between evolutionary algorithms (the genetic algorithm in particular) and some clonal selection algorithms.

This chapter attempted to present the field of clonal selection algorithms by presenting 1) a specific and reusable definition of clonal selection algorithms, 2) a general model for interpreting clonal selection algorithms, and 3) a general taxonomy for interpreting the current state of clonal selection algorithms research. It appears from this literature review that clonal selection algorithms are suitable for optimization domains and for classification domains. It

is important to note that CSA research is still in its infancy and these applications may be considered merely demonstrations of capability of the general method.

Finally, from an algorithm design perspective, there are likely many aspects of the clonal selection principle which have not been realized in the current state of the art clonal selection algorithms. A study of the immunological theory from a biological rather than algorithm vantage as well as associated physiology will likely reveal not only computationally interesting mechanisms and architectures, but may also suggest suitable general application domains. Some plausible areas for future clonal selection algorithm investigation may include the distributedness of the immune system physiology and the autonomy of the clonal response, the specific cellular and or genetic mechanisms employed during clonal expansion, and physiology and mechanisms of the antibody-antigen binding during the immune response.

Chapter 3
Preliminaries

In this chapter we discuss about some preliminary concepts related to our work. The concept of metaheuristic techniques and algorithm, representation of problem and different techniques to solve these problems are discussed.

3.1 Metaheuristics

To describe a major subfield, indeed the primary subfield, of stochastic optimization the term *metaheuristics* is often used. Stochastic optimization is the general class of algorithms and techniques and it employs some degree of randomness to find optimal (or nearly optimal) solutions to hard problems. Metaheuristics are applied to a very wide range of problems and are the most general of these kinds of algorithms. When we don't know what the optimal solution looks like, we don't know how to go about finding it in a principled way, we have very little heuristic information to go on, and brute-force search is out of the question because the space is too large, then these metaheuristic algorithms are used to find answers to these problems. But if we are given a candidate solution to our problem, we can test it and assess how good it is. That is, we know a good one when we see it. For example, in our problem, we have to find out a floorplan for VLSI design. We have idea to assess its quality, but we don't know about optimal result. So, we have used metaheuristic techniques to solve this problem.

3.2 Floorplanning

Floorplanning gives early feedback by thinking of layout at early stages which may suggest valuable architectural modifications. It fits very well in a top-down design strategy as well as in the stepwise refinement strategy. In modern electronic design process floorplans are created during the floorplanning design stage, which is an early stage in the hierarchical approach to chip design. Depending on the design methodology being followed, the actual definition of a floorplan may differ. In electronic design automation, a floorplan of an integrated circuit is a schematic representation of tentative placement of its major functional

blocks.

3.2.1 Admissible Floorplan

A floorplan is "admissible" if no module can be shifted left or bottom without moving other modules in the floorplan. Given a feasible floorplan (a floorplan on which no module overlaps with another), we can derive an admissible floorplan by compacting the modules to the left and bottom boundaries of the floorplan. The cost of the admissible floorplan is equal to or less than that of the original floorplan. Therefore, the search space of the VLSI floorplanning problem is limited to admissible floorplans.

3.2.2 VLSI Floorplanning

VLSI is a method by which the functionality of many different types of electronic components are set into a small space or chip. VLSI floorplanning refers to computing a floorplan for VLSI design. For VLSI design floorplanning is important. We have some search methods for VLSI floorplanning, which are described in following sections.

3.3 Different Metaheuristic Search Methods

In computer science, **local search** is a metaheuristic method for solving computationally hard optimization problems. Local search can be used on problems that can be formulated as finding a solution maximizing a criterion among a number of candidate solutions. Local search algorithms move from solution to solution in the space of candidate solutions (the search space) by applying local changes, until a solution deemed optimal is found or a time bound is elapsed.

Global optimization is a branch of applied mathematics that deals with the optimization of a function or a set of functions according to some criteria. Typically, a set of bound and more general constraints is also present, and the decision variables are optimized considering also the constraints. For global optimization we can also state stochastic optimization. **Stochastic optimization** (SO) methods are optimization methods that generate and use random variables. For stochastic problems, the random variables appear in the formulation of

the optimization problem itself, which involve random objective functions or random constraints.

The description of Local Search, Global Optimization and Population Methods have mostly been borrowed from [43] and described in the following sections.

3.4 Local Search

Local search algorithms are widely applied to numerous hard computational problems, including problems from computer science (particularly artificial intelligence, metaheuristic), mathematics, operations research, engineering, and bioinformatics. Few local search algorithms are presented in this section which have mostly been borrowed from [43].

3.4.1 Single State Method

A single-state metaheuristic uses a single candidate solution, complete or partial, at a time. This is updated on specified events, and the algorithm runs until the end condition is met. The best candidate solution is returned as optimal solution.

To optimize a candidate solution, we need to be able to do four things:

- Provide one or more initial candidate solutions. This is known as the initialization procedure.
- Assess the Quality of a candidate solution. This is known as the assessment procedure.
- Make a Copy of a candidate solution.
- Tweak a candidate solution, which produces a randomly slightly different candidate solution.

These steps are known as modification procedure. Then the metaheuristic algorithms will typically provide a selection procedure that decide which candidate solutions to retain and which to reject as it wanders through the space of possible solutions to the problem.

Hill-Climbing

Hill-Climbing is a mathematical optimization technique which belongs to the family of local search. It is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by incrementally changing a single element of the solution. If the change produces a better solution, an incremental change is made to the new solution, repeating until no further improvements can be found. The algorithm is stated in Algorithm 1.

Algorithm 1 Hill-Climbing

```
S ← some initial candidate solution
repeat
    R ← Tweak(Copy(S))
    if Quality(R) ≥ Quality(S) then
        S ← R
    end if
until S is the ideal solution or we have run out of time
return S
```

Hill climbing is good for finding a local optimum (a solution that cannot be improved by considering a neighbouring configuration) but it is not guaranteed to find the best possible solution (the global optimum) out of all possible solutions (the search space).

Steepest Ascent Hill-Climbing

Steepest Ascent Hill-Climbing is a variant of simple Hill-Climbing (HC). We make simple HC algorithm a little more aggressive by creating n "tweaks" to a candidate solution all at one time, and then adopt the best one. Because by sampling all around the original candidate solution and then picking the best, we're essentially sampling the gradient and marching straight up it which is written in Algorithm 2.

Algorithm 2 Steepest Ascent Hill-Climbing

$n \leftarrow$ number of tweaks desired to sample the gradient
$S \leftarrow$ some initial candidate solution
repeat
  $R \leftarrow Tweak(Copy(S))$
  for $n-1$ times **do**
    $W \leftarrow Tweak(Copy(S))$
    if $Quality(W) \geq Quality(R)$ **then**
      $R \leftarrow W$
    end if
  end for
  if $Quality(R) \geq Quality(S)$ **then**
    $S \leftarrow R$
  end if
until S is the ideal solution or we have run out of time
return S

Hill-Climbing with Random Restarts

Another variant of simple HC. Random Search is the extreme in exploration (and global optimization). In this algorithm, We do Hill-Climbing for a certain random amount of time. Then when time is up, we start over with a new random location and do Hill-Climbing again for a different random amount of time. The algorithm is state in Algorithm 3.

3.5 Global Optimization Algorithm

A **global optimization algorithm** can eventually find the global optimum, if we run this algorithm for a long time. Almost always, every location in the search space will be visited by a global optimization algorithm. The single-state algorithms cannot guarantee this. Because, if we're stuck in a sufficiently broad local optimum, tweak may not be strong enough to get us out of it. Thus the algorithms so far have been **local optimization algorithms**.

Algorithm 3 Hill-Climbing with Random Restarts

$T \leftarrow$ distribution of possible time intervals
$S \leftarrow$ some initial candidate solution
$Best \leftarrow S$
repeat
  $time \leftarrow$ random time in the near future, chosen from T
  repeat
    $R \leftarrow Tweak(Copy(S))$
    if $Quality(R) \geq Quality(S)$ **then**
      $S \leftarrow R$
    end if
  until S is the ideal solution or we have run out of time
  if $Quality(S) \geq Quality(Best)$ **then**
    $Best \leftarrow S$
  end if
  $S \leftarrow$ some random candidate solution
until *Best* is the ideal solution or we have run out of time
return *Best*

3.6 Population Methods

Population-based methods differ from the previous methods. They keep around a sample of candidate solutions rather than a single candidate solution. Each of the solutions is involved in tweaking and quality assessment. Most population-based methods are inspired by the concepts from biology. One particularly popular set of techniques, collectively known as **Evolutionary Computation (EC)**, borrows liberally from population biology, genetics, and evolution. An algorithm chosen from this collection is known as an **Evolutionary Algorithm (EA)**. Most EAs may be divided into **generational** algorithms, which update the entire sample once per iteration, and **steady-state** algorithms, which update the sample a few candidate solutions at a time. Common EAs include the **Genetic Algorithm (GA)** and **Evolution Strategies (ES)**. There are both generational and steady-state versions of each.

The basic generational evolutionary computation algorithm first constructs an initial population, then iterates through three procedures. First, it **assesses the fitness** of all the individuals in the population. Second, it uses this fitness information to **breed** a new population of children. Third, it **joins** the parents and children in some fashion to form a new next-generation population, and the cycle continues. Algorithm 4 is an Evolutionary algorithm.

Algorithm 4 An Abstract Generational Evolutionary Algorithm (EA)

$P \leftarrow$ Build Initial Population
$Best \leftarrow \square$
repeat
  AssessFitness(P)
  for each individual $P_i \in P$ **do**
    if $Best = \square$ **or** $Fitness(P_i) > Fitness(Best)$ **then**
      $Best \leftarrow P_i$
    end if
  end for
  $P \leftarrow Join(P, Breed(P))$
until *Best* is the ideal solution or we have run out of time
return *Best*

3.6.1 Evolution Strategies

Evolution Strategies (ES) were developed by Ingo Rechenberg and Hans-Paul Schwefel at the Technical University of Berlin in the mid 1960s. ES employ a simple procedure for selecting individuals called Truncation Selection. It only uses mutation as the Tweak operator. The (μ, λ) algorithm is one of the simplest ES algorithm. The procedure begins with a population of λ number of individuals, which is generated randomly. Then it iterates as follows. First it assesses the fitness of all the individuals. Then it deletes from the population all but the μ fittest ones. Each of the μ fittest individuals gets to produce λ/μ children through an ordinary Mutation. Then it creates λ new children. Its Join operation is simple: the children just replace the parents, who are discarded. The iteration continues. In short, μ is the number of parents which survive, and λ is the number of kids that the μ parents make in total. λ should be a multiple of μ. ES practitioners usually refer to their algorithm by the choice of μ and λ. For example, if $\mu = 5$ and $\lambda = 20$, then we have a $(5, 20)$ Evolution Strategy.

3.6.2 The Genetic Algorithm

The **Genetic Algorithm (GA)** was invented by John Holland at the University of Michigan in the 1970s [36]. It is similar to a (μ, λ) Evolution Strategy in many respects. For example, it iterates through fitness assessment, selection and breeding, and population reassembly. To breed, it begins with an empty population of children. Then it selects two parents from the original population, then copies, crosses them over with one another, and mutates the results. This forms two children, which is added to the child population. It repeats this process until the child population is entirely filled. The algorithm in pseudocode is presented in Algorithm 5.

3.7 Artificial Immune System (AIS)

The following description of AIS has mostly been borrowed from [5]. Nature has acted as inspiration for many aspects of computer science. A trivial example of this is the use of trees as a metaphor, consisting of branched structures, with leaves, nodes and roots. Of

Algorithm 5 The Genetic Algorithm (GA)

$popsize \leftarrow$ desired population size

$P \leftarrow \{\}$

for popsize times **do**

$P \leftarrow P \bigcup \{$ new random individual $\}$

end for

$Best \leftarrow \square$

repeat

for each individual $P_i \in P$ **do**

$AssessFitness(P_i)$

if $Best = \square$ **or** $Fitness(P_i) > Fitness(Best)$ **then**

$Best \leftarrow P_i$

end if

end for

$Q \leftarrow \{\}$

for $popsize/2$ times **do**

Parent $P_a \leftarrow$ SelectWithReplacement(P)

Parent $P_b \leftarrow$ SelectWithReplacement(P)

Children $C_a, C_b \leftarrow Crossover(Copy(P_a), Copy(P_b))$

$Q \leftarrow Q \bigcup \{Mutate(C_a), Mutate(C_b)\}$

end for

$P \leftarrow Q$

until *Best* is the ideal solution or we have run out of time

return *Best*

course, a tree structure is not a simulation of a tree, but it abstracts the principal concepts to assist in the creation of useful computing systems. Bio-inspired algorithms and techniques are developed not as a means of simulation, but because they have been inspired by the key properties of the underlying metaphor. The algorithms attempt to improve computational techniques by mimicking (to some extent) successful natural phenomena, with the goal of achieving similar desirable properties as the natural system.

The major part of AIS work to date has been the development of three algorithms derived from more simplified models; negative selection, clonal selection and immune networks. However, these first generation AIS algorithms have often shown considerable limitations when applied to realistic applications. For this reason, a second generation of AIS is emerging, using models derived from cutting-edge immunology as their basis, not simply mechanisms derived from basic models.

3.7.1 Biological Overview

The biological immune system is an elaborate defense system which has evolved over millions of years. While many details of the immune mechanisms (innate and adaptive) and processes (humeral and cellular) are yet unknown (even to immunologists), it is, however, well-known that the immune system uses multilevel (and overlapping) defense both in parallel and sequential fashion, shown in Figure 3.1. Depending on the type of the pathogen, and the way it gets into the body, the immune system uses different response mechanisms (differential pathways) either to neutralize the pathogenic effect or to destroy the infected cells. A detailed overview of the immune system can be found in many textbooks, for instance Kubi (2002). The immune features that are particularly relevant to our tutorial are matching, diversity and distributed control. Matching refers to the binding between antibodies and antigens. Diversity refers to the fact that, in order to achieve optimal antigen space coverage, antibody diversity must be encouraged according to Hightower et al (1995). Distributed control means that there is no central controller; rather, the immune system is governed by local interactions among immune cells and antigens. Two of the most important-cells in this process are white blood cells, called T-cells, and B-cells. Both of these originate in the bone marrow, but T-cells pass on to the thymus to mature, before they circulate the body in the blood and lymphatic vessels. The T-cells are of three types; T helper cells which are

essential to the activation of B-cells, Killer T-cells which bind to foreign invaders and inject poisonous chemicals into them causing their destruction, and suppressor T-cells which inhibit the action of other immune cells thus preventing allergic reactions and autoimmune diseases. B-cells are responsible for the production and secretion of antibodies, which are specific proteins that bind to the antigen. Each B-cell can only produce one particular antibody. The antigen is found on the surface of the invading organism and the binding of an antibody to the antigen is a signal.

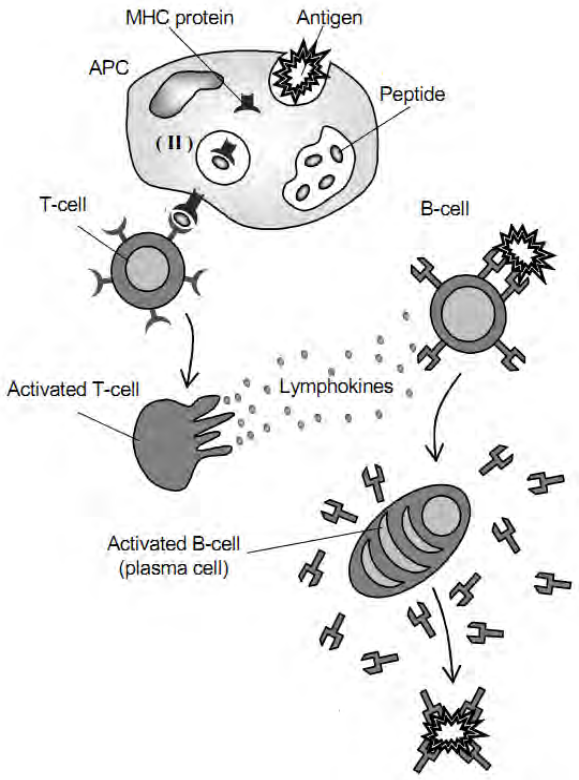


Figure 3.1: Biological Overview of Artificial Immune System. Figure borrowed from [5]

As mentioned above, the human body is protected against foreign invaders by a multi-layered system. The immune system is composed of physical barriers such as the skin and respiratory system; physiological barriers such as destructive enzymes and stomach acids; and the immune system, which has can be broadly divided under two heads - Innate (nonspecific) Immunity and Adaptive (specific) Immunity, which are inter-linked and influence each other. The Adaptive Immunity again is subdivided under two heads - Humoral Immunity and Cell Mediated Immunity. [5]

3.7.2 Terminology and Definition

Innate Immunity: The Innate Immunity is present at birth. Physiological conditions such as pH, temperature and chemical mediators provide inappropriate living conditions for foreign organisms. Also micro-organisms are coated with antibodies and/or complement products (opsonization) so that they are easily recognized. Extracellular material is then ingested by macrophages by a process called phagocytosis. Also TDH Cells influences the phagocytosis of macrophages by secreting certain chemical messengers called lymphokines. The low levels of sialic acid on foreign antigenic surfaces make C_{3b} bind to these surfaces for a long time and thus activate alternative pathways. Thus MAC is formed, which puncture the cell surfaces and kill the foreign invader.

Adaptive Immunity

Adaptive Immunity is the main focus of interest here as learning, adaptability, and memory are important characteristics of Adaptive Immunity. It is subdivided under two heads - Humoral Immunity and Cell Mediated Immunity.

Humoral Immunity

Humoral immunity is mediated by antibodies contained in body fluids (known as humors). The humoral branch of the immune system involves interaction of B cells with antigen and their subsequent proliferation and differentiation into antibody-secreting plasma cells. Antibody functions as the effectors of the humoral response by binding to antigen and facilitating its elimination. When an antigen is coated with antibody, it can be eliminated in several ways. For example, antibody can cross-link the antigen, forming clusters that are more readily ingested by phagocytic cells. Binding of antibody to antigen on a microorganism also can activate the complement system, resulting in lysis of the foreign organism.

Cellular Immunity

Cellular immunity is cell-mediated; effector T cells generated in response to antigen are responsible for cell-mediated immunity. Cytotoxic T lymphocytes (CTLs) participate in

cell-mediated immune reactions by killing altered self-cells; they play an important role in the killing of virus-infected cells and tumor cells. Cytokines secreted by TDH can mediate the cellular immunity, and activate various phagocytic cells, enabling them to phagocytose and kill microorganisms more effectively. This type of cell-mediated immune response is especially important in host defense against intracellular bacteria and protozoa.

Antibody

A feature vector coupled with its associated output or class; the feature vector-output combination is referred to as an antibody when it is part of an memory cell .

Antigen

This is the same in representation as an antibody; however, the feature vector-class combination is referred to as an antigen when it is being presented to the ARBS for stimulation and/or response.

B-cells

B-cells are responsible for the production and secretion of antibodies, which are specific proteins that bind to the antigen. Each B-cell can only produce one particular antibody. The antigen is found on the surface of the invading organism and the binding of an antibody to the antigen is a signal to destroy the invading cell.

T-cells

T-cells are of three types; T helper cells which are essential to the activation of B-cells, Killer T-cells which bind to foreign invaders and inject poisonous chemicals into them causing their destruction, and suppressor T-cells which inhibit the action of other immune cells thus preventing allergic reactions and autoimmune diseases.

Initialization of Population

To implement a basic Artificial Immune System, four decisions have to be made: Encoding, Similarity Measure, Selection and Mutation. Once an encoding has been fixed and a suitable similarity measure is chosen, the algorithm will then perform selection and mutation, both based on the similarity measure, until stopping criteria are met. In this section, we will describe each of these components in turn. Along with other heuristics, choosing a suitable encoding is very important for the algorithm's success. Similar to Genetic Algorithms, there is close inter-play between the encoding and the fitness function (the later is in Artificial Immune Systems referred to as the 'matching' or 'affinity' function). Hence both ought to be thought about at the same time. For the current discussion, let us start with the encoding.

First, let us define what we mean by 'antigen' and 'antibody' in the context of an application domain. Typically, an antigen is the target or solution, e.g. the data item we need to check to see if it is an intrusion, or the user that we need to cluster or make a recommendation for. The antibodies are the remainder of the data, e.g. other users in the data base, a set of network traffic that has already been identified etc. Sometimes, there can be more than one antigen at a time and there are usually a large number of antibodies present simultaneously [5, pp. 26].

Antigens and antibodies are represented or encoded in the same way. For most problems the most obvious representation is a string of numbers or features, where the length is the number of variables, the position is the variable identifier and the value (could be binary or real) of the variable.

Affinity

A measure of "closeness" or similarity between two anti-bodies or antigens. In the current implementation, this value is guaranteed to be between 0 and 1 inclusively and is calculated simply as the Euclidean distance of the two objects' feature vectors. Thus, small affinity values indicate strong affinity.

Somatic Hyper Mutation

The mutation most commonly used in Artificial Immune Systems is very similar to that found in Genetic Algorithms, e.g. for binary strings bits are flipped, for real value strings one value is changed at random, or for others the order of elements is swapped. In addition, the mechanism is often enhanced by the 'somatic' idea, i.e., the closer the match (or the less close the match, depending on what we are trying to achieve), the more (or less) disruptive the mutation.

However, mutating the data might not make sense for all problems considered. For instance, it would not be suitable for the movie recommender. Certainly, mutation could be used to make users more similar to the target, however, the validity of recommendations based on these artificial users is questionable and if over-done, we would end up with the target user itself. Hence for some problems, somatic hypermutation is not used, since it is not immediately obvious how to mutate the data sensibly such that these artificial entities still represent plausible data. Nevertheless, for other problem domains, mutation might be very useful. For instance, taking the negative selection approach to intrusion detection, rather than throwing away matching detectors in the first phase of the algorithm, these could be mutated to safe time and effort. Also, depending on the degree of matching the mutation could be more or less strong. This was in fact one extension implemented by Hofmeyr and Forrest.

Candidate Memory cells

The antibody of the same class as the training antigen, which was the most stimulated after exposure to the given antigen.

Seed cells

It is an antibody , drawn from the training set , used to initialize Memory Cell.

3.7.3 Clonal Selection Algorithm

Clonal selection theory is used to explain basic response of adaptive immune system to antigenic stimulus. It establishes the idea that only those cells capable of recognizing an antigen will proliferate while other cells are selected against. Clonal selection operates on both B and T cells. B cells, when their antibodies bind with an antigen, are activated and differentiated into plasma or memory cells. Prior to this process, clones of B cells are produced and undergo somatic hyper mutation. As a result, diversity is introduced into the B cell population. Plasma cells produce antigen-specific antibodies that are work against antigen. Memory cells remain with the host and promote a rapid secondary response [6].

3.7.4 Negative Selection Algorithm

Negative selection is a mechanism to protect body against self-reactive lymphocytes. It utilizes the immune system′s ability to detect unknown antigens while not reacting to the self cells. During the generation of T-cells, receptors are made through a pseudo-random genetic rearrangement process. Then, they undergo a censoring process in the thymus, called the negative selection. In this process, T-cells that react against self-proteins are destroyed and only those that do not bind to self-proteins are allowed to leave the thymus. These matured T-cells then circulate throughout the body performing immunological functions and protecting the body against foreign antigens [6].

3.7.5 Immune Networks Theory

The main idea of immune networks theory was that the immune system maintains an idiotypic network of interconnected B cells for antigen ecognition. These cells interconnect with each other in certain ways that lead to the stabilization of the network. Two B cells are connected if the affinities they share exceed a certain threshold, and the strength of the connection is directly proportional to the affinity they share [6].

3.8 Clonal Selection Algorithm

In artificial immune systems, clonal selection algorithms are a class of algorithms which are inspired by the clonal selection theory of acquired immunity that explains how B and T lymphocytes improve their response to antigens over time called affinity maturation. These algorithms focus on the Darwinian attributes of the theory where selection is inspired by the affinity of antigen-antibody interactions, reproduction is inspired by cell division, and variation is inspired by somatic hypermutation. Clonal selection algorithms are most commonly applied to optimization and pattern recognition domains, some of which resemble parallel hill climbing and the genetic algorithm without the recombination operator.

3.8.1 Theory

When an animal is exposed to an antigen, some sub population of its bone marrow derived cells (B lymphocytes) respond by producing antibodies (Ab). Each cell secretes a single type of antibody, which is relatively specific for the antigen. By binding to these antibodies (cell receptors), and with a second signal from accessory cells, such as the T-helper cell, the antigen stimulates the B cell to proliferate (divide) and mature into terminal (non-dividing) antibody secreting cells, called plasma cells. The process of cell division (mitosis) generates a clone, i.e., a cell or set of cells that are the progenies of a single cell. While plasma cells are the most active antibody secretors, large B lymphocytes, which divide rapidly, also secrete antibodies, albeit at a lower rate [25]. On the other hand, T cells play a central role in the regulation of the B cell response and are preeminent in cell mediated immune responses, but will not be explicitly accounted for the development of our model. Lymphocytes, in addition to proliferating and/or differentiating into plasma cells, can differentiate into long-lived B memory cells. Memory cells circulate through the blood, lymph and tissues, and when exposed to a second antigenic stimulus commence to differentiate into large lymphocytes capable of producing high affinity antibodies, pre-selected for the specific antigen that had stimulated the primary response. Figure 3.2 depicts the clonal selection principle.

The main features of the clonal selection theory that will be explored in this thesis are :

- Proliferation and differentiation on stimulation of cells with antigens;

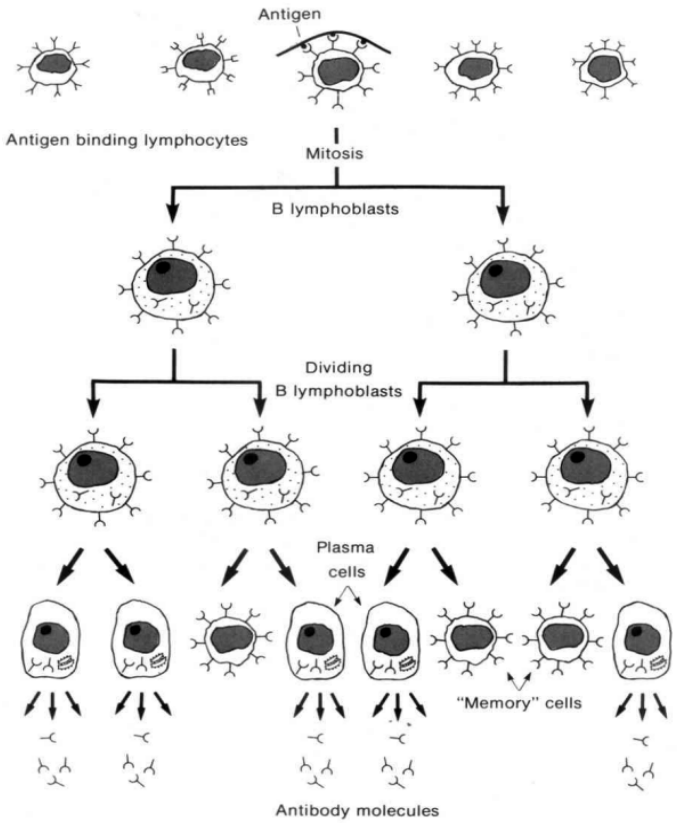


Figure 3.2: A simple overview of the clonal selection process. Figure borrowed from [9]

- Generation of new random genetic changes, subsequently expressed as diverse antibody patterns, by a form of accelerated somatic mutation (a process called affinity maturation);
- Elimination of newly differentiated lymphocytes carrying low affinity antigenic receptors.

The immune system consists of two main parts, innate immunity which consists of basic elements the organism is born with such as skin and natural barriers, and acquired immunity located only in vertebrates (organisms with a spinal column/ backbone). Acquired or learned immunity supplements innate immunity and is acquired through contact with antigens (something that evokes an immune response). The following lists the primary features of acquired immunity taken from :

Specificity: The systems able to discriminate between different molecular entities and specialise.

Adaptiveness: The systems ability to respond to previously unseen molecules, even those molecules that previously never existed on earth (man made).

Discrimination between self and non-self: This critical element of the immune system indicates that the system is capable of recognising and responding to foreign elements, though is able to tell the difference between what is foreign and potentially harmful, and what is actually apart of its own system. Without this discrimination, the immune system would respond against the organism for which it exists to protect.

Memory: The systems ability to recall previous contact with foreign molecules and respond in a learned manner. This means that a larger and more rapid response occurs on subsequent encounters with an antigen, this effect is called an anamnestic response.

3.9 Representation

The **representation** of an individual is the approach we take to constructing, tweaking, and presenting the individual for fitness assessment. Although often we shall refer to the representation as the data structure used to define the individual (a vector, a tree, etc.) it is useful to think of the representation not as the data type but instead simply as two functions:

- The **initialization** function used to generate a random individual.
- The Tweak function, which takes one individual (or more) and slightly modifies it. To this we might add more:
- The **fitness assessment** function.
- The Copy function.

These functions are the only places where many optimization algorithms deal with the internals of individuals. Otherwise the algorithms treat individuals as black boxes. By handling these functions specially, we can separate the entire concept of representation from the system. Much of the success or failure of a metaheuristic lies in the design of the representation of the individuals, because their representation, and particularly how they Tweak, has such a strong impact on the trajectory of the optimization procedure as it marches through the fitness landscape (that is, the quality function).

Most popular representations are as follows:

- Vector
- Direct Encoded Graphs
- Trees
- Genetic Programming

In our work we have used O-Tree representation. So, here we are going to elaborate O-Tree representation.

3.9.1 O-Tree Representation

An ordered tree (O-tree) is an oriented tree in which the children of a node are somehow "ordered". This tree contains finite set of one or more nodes. There is one special designated node called the root of the tree. The root has zero or more branches. The branches are directed edges pointed from the root to the children.

One of the most efficient nonslicing representations is the O-tree representation. The representation not only covers all optimal floorplans but also has a smaller search space. This representation gives geometrical relations among modules, which is valuable for identifying meaningful building blocks when designing genetic operators of evolutionary algorithms. Hence, the O-tree representation is adopted in this thesis.

Representation of floorplan using O-tree

A floorplan with n rectangular modules can be represented in a horizontal (vertical) O-tree of $(n+1)$ nodes, of which n nodes correspond to n modules $m_1, m_2, ..., m_n$ and one node corresponds to the left (bottom) boundary of the floorplan [32]. The left (bottom) boundary is a dummy module with zero width (height) placed at $x = 0 (y = 0)$. In a horizontal O-tree, there exists a directed edge from module m_i to module m_j if and only if $x_j = x_i + w_i$, where x_i is the x coordinate of the left bottom position of m_i, x_j is the x coordinate of the left-bottom position of m_j , and w_i is the width of m_i. In a vertical O-tree, there exists a directed edge from module m_i to module m_j if and only if $y_j = y_i + h_i$, where y_i is the y coordinate of the left-bottom position of m_i, y_j is the y coordinate of the left bottom position

of m_j , and h_i is the height of m_i. Figure 3.3 (a) shows a floorplan and its horizontal O-tree representation. An O-tree can be encoded in a tuple (T, π), where T is a $2n-$bit string specifying the structure of the O-tree, and π is a permutation of the nodes. For a horizontal O-tree, the tuple is obtained by depth-first traversing the nodes and edges of the O-tree. When visiting a node other than the root, we append the node to π. When visiting an edge in descending direction, we append a 0 to T, and when visiting an edge in ascending direction, we append a 1 to T. The horizontal O-tree shown in Figure 3.3 (a) is encoded into $(00110100011011, adbcegf)$. The idea of the encoding is illustrated in Figure 3.3 (b). We can use the same idea to encode a vertical O-tree as it was used in [56].

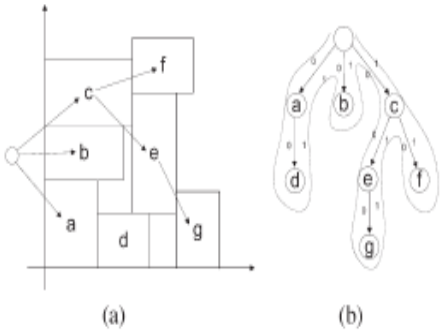


Figure 3.3: Illustration of (a) O-tree representation and (b) encoding

3.10 Problem Statement

VLSI is the process of creating an integrated circuit (IC) by combining thousands of transistors into a single chip, where an IC is a set of electronic circuits on one small plate of semiconductor material. In electronic design automation, a floorplan of an integrated circuit is a schematic representation of tentative placement of its major functional blocks. So, for VLSI design, floorplanning is very important. In VLSI floorplanning problem, a number of modules are given and these modules are placed such a way that no two modules overlap each other.

3.10.1 Problem Description

Given a set of circuit components, or "modules" and a net list specifying interconnections between the modules, the goal of VLSI floorplanning is to find a floorplan for the modules

such that no module overlaps with another and the area of the floorplan and the interconnections between the modules are minimized. In Figure 3.4, an example of a VLSI Floorplan is given.

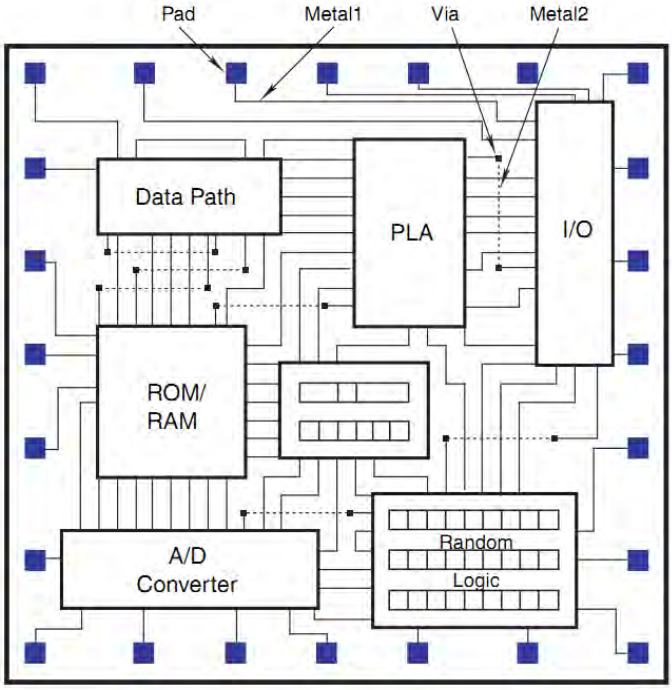


Figure 3.4: An example of a VLSI Floorplan

3.10.2 Formal Definition

Let the set of blocks be $B = b_1, b_2 ... b_N$ where N is a number of modules. Each block b_i, where $1 <= i <= N$, is rectangular in shape with width W_i and height H_i. The modules are placed in such a way that no two modules overlap with each other.

So, the input of this problem is n number of blocks with their widths and heights. Output of this problem is defined by a pair of number containing two coordinates for each block. Finally, the constraint of the problem is to place all the blocks such a way that no two blocks overlap each other.

Our objective is to minimize the floorplan area. On the other hand, the space in the floorplan, which is not covered by any module, is called white space or dead space. So, to minimize the total area in the floorplan, we have to minimize the dead space.

3.11 The Data Structure

Let the set of blocks be $B = b_1, b_2 ... b_N$ where N is a number of modules. Each block b_i, where $1 \leq i \leq N$, is rectangular in shape with width W_i and height H_i. The modules are placed in such a way that no two modules overlap with each other. There are two types of modules in floorplanning.

Hard module is the module whose shape (i.e., width and height) is fixed, and is denoted as (W, H) where W is the width and H is the height of the module. On the other hand, width (W) and height (H) of the module can be varied as long as the aspect ratio is within the given range but the area (A) is fixed for the **Soft module**.

Unlike a hard module that has a fixed dimension (width and height); the shape of a soft module is to be decided during floorplanning, although its area is fixed. Therefore, a floorplanner needs to find a desired aspect ratio for each soft module to optimize the floorplan cost.

Soft modules are used to fill up the unused area by changing the shapes and dimensions of the modules, while keeping the relative positions between the modules as described by the constraint graphs unchanged.

A hard module is free to move and rotate. The height and width are given, and it has no flexibility in shape. Most of the practical VLSI problem is like this, where we cannot have the flexibility in shape and also we have to optimize the area. On the other hand, we hardly ever face a problem which is flexible in shape. For these reasons, we are motivated to use hard module in this thesis.

3.12 Floorplan Structure

Floorplans can be either slicing or non-slicing. A slicing floorplan can be obtained by repetitively cutting the floorplan horizontally or vertically, whereas a non-slicing floorplan cannot.

In the non-slicing floorplan there is no requirement for recursive construction, and tighter packings are often possible using this approach. On the other hand, slicing floorplan can be obtained by recursively dividing a rectangle into two parts with either a vertical or a

horizontal line, it is possible to fully exploit the available flexibility in the circuit modules and efficiently combine module placement and area optimization into a single algorithm.

Our main motivation for using a non-slicing floorplan approach is to get better results than using slicing representations, and it is commonly belived. The area optimization problem for non-slicing floorplans is NP-Hard [56].

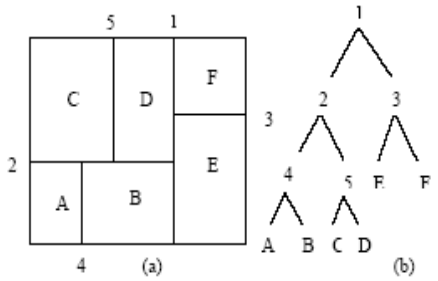


Figure 3.5: Slicing floorplan and its slicing tree (a) Slicing floorplan (b) Slicing tree

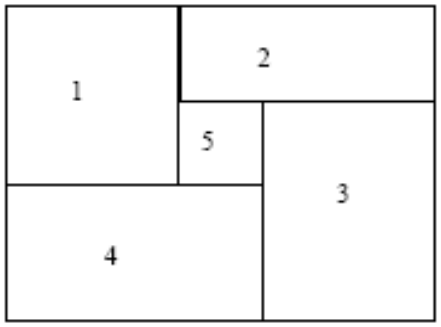


Figure 3.6: Non-slicing floorplan

In Figure 3.5: (a) letter denotes modules and number denotes horizontal and vertical cut division. Figure 3.6 shows a non-slicing floorplan and the number in the Figure 3.6 denotes the number of modules in the floorplan. In this thesis we have considered non-slicing based floorplan representation.

3.13 Analysis techniques

3.13.1 Statistical Test

In this test, we find the best solution from a number of candidate solutions, calculate the mean from the number of solutions and also calculate the standard deviation of the solutions.

This test is important, because from this test we can provide the best result as well as we can understand the deviations between the solutions.

3.13.2 Kruskal-Wallis Test

We use the *Kruskal-Wallis test* when we have one nominal variable and one measurement variable that is severely non-normal, or when we have one nominal variable and one ranked variable. It tests whether the mean ranks of the measurement variable are the same in all the groups. A nominal variable is a variable with values which have no numerical value, such as gender or occupation [46].

It is a non-parametric test. That is, it does not assume that the data coming from a distribution can be completely described by two parameters (mean and standard deviation). Like most non-parametric tests, we perform it on ranked data, so we convert the measurement observations to their ranks in the overall data set, where the smallest value gets a rank of 1, the next smallest gets a rank of 2, and so on.

If our original data set actually consists of one nominal variable and one ranked variable, we must use Kruskal-Wallis test [46]. But besides these, we can use this test for any analysis.

While working with a measurement variable, the Kruskal-Wallis test starts by substituting the rank in the overall data set for each measurement value. The smallest value gets a rank of 1, the second-smallest gets a rank of 2, etc. Tied observations get average ranks. For example, we have 4 observations with values 2, 4, 4 and 8 respectively. Hence, their rank will be 1, 2.5, 2.5 and 4 respectively. After that we calculate sum of ranks for each groups, and the test statistic. Test statistic is calculated by a rather formidable formula that basically represents the variance of the ranks among groups, with an adjustment for the number of ties. Thus, we can compare two groups by their mean rank.

This chapter has described preliminary concepts of metaheuristic, floorplan, data structures, algorithms, etc. There are problems, for which, we have very little heuristic information to go on as well as search space is too large to apply brute-force search. Thus, different

types of metaheuristic algorithms and methods are discussed here. Representation and data structures of a problem are very important to solve that problem. These are also described in this chapter. Finally, analysis plays a key role to develop new concepts. Hence, we have also discussed few analysis techniques.

Chapter 4
Our Algorithms

The purpose of our thesis is to present a number of metaheuristic algorithms which appear to be very effective at solving the floorplanning problem and also to minimize the total area (or minimize the dead space) of the floorplan.

Our algorithms use an effective method (i.e., CSA) with mutation operator to explore the search space and capitalizes on the single state methods introduced in the Section 3.4.1 to exploit information in the search region. In Chapter 3 we describe these algorithms in general. In this chapter, we are going to present these algorithms to solve VLSI floorplanning problem.

Additionally, we present CSA with mutation operator and apply a number of single state methods separately to solve VLSI floorplanning problem. Moreover, we present all the single state methods separately in order to solving the problem.

4.1 Initial population

In metaheuristic, there are either single solution searches or population based searches. Single solution approaches focus on modifying and improving a single candidate solution. On the other hand, population based searches have a collection of candidate solutions. A number of individuals are there in the initial population. An individual is an O-tree (I, π) representing an admissible VLSI floorplan F. The algorithm starts with randomly generating a sequence of modules π. Then, it inserts the modules into an initially empty O-tree I in the randomly generated order. When inserting a module into I, it checks all external insertion positions for the module and inserts the module at the position that gives the best fitness. Finally, after checking the fitness of the individual, our algorithm decides whether it will include this individual in this population or not. This algorithm is invoked iteratively to generate an initial population of individuals. Initial population algorithm for VLSI floorplanning problem is stated in Algorithm 6.

Algorithm 6 InitialPopulation($Modules$)

1: $best \leftarrow \square$
2: $P \leftarrow \{\}$
3: **for** 10 times **do**
4: $M \leftarrow Modules$
5: **repeat**
6: $m \leftarrow$ randomly select a module from M
7: place m into an appropriate position of floorplan I
8: remove m from M
9: **until** M is empty
10: **if** $FloorPlanArea(I) < FloorPlanArea(best)$ **then**
11: $best \leftarrow I$
12: **end if**
13: $P \leftarrow P \cup \{I\}$
14: **end for**
15: **return** $best$

4.2 Single State Metaheuristics Algorithms for the VLSI Floorplanning Problem

In this section we are going to discuss how we have used single state metaheuristic algorithms to exploit the result for our VLSI floorplanning problem.

For these single state metaheuristic algorithms, our algorithm generates some candidate solutions. Then these algorithms are applied separately to these candidate solutions and *best* solution is calculated. The process is repeated until a preset runtime is up.

4.2.1 Hill-Climbing

According to O-tree representation, the blocks are placed either as sibling or as child of the current node. So there may be lots of empty positions which are available in the plan and costs some dead spaces. But there may also remain some blocks which are placed at the outer boundary (Top most/ right most), which can be placed in those empty positions. By

doing this, it can minimize the total area of the floorplan. In Hill-Climbing algorithm, we have searched for this scenario and assess the quality of the current solution by calculating the area of the floorplan. If it becomes better than *best*, then we replace *best* by current solution. Algorithm 7 presents Hill-Climbing for VLSI floorplanning problem.

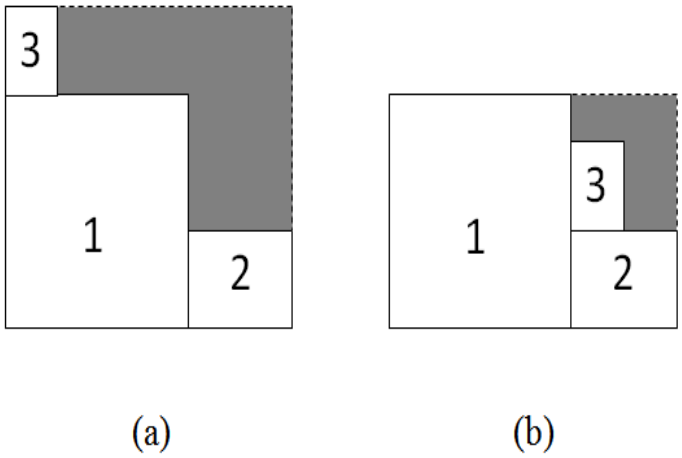


Figure 4.1: VLSI floorplan (a) before applying HC and (b) after applying HC

Figure 4.1(a) shows a sample floorplan before applying Hill-Climbing algorithm and Figure 4.1(b) shows the improved floorplan after applying Hill-climbing algorithm. From this figure, we have seen that, Hill-Climbing algorithm can reduce the dead space of the floorplan.

4.2.2 Steepest Ascent Hill-Climbing

It is a variant of simple Hill-Climbing algorithm. In simple hill climbing, a node from the search space is chosen. On the other hand, this modified algorithm is called Steepest Ascent Hill-Climbing, because by sampling all around the original candidate solution and then picking the best, we're essentially sampling the gradient and marching straight up it.

So, in this algorithm we tried to utilize all the available empty positions by shifting blocks from the outer boundary. For a particular solution, we tried to find 10 other solutions which are closer to the current solution. Then we compare each solution with the current solution, and update the *best* solution accordingly. Algorithm 8 presents Steepest Ascent Hill-Climbing for VLSI floorplanning Problem.

Algorithm 7 Hill-Climbing($best$, I)

1: **repeat**
2: $R \leftarrow Copy(I)$
3: $m \leftarrow$ select a module from R who has the highest width or height
4: **for** each available pisition (x, y) in floorplan I **do**
5: place m in (x, y)
6: **if** $FloorPlanArea(R) < FloorPlanArea(I)$ **then**
7: $I \leftarrow R$
8: **end if**
9: **if** $FloorPlanArea(R) < FloorPlanArea(best)$ **then**
10: $best \leftarrow R$
11: **end if**
12: **end for**
13: **until** *best* is an ideal solution or we have run out of time
14: **return** *best*

4.2.3 Random Restart Hill-Climbing

Random-restart hill climbing is a meta-algorithm built on top of the hill climbing algorithm. In this algorithm we applied Hill-Climbing and get best solution i.e., *best*, and after a time interval it starts from a new solution. It compares the new candidate solutions with the existing *best* and update accordingly.

For VLSI floorplanning problem we have used this metaheuristic. We have fixed a random time in near future t. During this time interval this algorithm mainly applies Hill-Climb to a solution. After this period it updates the *best* solution and again repeats the whole process until it is an ideal solution or we have run out of time. Algorithm 9 presents Hill-Climbing with Random Restarts for VLSI Floorplanning Problem.

4.3 Clonal Selection Algorithm for VLSI floorplanning Problem

We have used clonal selection algorithm with mutation operator for VLSI floorplanning problem to explore the search space. From the current population we select an individual

Algorithm 8 SteepestAscentHill-Climbing($best$, I)

1: **repeat**
2: $R \leftarrow Copy(I)$
3: **for** 10 times **do**
4: $W \leftarrow Copy(I)$
5: $T \leftarrow$ select a module from W who has the highest width or height
6: **for** each available pisition (x, y) in floorplan I **do**
7: place T in (x, y)
8: **if** $FloorPlanArea(W) < FloorPlanArea(R)$ **then**
9: $R \leftarrow W$
10: **end if**
11: **end for**
12: **if** $FloorPlanArea(R) < FloorPlanArea(I)$ **then**
13: $I \leftarrow R$
14: **end if**
15: **if** $FloorPlanArea(R) < FloorPlanArea(best)$ **then**
16: $best \leftarrow R$
17: **end if**
18: **end for**
19: **until** $best$ is an ideal solution or we have run out of time
20: **return** $best$

Algorithm 9 RandomRestartHill-Climbing($best$, I)

repeat
  $t \leftarrow$ random time in near future
  repeat
    $R \leftarrow Copy(I)$
    $m \leftarrow$ select a module from R who has the highest width or height
    for each available pisition (x, y) in floorplan I **do**
      place m in (x, y)
      if $FloorPlanArea(R) < FloorPlanArea(I)$ **then**
        $I \leftarrow R$
      end if
    end for
  until I is an ideal solution or we have run out of time
  if $FloorPlanArea(I) < FloorPlanArea(best)$ **then**
    $best \leftarrow I$
  end if
until $best$ is an ideal solution or we have run out of time
return $best$

to make a clone of that individual. Then we applied mutation to that clone and assess the fitness of the solution. If it is better than the individual from which it is cloned, then this individual is replaced by this clone. Again if the clone is better than *best*, it also replaces the *best*. We performed this for ten generations.

4.3.1 Mutation

In this section, we discuss the mutation operation we have applied in our CSA algorithm. After generating the clones from the population, mutation is applied to those clones. Then our algorithms check whether there exists any individual in the population, which is not better than the individuals of clones. If such individual from the population is found, then that individual is replaced by that individual from the clones. In general, the mutation operator identifies the top-level subtrees of the O-tree and randomly changes the order of the subtrees which is illustrated in Figure 4.2, in which Figure 4.2(a) shows the initial O-tree and Figure 4.2(b) shows the mutated O-tree. From this figure, we have seen the procedure of changing the shape of the solution by mutation operator. In this example, after mutation, the position of b, c and d have changed.

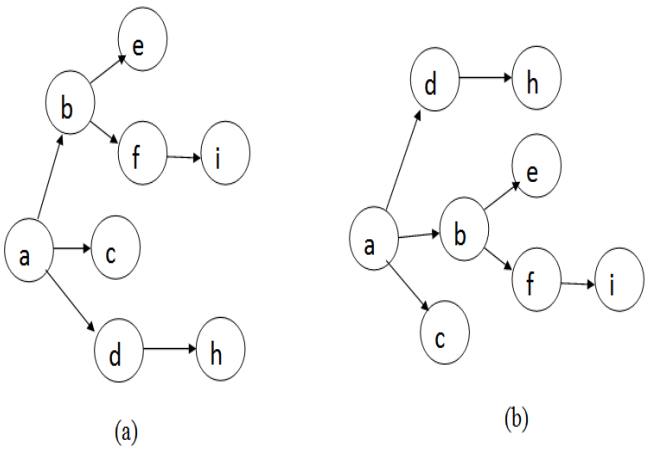


Figure 4.2: Illustration of (a) O-tree representation and (b) muted O-tree

4.4 Description of the CSA algorithms

For hybrid CSA algorithms, it randomly generates a population of individuals using the technique described above. Then, the algorithm starts evolving the population generation by

generation. In each generation, the algorithm uses the genetic operators on the individuals in the population to create new promising search points (admissible floorplans) and uses Hill-Climb and its variants described in Section 3.4.1 separately to optimize them. *best* is replaced by current admissible floorplan, if the FloorplanArea of the admissible floorplans is smaller than FloorplanArea of the *best*. The process is repeated until a preset runtime is up. Our algorithm is presented in Algorithm 10 and flowchart is shown in Figure 4.3.

The variants of our algorithm are:

- VAR1- clonal selection algorithm with Hill-climbing
- VAR2- clonal selection algorithm with Steepest Ascent Hill Climbing
- VAR3- clonal selection algorithm with Random-restart Hill Climbing

For these variants, our algorithm uses the CSA with mutation on the individuals in the population to create new promising search points and uses Hill-Climb, Steepest Ascent Hill-Climb and Random Restart Hill-Climb (described in Section 4.2.1, 4.2.2 and 4.2.3 respectively) separately. This is how we hybridize CSA. *best* is replaced by current admissible floorplan, if the FloorplanArea of the admissible floorplans is smaller than FloorplanArea of the *best*. The process is repeated until a preset runtime is up.

In this chapter we have presented all the techniques we applied in our algorithm. We have presented our algorithm in Algorithm 10. This is a generic algorithm for all three variants of our algorithms. By applying Hill-Climbing, Steepest Ascent Hill-Climbing and Random Restart Hill-Climbing separately we can get algorithms of all the variants.

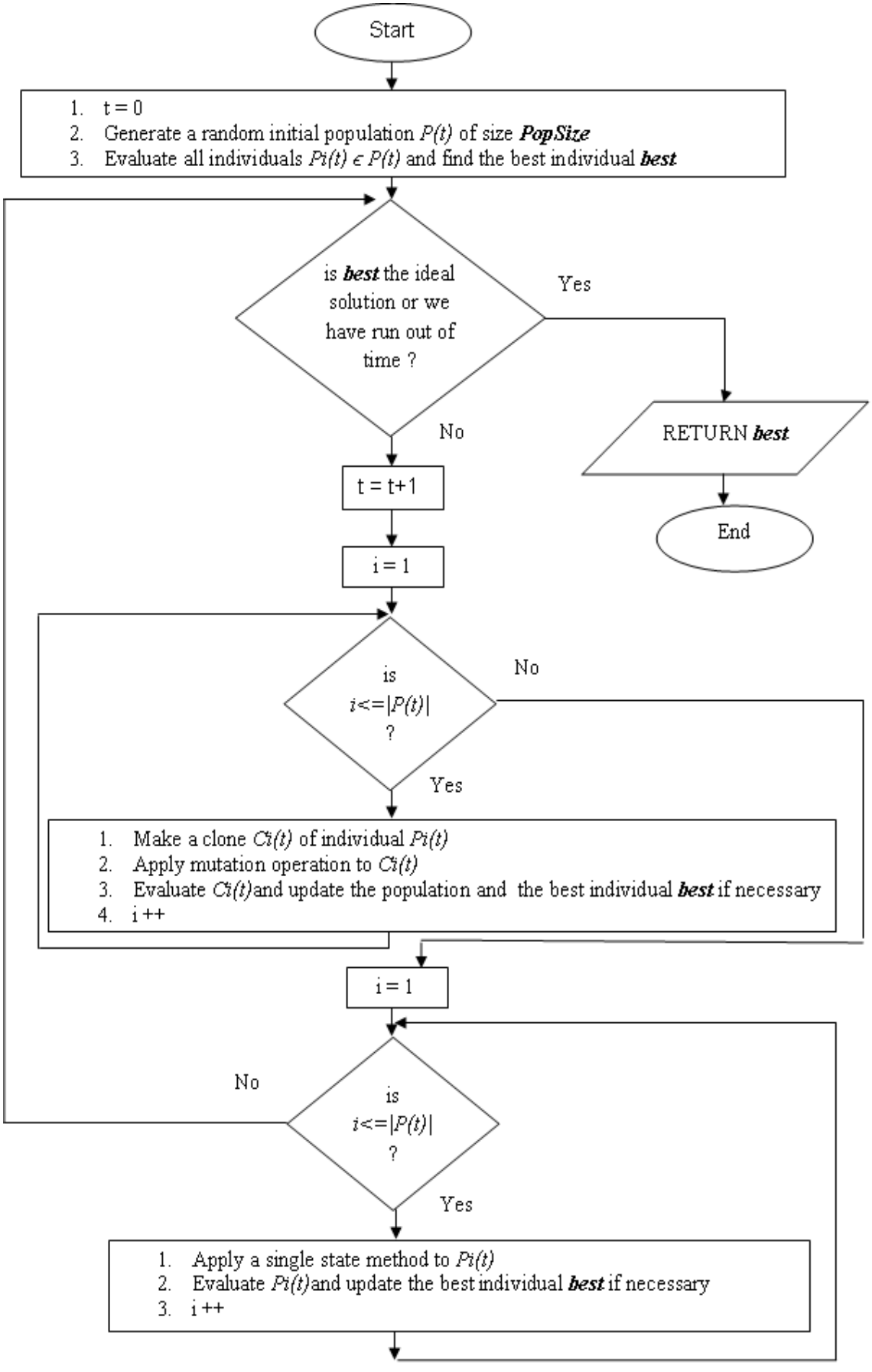


Figure 4.3: Flowchart of our algorithm

Algorithm 10 OurAlgorithm(*best*,*P*)

1: $m \leftarrow$ number of clones
2: $C \leftarrow \{\}$
3: **repeat**
4: **for** m times **do**
5: $C \leftarrow C \cup \{$randomly select an individual from P and then copy$\}$
6: **end for**
7: $count \leftarrow 0$
8: **for** each clone $c \in C$ **do**
9: $X \leftarrow$ mutate(c)
10: **if** $FloorPlanArea(X) < FloorPlanArea(c)$ **then**
11: $N \leftarrow N \cup \{X\}$
12: $count \leftarrow count + 1$
13: **end if**
14: **end for**
15: **for** $i \leftarrow 1$ to $count$ **do**
16: **for** each individual $I \in P$ **do**
17: **if** $FloorPlanArea(N_i) < FloorPlanArea(I)$ **then**
18: $I \leftarrow N_i$
19: **end if**
20: **end for**
21: **if** $FloorPlanArea(N_i) < FloorPlanArea(best)$ **then**
22: $best \leftarrow N_i$
23: **end if**
24: **end for**
25: **for** each individual $I \in P$ **do**
26: Hill-Climbing($best$, I)
27: **end for**
28: **until** *best* is an ideal solution or we have run out of time
29: **return** *best*

Chapter 5
Experiments and Results

This chapter presents the experimental results based on some existing benchmarks in the literature. It also describes the experiment data as well as environment and examine the experimental results. Finally the comparison between different algorithms are presented. For comparison purpose we perform statistical test and KruskalWallis test on the results using a statistical software, which is available in the website [1]. In this thesis, we have presented a number of metaheuristic algorithms which are listed in Table 5.1.

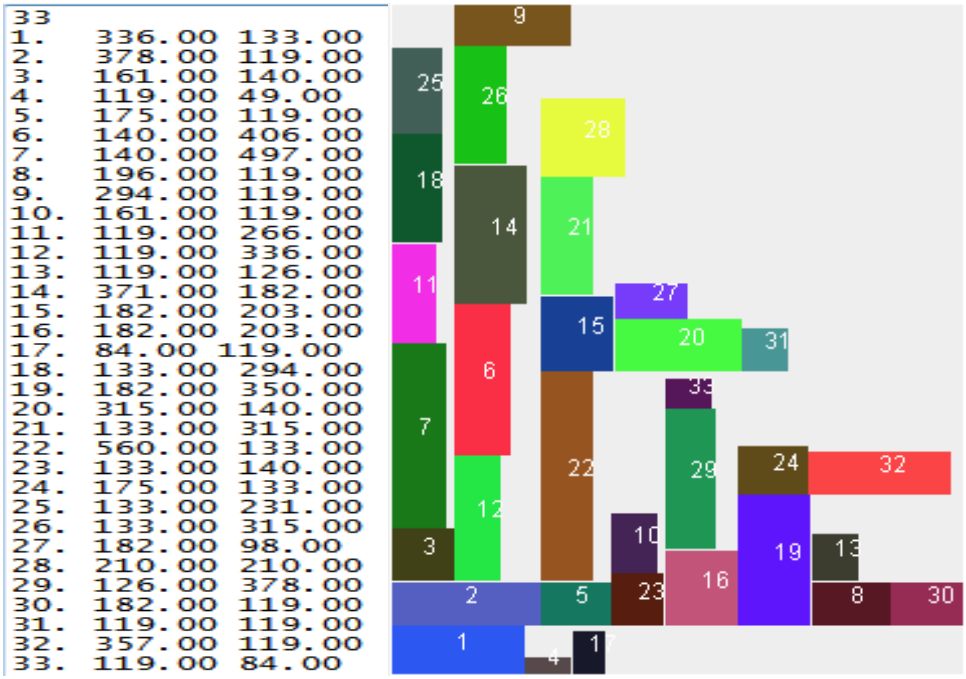
Table 5.1: New Metaheuristics Algorithms for VLSI Floorplanning

Serial	Our Algorithms	Acronyms
1.	Hill-Climbing Algorithm	HC
2.	Steepest Ascent Hill-Climbing Algorithm	SAHC
3.	Random Restart Hill-Climbing Algorithm	RRHC
4.	Clonal Selection Algorithm with HC	VAR1
5.	Clonal Selection Algorithm with SAHC	VAR2
6.	Clonal Selection Algorithm with RRHC	VAR3

In this chapter, we have presented inputs and outputs of our experiments using figures. So the description of those inputs and outputs are as follows. In the input file, at the first line, an integer n is given. This number indicates that, there are n number of modules in this floorplan. Information of these modules are given in next n lines. An module is represented as $s.\,W\,H$, where s, W and H indicate the index, width and height of that module respectively. In the output, the number of the rectangle corresponds to the indecies of the blocks/modules in the input file.

5.1 Benchmark Datasets

The benchmarks used in our empirical studies include two popular MCNC benchmark problems for the VLSI floorplanning problem: ami33 and ami49 [2]. The benchmark ami33 has 33 modules, 123 nets, 480 pins, and 42 inputoutput (IO) pads. The benchmark ami49 has 49 modules, 408 nets, 931 pins, and 42 IO pads. Other benchmarks of MCNC are apte and



(a) Input (b) Output

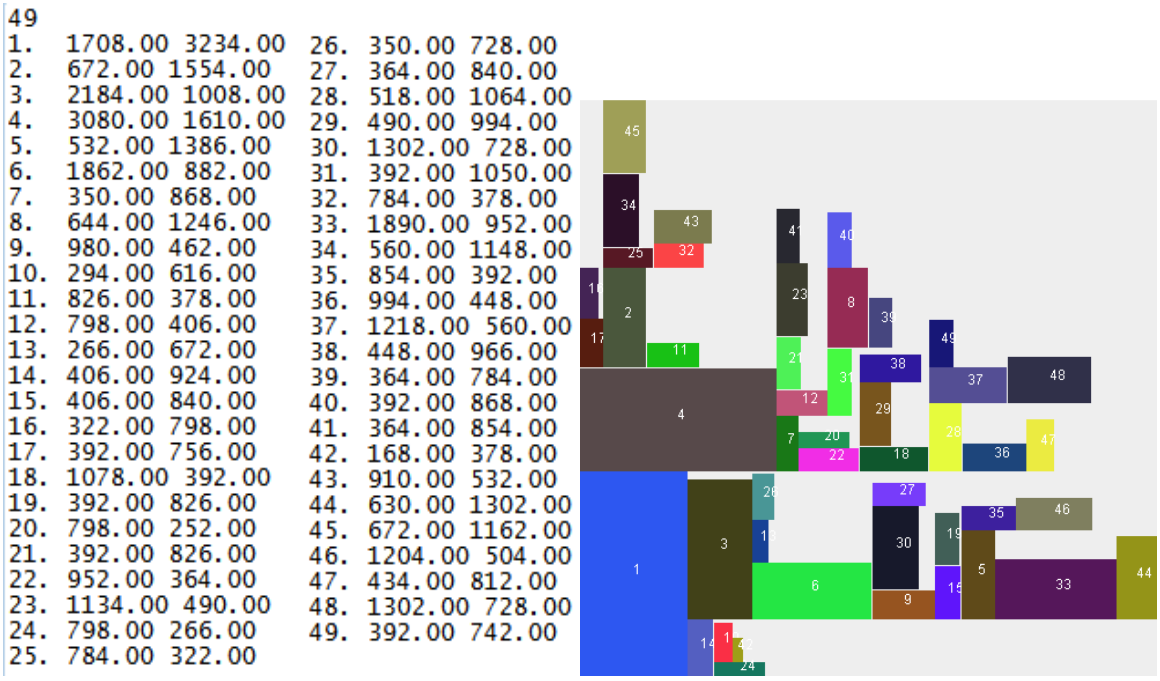
Figure 5.1: Simulation result of VLSI floorplanning by HC Algorithm on ami33 benchmark. After 30 runs, this is the final solution. Dead space is 44.11% of the total floorplan.

Xerox. Another popular benchmark is Gigascale Systems Research Center (GSRC).

The circuit characteristics of MCNC (MCNC, 2004) and GSRC (GSRC, 2006) benchmarks are presented in Table 5.2 and Table 5.3 respectively.

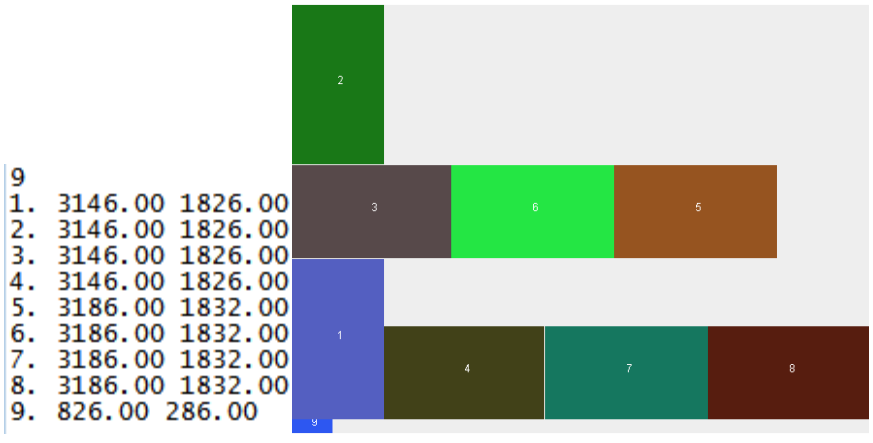
Table 5.2: The Standard MCNC benchmark in circuits

Circuit	no. of modules	no. of nets	no. of I/O pads	no. of pins	area
ami33	33	123	42	522	1.16
ami49	49	408	22	953	35.4
apte	9	97	73	287	46.56
hp	11	83	45	309	8.30
xerox	10	203	2	698	19.35



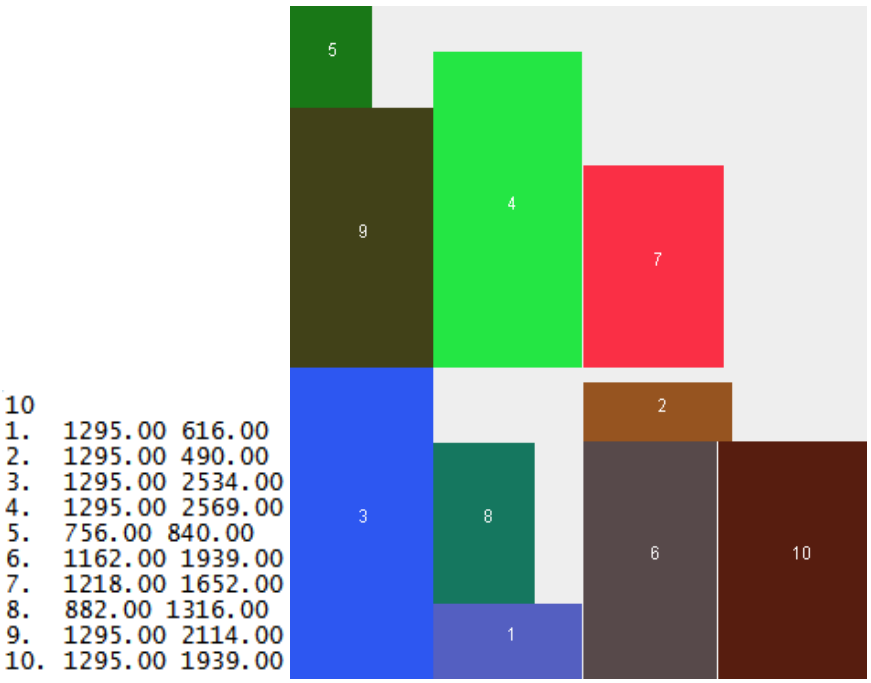
(a) Input (b) Output

Figure 5.2: Simulation result of VLSI floorplanning by HC Algorithm on ami49 benchmark. After 30 runs, this is the final solution. Dead space is 46.91% of the total floorplan.



(a) Input (b) Output

Figure 5.3: Simulation result of VLSI floorplanning by HC Algorithm on apte benchmark. After 30 runs, this is the final solution. Dead space is 33.53% of the total floorplan.

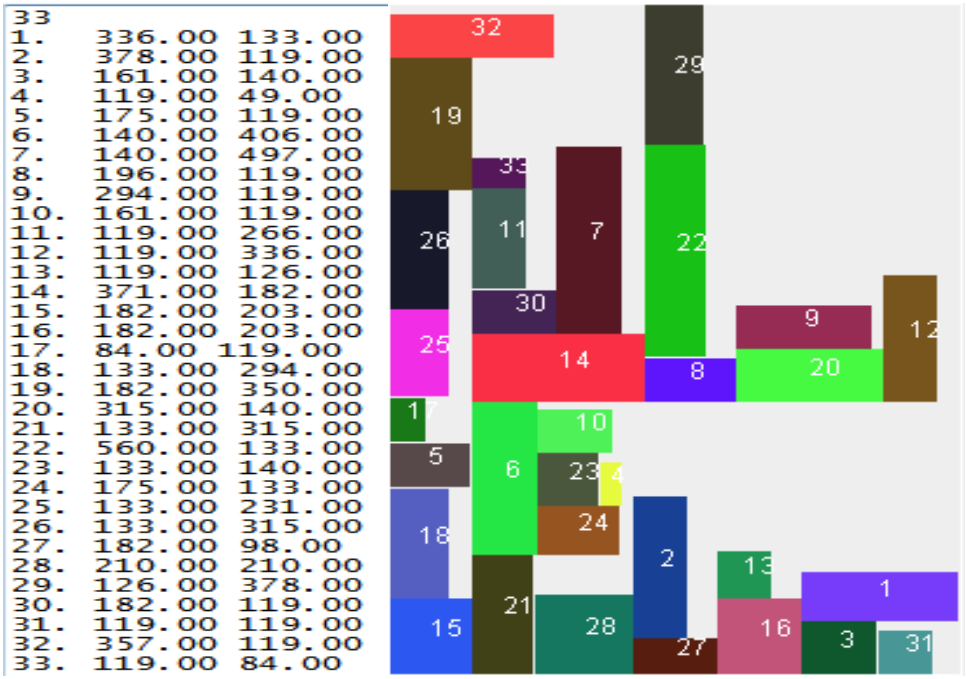


(a) Input (b) Output

Figure 5.4: Simulation result of VLSI floorplanning by HC Algorithm on xerox benchmark. After 30 runs, this is the final solution. Dead space is 30.14% of the total floorplan.

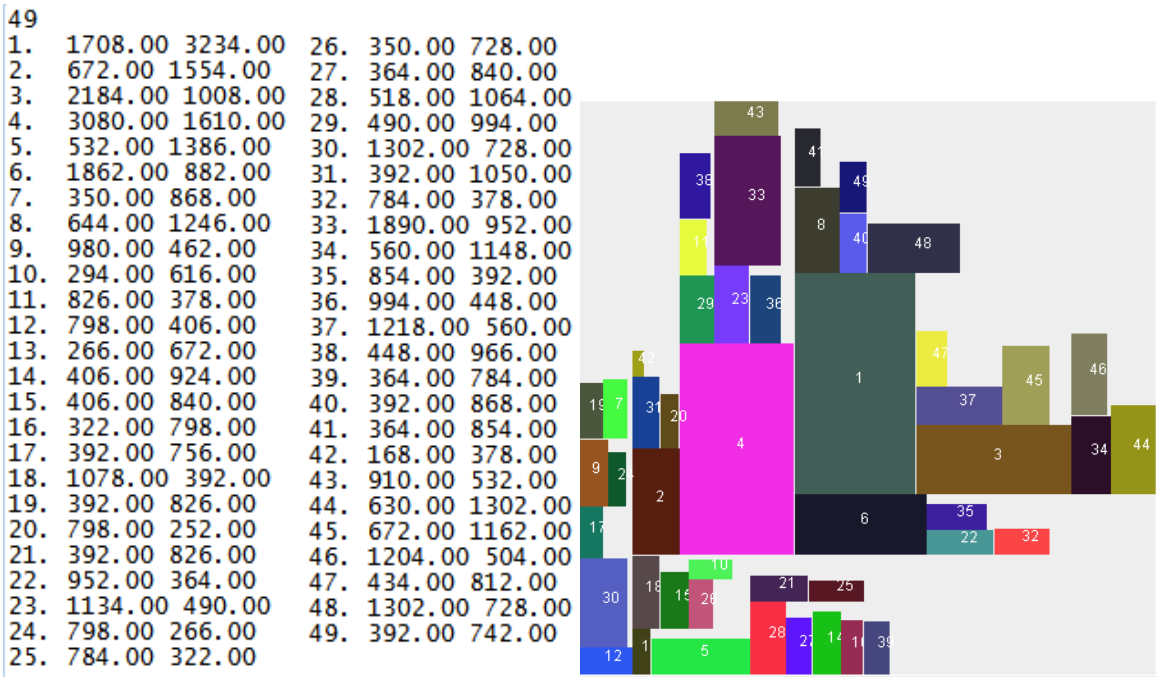
Table 5.3: The Standard GSRC benchmark in circuits

Circuit	no. of modules	no. of nets	no. of I/O pads	no. of pins	area
n10	10	118	69	248	22.17
n30	30	349	212	723	20.86
n50	50	485	209	1050	19.86
n100	100	885	334	1873	17.95
n200	200	1585	564	3599	17.57



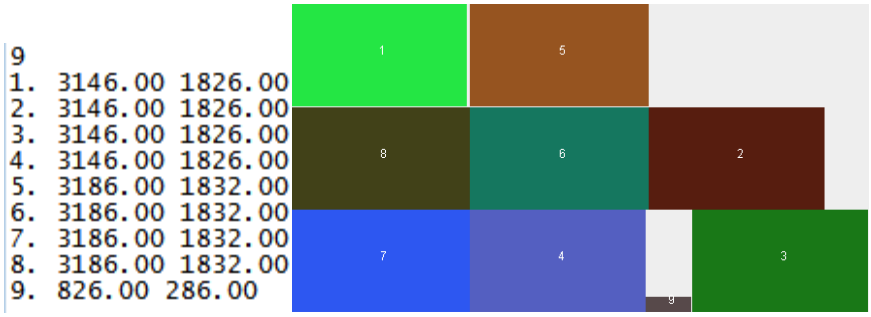
(a) Input (b) Output

Figure 5.5: Simulation result of VLSI floorplanning by SAHC Algorithm on ami33 benchmark. After 30 runs, this is the final solution. Dead space is 44.11% of the total floorplan.



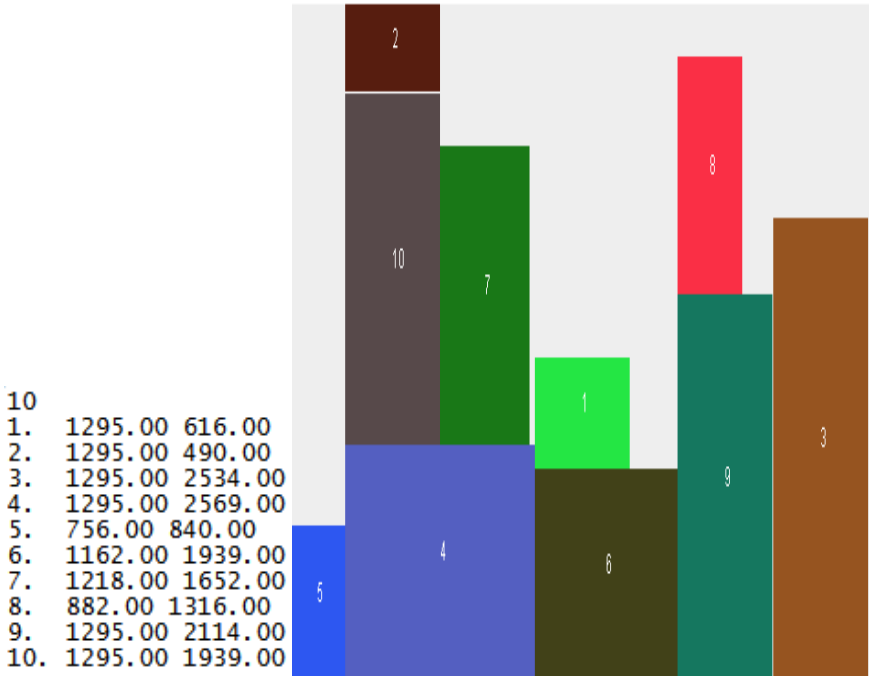
(a) Input (b) Output

Figure 5.6: Simulation result of VLSI floorplanning by Algorithm on ami49 benchmark. After 30 runs, this is the final solution. Dead space is 46.91% of the total floorplan.



(a) Input (b) Output

Figure 5.7: Simulation result of VLSI floorplanning by SAHC Algorithm on apte benchmark. After 30 runs, this is the final solution. Dead space is 20.53% of the total floorplan.



(a) Input (b) Output

Figure 5.8: Simulation result of VLSI floorplanning by SAHC Algorithm on xerox benchmark. After 30 runs, this is the final solution. Dead space is 30.14% of the total floorplan.

5.2 Experimental Setup

The experiments have been performed using Intel Atom CPU *N*2600 at 1.60 GHz dual core processor having 2GB RAM running Windows 7 operating system. Using the same environment we have implemented all the variants of our algorithm. During our experiments we followed the same procedures that Tang and Yao followed in their work [56]. For example, in [56], Tang and Yao fixed the population size to 10. They focused on minimizing the area of the floorplan and recorded the minimal floorplan area obtained in 30 runs.

Ususally, we fix the parameters after running some preliminary experiments on a small scale. In our thesis we have also fixed our parameters for different metaheuristic algorithms.

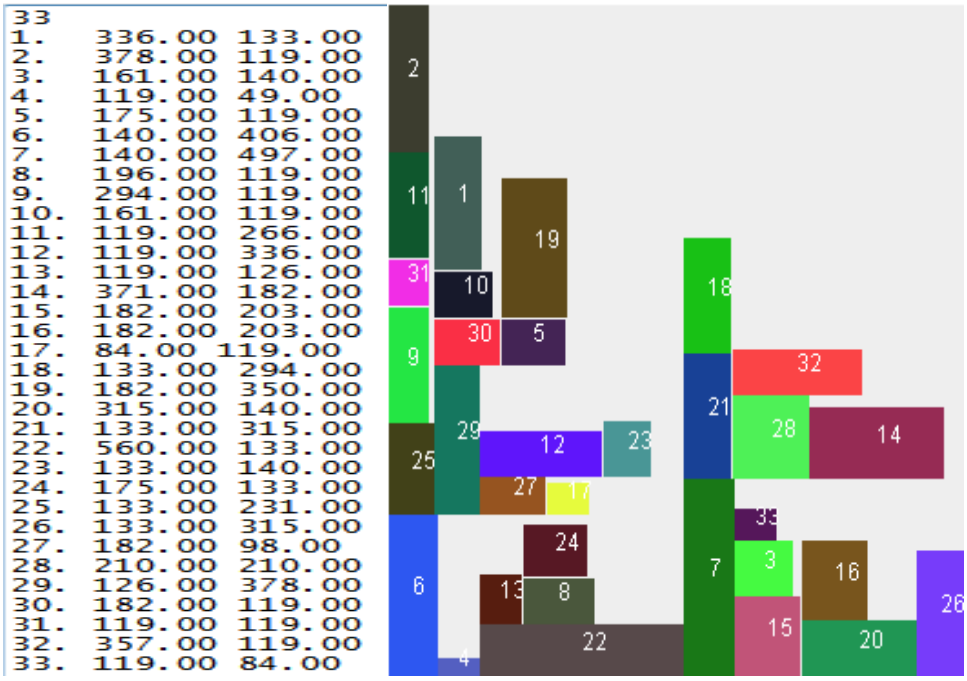
For our experiments, the population size was set to 10. So, we created 10 random individuals for the population. In our population based method, when we increased the population size, due to random individuals it explored the search space. On the other hand, when we decreased this value, then there created a chance to miss the better result. So, we fixed it to 10 as it was fixed in [56].

We focused on minimizing the area of the floorplan and recorded the floorplan area obtained in 30 runs. From the standard deviation presented in Table 5.4, we can see that our results do not deviate much.

We have fixed the clone size to 5. Because we have 10 individuals in our population. So we cannot take larger size to make clone of those individuals.

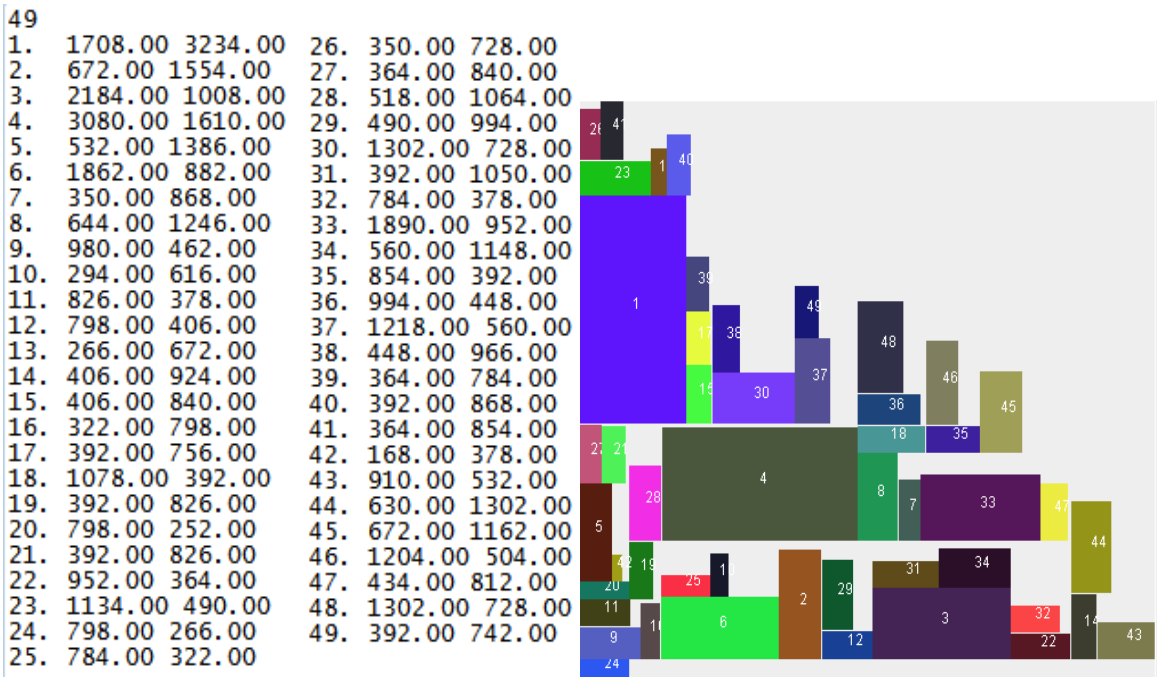
5.3 Experimental Results and Discussion

In [56], Tang and Yao proved that their memetic algorithm (MA) is better than some other algorithms. So we have compared all the variants of our algorithm with all of those algorithms. The procedure for generating the initial population and selection scheme are exactly the same with those used in the MA. But genetic operators are not same. MA used crossover, on the other hand, in our thesis we have used Clonal Selection Algorithm with mutation for exploration. For exploitation, our algorithms use HC with its variants which give three variations of metaheuristic algorithms.



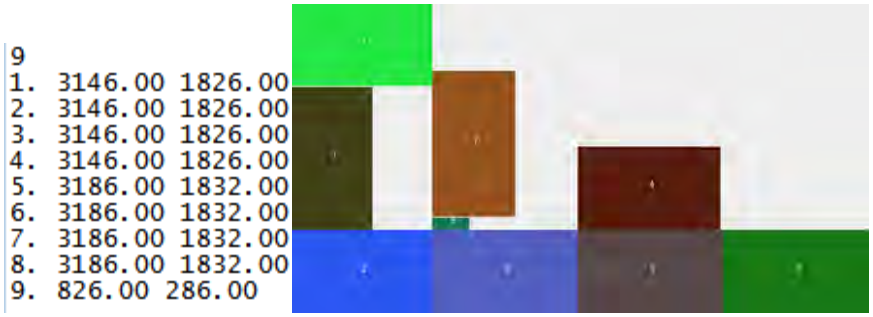
(a) Input (b) Output

Figure 5.9: Simulation result of VLSI floorplanning by RRHC Algorithm on ami33 benchmark. After 30 runs, this is the final solution. Dead space is 44.11% of the total floorplan.



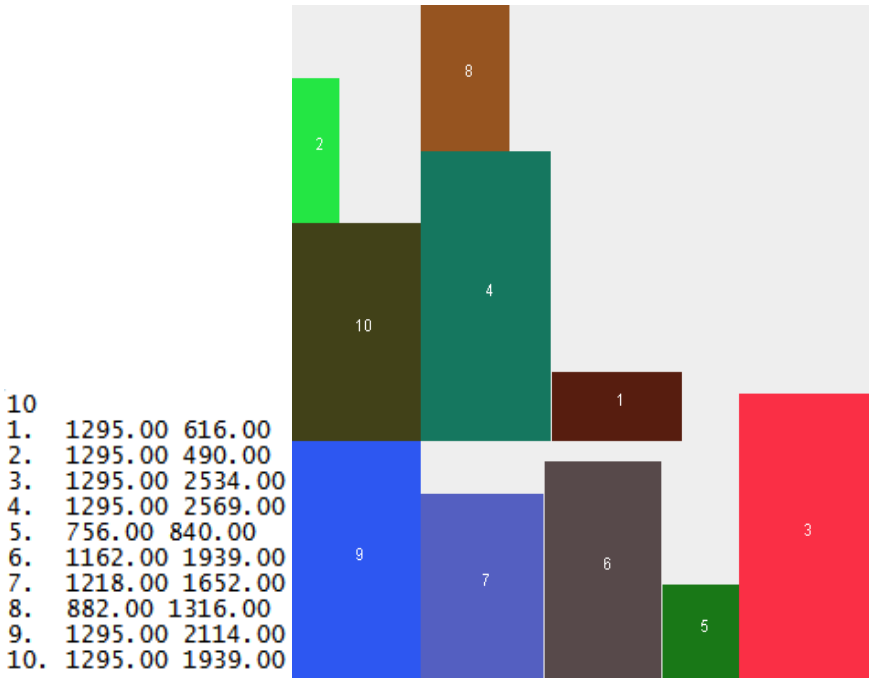
(a) Input (b) Output

Figure 5.10: Simulation result of VLSI floorplanning by RRHC Algorithm on ami49 benchmark. After 30 runs, this is the final solution. Dead space is 46.91% of the total floorplan.



(a) Input (b) Output

Figure 5.11: Simulation result of VLSI floorplanning by RRHC Algorithm on apte benchmark. After 30 runs, this is the final solution. Dead space is 32.82% of the total floorplan.



(a) Input (b) Output

Figure 5.12: Simulation result of VLSI floorplanning by RRHC Algorithm on xerox benchmark. After 30 runs, this is the final solution. Dead space is 30.14% of the total floorplan.

At first, we have implemented Hill-Climbing algorithm and its variants to solve our problem. These techniques are discussed in Section 3.4.1. Then we have implemented Clonal Selection Algorithm with mutation and included HC and its variants as separate metaheuristic techniques, which is presented in Section 3.7.3 and Section 3.8 and have created three variants.

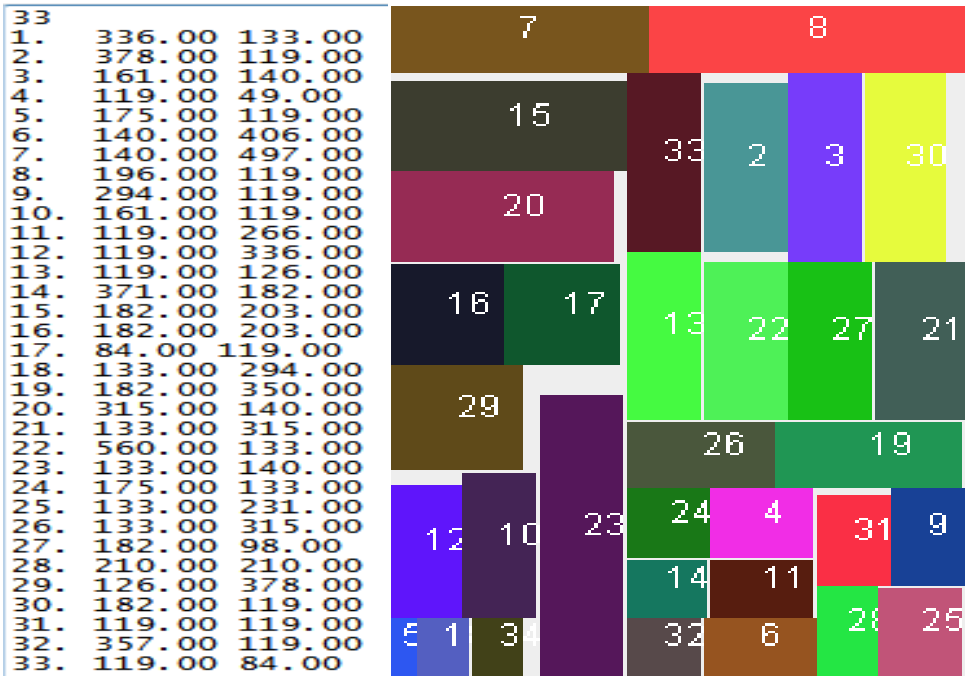
Our algorithms are described in detail in Chapter 4. In that chapter we presented our noval concept to solve VLSI floorplan problem.

We have run each of our algorithms on each of the benchmark problems for 30 times, recorded the results obtained from each run, and compared the results of other algorithms. From these 30 runs we have calculated the best area as well as the mean area. Again, we have calculated the standard deviation of these areas to understand the variations of the area in these 30 runs. Table 5.4 shows the statistics for the experiment and previous works' result. It can be seen from Table 5.4 that for the benchmark problems (ami33 and ami49), one of the variant of our algorithms, Clonal Selection Algorithm with Hill-Climbing (VAR1) has 0.002% to 0.05% and 0.02% to 0.66% improvement over the MA in the best and average solutions respectively. For other benchmarks VAR1 also provides better result. To examine the significance of the statistical results, we further performed a Kruskal-Wallis test on the results using a statistical software [1]. The statistical test results are shown in Table 5.5. In this test, the test has started by substituting the rank in the overall data set for each measurement value from the smallest value to the largest value. That is, each value has got a rank. Then we have calculated the rank sum and then mean rank for each of the benchmarks and for all algorithms. From this table we have seen that, mean rank of VAR1 algorithm is minimum than mean ranks of other algorithms. This means, VAR1 algorithm has provided better areas from 30 runs than other algorithms.

In [56], Tang and Yao compared their algorithm with mDA and GA algorithms using two popular MCNC benchmarks (ami33, ami49) only. They did not use GSRC benchmark for their experiments. On the other hand, a recent work performed a greedy clonal selection algorithm with both MCNC and GSRC benchmarks in their experiments [3]. This is why, to compare we ran all of our algorithms in all benchmarks. Then we compared our algorithms with MA, mDA, GA using MCNC benchmarks and with greedy clonal selection algorithm using all benchmarks.

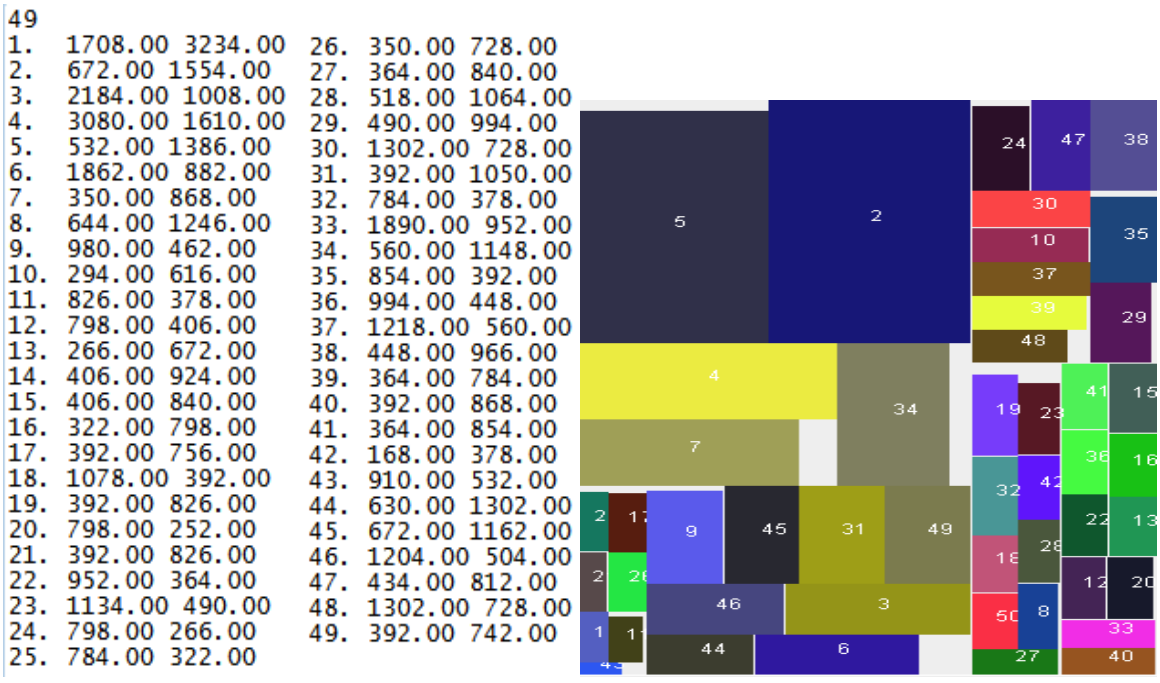
Table 5.4: Statistics for the test results on the performance of our algorithm

benchmarks		*ami*33	*ami*49	*apte*	*xerox*
mDA	best	1225931	37902872	not found	not found
	mean	1240946	38231446	not found	not found
	std dev	8074	174441	NA	NA
GA	best	1241219	38069668	not found	not found
	mean	1278441	38797586	not found	not found
	std dev	14990	255458	NA	NA
MA	best	1190896	37487940	not found	not found
	mean	1211159	38093769	not found	not found
	std dev	8155	228102	NA	NA
HC	best	2068976	66767008	70046592	27697936
	mean	2522007	74004112	80925500	29793628.43
	std dev	154022.09	5440309.82	6465274.98	2169292.25
SAHC	best	2068976	66767008	70046592	27697936
	mean	2355342	72097005.87	80925500	29276093.7
	std dev	254948.03	5864022.02	6868934.94	2163565.83
RRHC	best	2068976	66767008	69305760	27697936
	mean	2862614	76414337.07	80925500	31874686.2
	std dev	315878.05	4606414.53	6499594.47	4981555.08
VAR1	best	**1190285**	**37487346**	**47528748**	**20063517**
	mean	**1203272**	**38087346**	**48884114**	**21319177**
	std dev	49422.1	932183.2	2757073.5	2554252.1
VAR2	best	**1190285**	**37487346**	**47528748**	**20063517**
	mean	1222752	38220679.33	49561798	21947007
	std dev	73838.37	980265.04	3158624.47	2926263.36
VAR3	best	**1190285**	**37487346**	**47528748**	**20063517**
	mean	1204195	3814292.7	49561798	21947007
	std dev	49295.86	910305.51	3158624.47	2926263.36



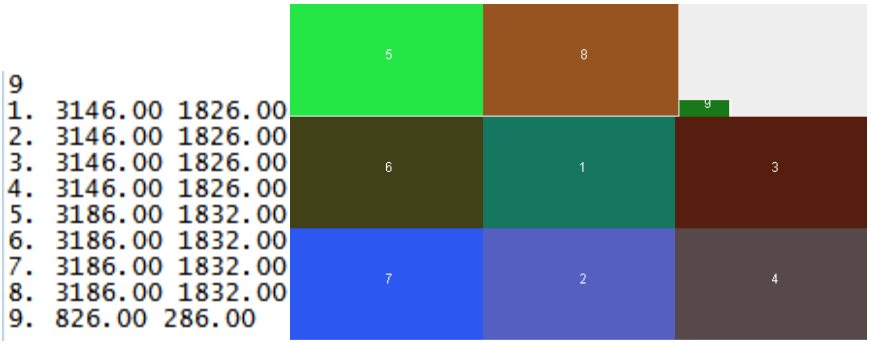
(a) Input (b) Output

Figure 5.13: Simulation result of VLSI floorplanning by VAR1 Algorithm on ami33 benchmark. After 30 runs, this is the final solution. Dead space is 2.84% of the total floorplan.



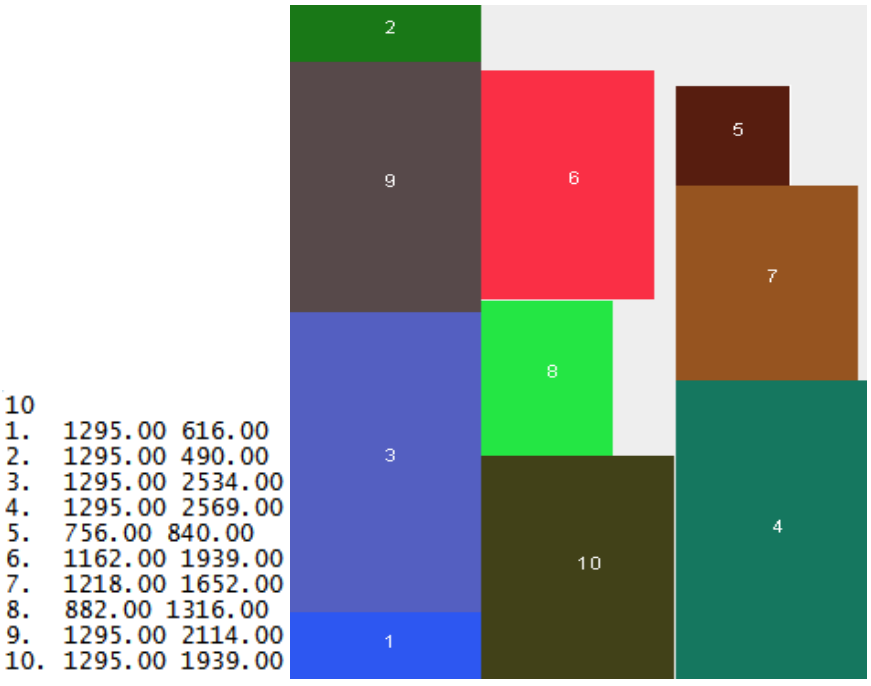
(a) Input (b) Output

Figure 5.14: Simulation result of VLSI floorplanning by VAR1 Algorithm on ami49 benchmark. After 30 runs, this is the final solution. Dead space is 5.45% of the total floorplan.



(a) Input (b) Output

Figure 5.15: Simulation result of VLSI floorplanning by VAR1 Algorithm on apte benchmark. After 30 runs, this is the final solution. Dead space is 2.03% of the total floorplan.

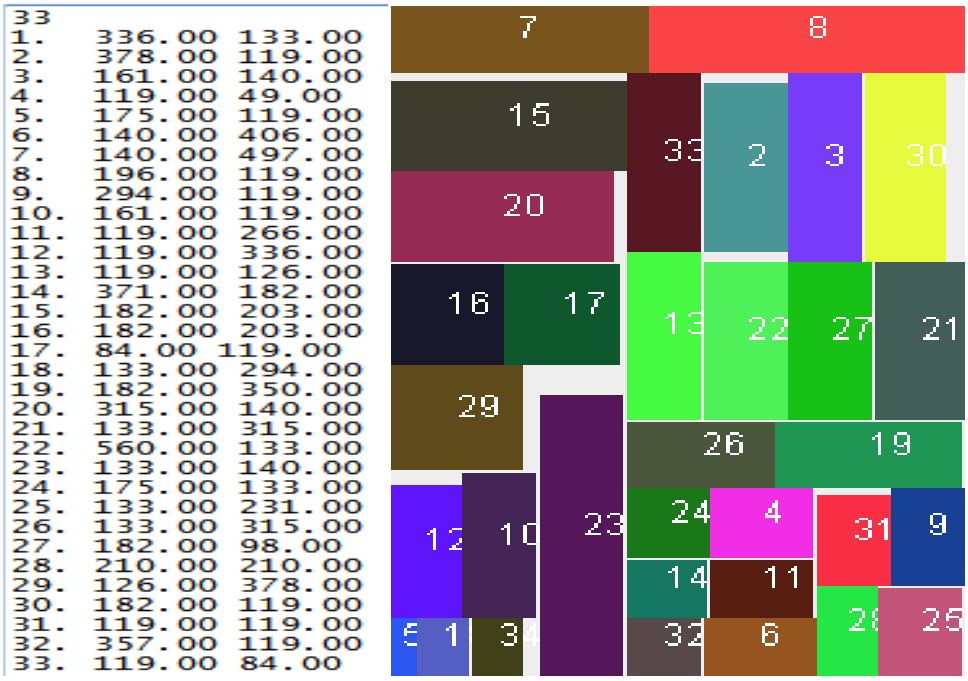


(a) Input (b) Output

Figure 5.16: Simulation result of VLSI floorplanning by VAR1 Algorithm on xerox benchmark. After 30 runs, this is the final solution. Dead space is 3.55% of the total floorplan.

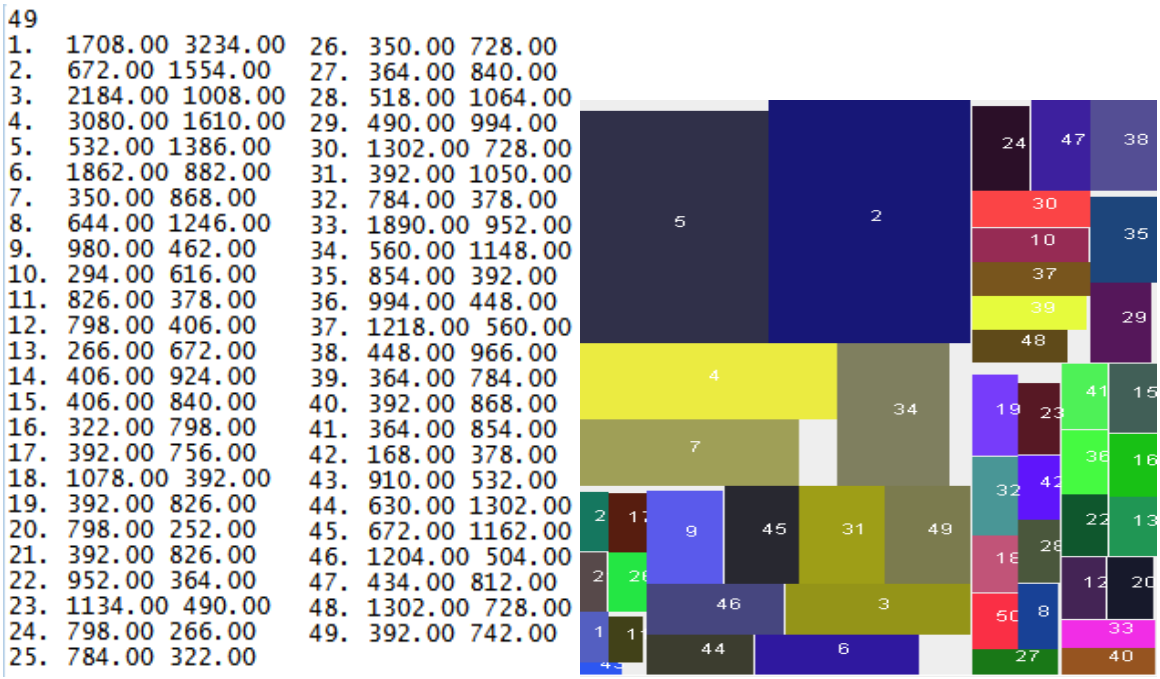
Table 5.5: Kruskal-Wallis test results for the comparison between the mDA, GA, MA and variants of our algorithms

benchmarks		*ami*33	*ami*49	*apte*	*xerox*
MA	n	30	30	not found	not found
	rank sum	2847	2295	NA	NA
	mean rank	94.9	76.5	NA	NA
HC	n	30	30	30	30
	rank sum	4814.5	4974	4182	4084
	mean rank	160.48	165.8	139.4	136.13
SAHC	n	30	30	30	30
	rank sum	4489.5	4667	3814	3804
	mean rank	149.65	155.57	127.13	126.8
RRHC	n	30	30	30	30
	rank sum	5591	5254	4199	4307
	mean rank	186.37	175.13	139.97	143.57
VAR1	n	30	30	30	30
	rank sum	**1352**	**1573.5**	**1275**	**1275**
	mean rank	**45.07**	**52.45**	**42.5**	**42.5**
VAR2	n	30	30	30	30
	rank sum	1580	1726.5	1410	1410
	mean rank	52.67	57.55	47	47
VAR3	n	30	30	30	30
	rank sum	1481	1665	1410	1410
	mean rank	49.37	55.5	47	47



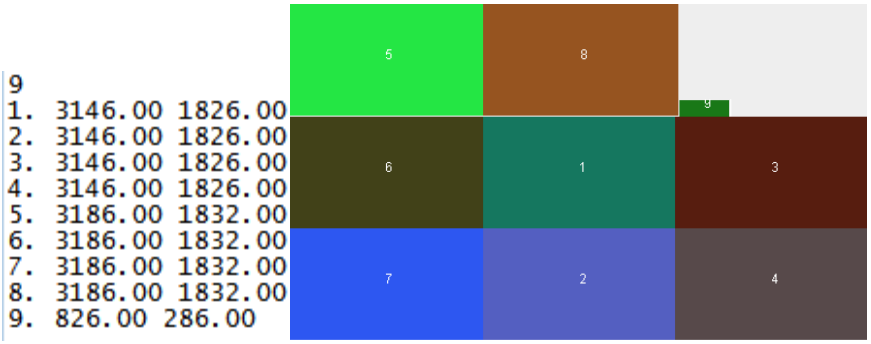
(a) Input (b) Output

Figure 5.17: Simulation result of VLSI floorplanning by VAR2 Algorithm on ami33 benchmark. After 30 runs, this is the final solution. Dead space is 2.84% of the total floorplan.



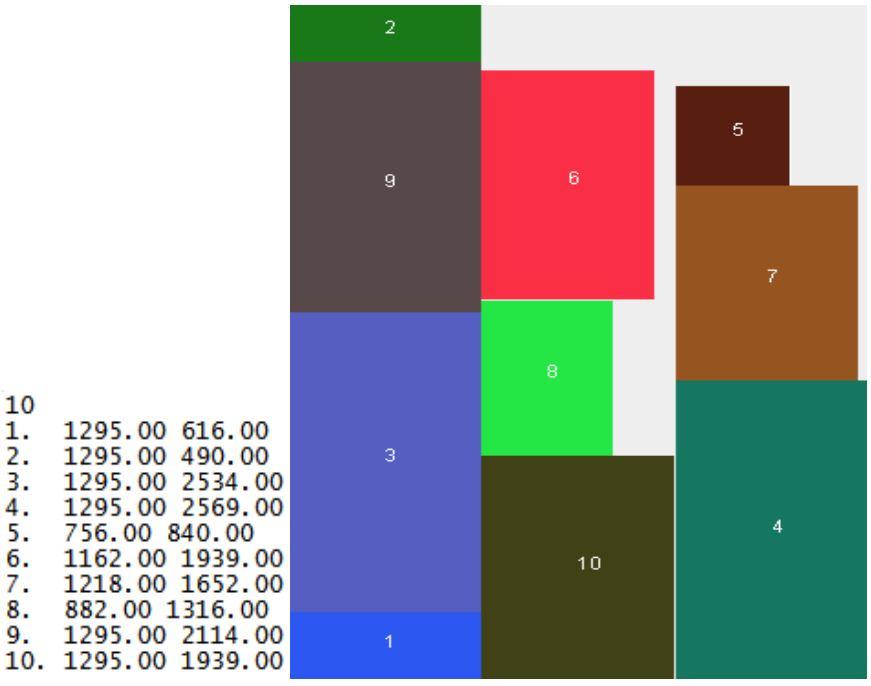
(a) Input (b) Output

Figure 5.18: Simulation result of VLSI floorplanning by VAR2 Algorithm on ami49 benchmark. After 30 runs, this is the final solution. Dead space is 5.45% of the total floorplan.



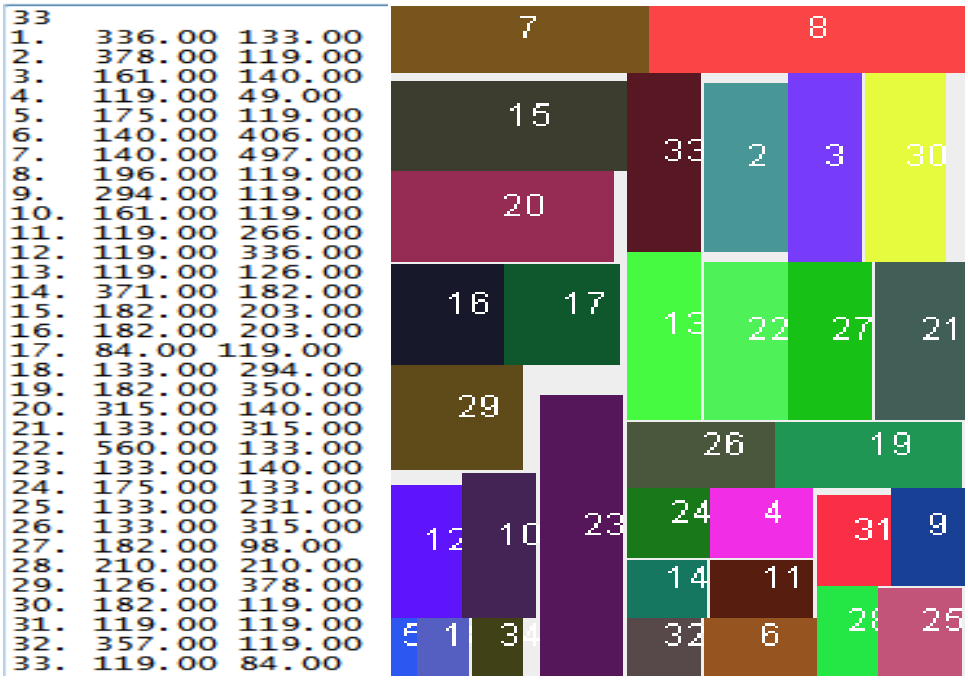
(a) Input (b) Output

Figure 5.19: Simulation result of VLSI floorplanning by VAR2 Algorithm on apte benchmark. After 30 runs, this is the final solution. Dead space is 2.03% of the total floorplan.



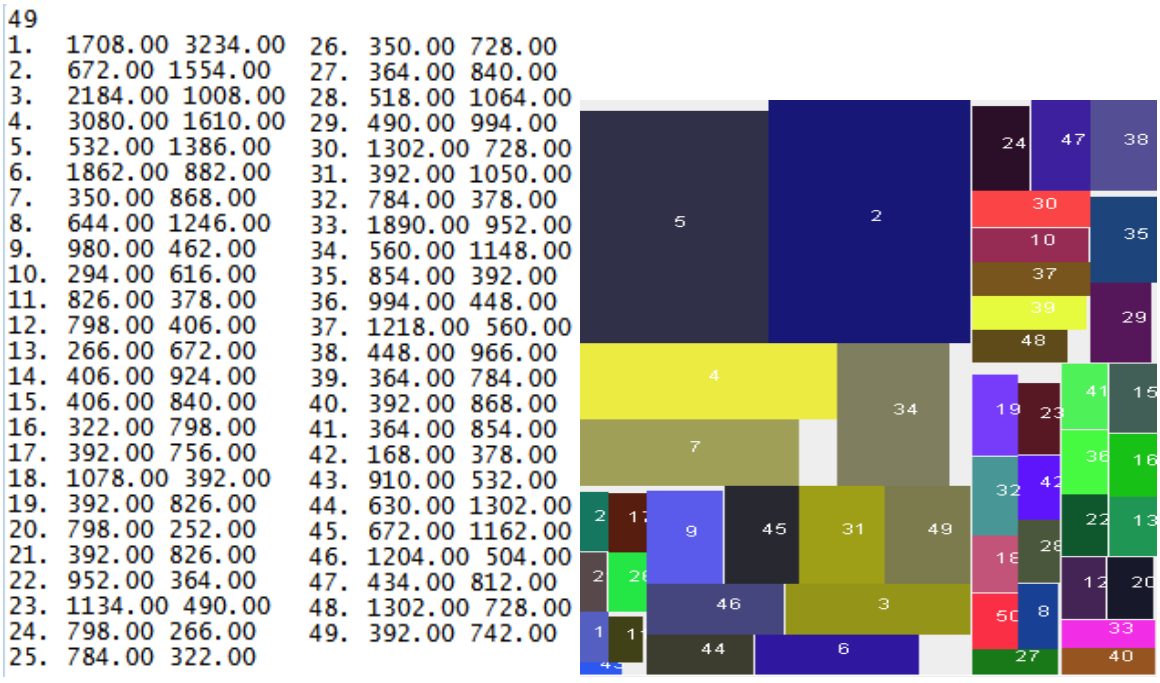
(a) Input (b) Output

Figure 5.20: Simulation result of VLSI floorplanning by VAR2 Algorithm on xerox benchmark. After 30 runs, this is the final solution. Dead space is 3.55% of the total floorplan.



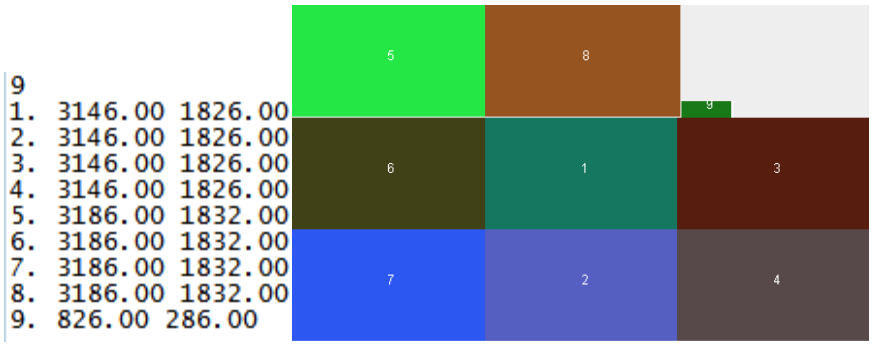
(a) Input (b) Output

Figure 5.21: Simulation result of VLSI floorplanning by VAR3 Algorithm on ami33 benchmark. After 30 runs, this is the final solution. Dead space is 2.84% of the total floorplan.



(a) Input (b) Output

Figure 5.22: Simulation result of VLSI floorplanning by VAR3 Algorithm on ami49 benchmark. After 30 runs, this is the final solution. Dead space is 5.45% of the total floorplan.



(a) Input (b) Output

Figure 5.23: Simulation result of VLSI floorplanning by VAR3 Algorithm on apte benchmark. After 30 runs, this is the final solution. Dead space is 2.03% of the total floorplan.

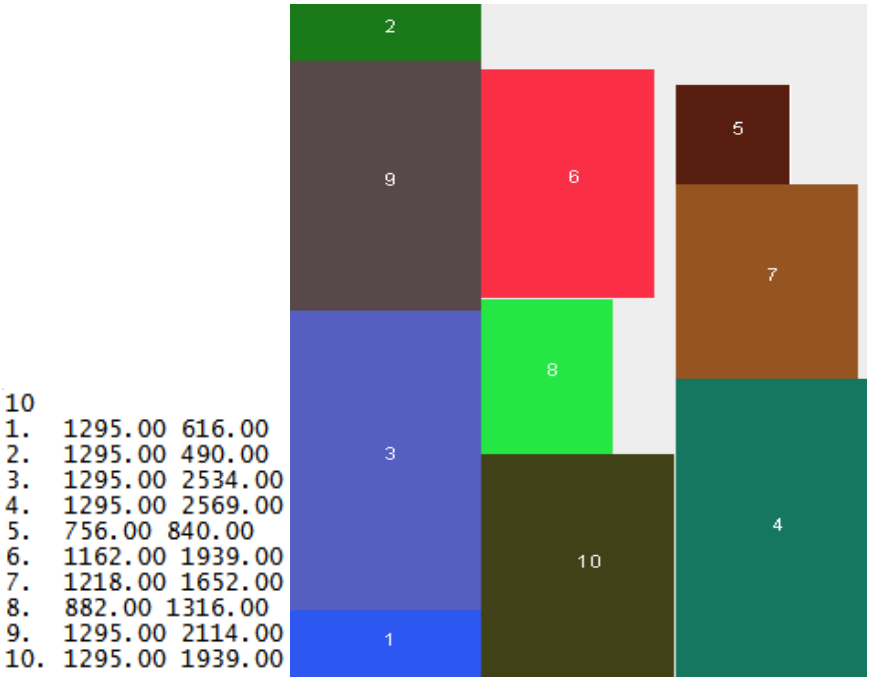
Table 5.6 shows that VAR1 of our algorithms provide better result than greedy CSA in [3].

Table 5.6: Cost or Dead Space of VLSI floorplanning for different Benchmark Dataset to compare VAR1 of our algorithms with greedy clonal selection algorithm

Benchmark Dataset	Cost(in percentage %) for Greedy CSA	Cost(in percentage %) for our algorithms
ami33	16.495	**2.925853**
ami49	13.4480	**5.760749**
apte	6.154	**2.077075**
xerox	22.551	**3.68584**
GSRC n10	22.169	**10.65775**

We also compared VAR1 of our algorithms with a previous greedy clonal selection algorithm for VLSI floorplanning problem. That gready clonal selection algorithm followed a greedy approach to make initial population by sorting all the blocks by their heights and widths [3]. Experimental results show that VAR1, VAR2 and VAR3 of our algorithms are better than that approach followed in [3]. The comparison between these two algorithms for popular MCNC benchmarks are shown in Figure 5.25, where we can see that, by using our algorithms (VAR1, VAR2 and VAR3) we can produce floorplan with minimum area than that greedy approach.

We can also compare the greedy CSA algorithm and our algorithms with respect to percentage of dead space. For MCNC and GSRC benchmarks we have compared these algorithms which is presented in Figure 5.26. From the figure we can see that our algorithm costs less



(a) Input (b) Output

Figure 5.24: Simulation result of VLSI floorplanning by VAR3 Algorithm on xerox benchmark. After 30 runs, this is the final solution. Dead space is 3.55% of the total floorplan.

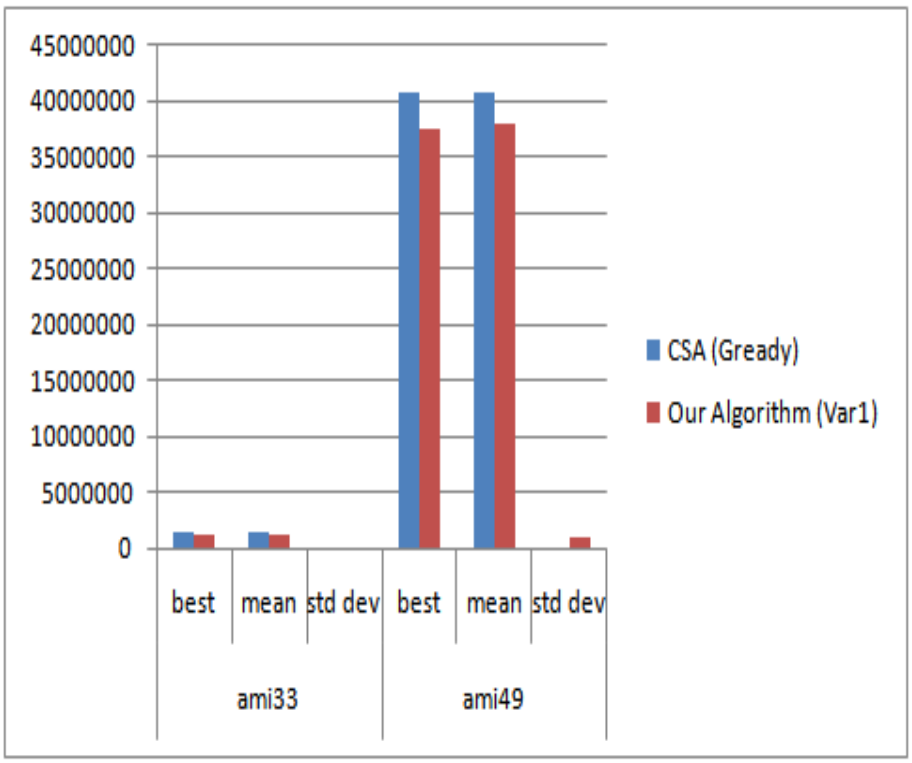


Figure 5.25: Comparison between Our Algorithm (VAR1) with a greedy CSA

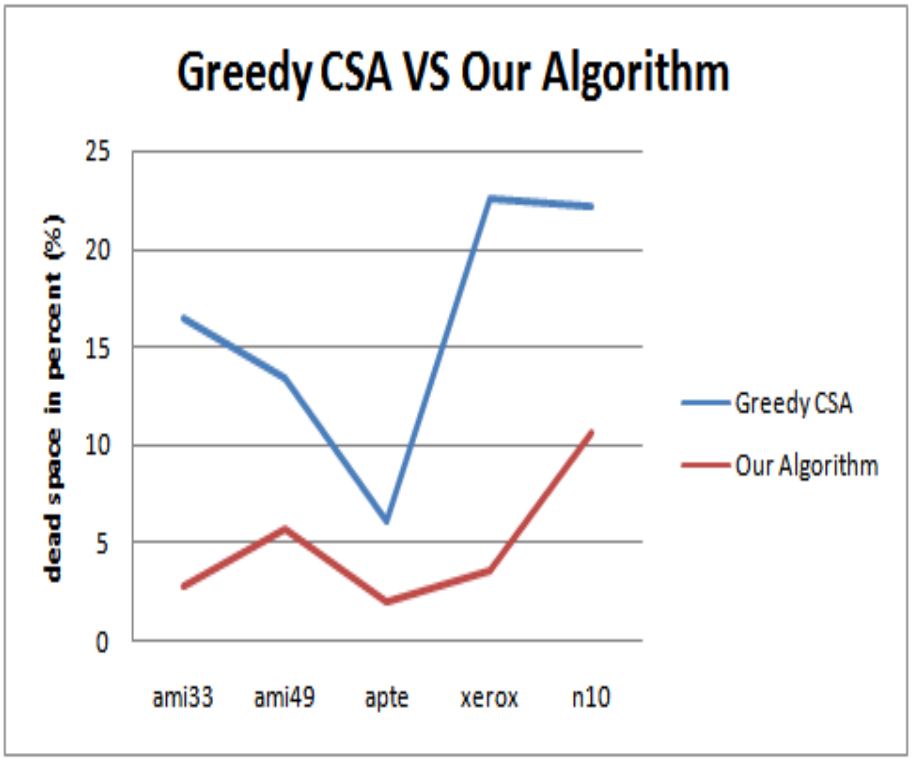


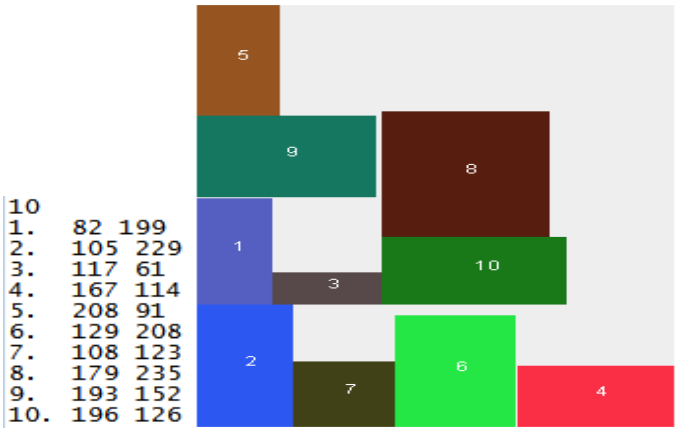
Figure 5.26: Comparison between Our Algorithm (VAR1) with a greedy CSA

dead space than greedy CSA.

5.4 Handling Interference

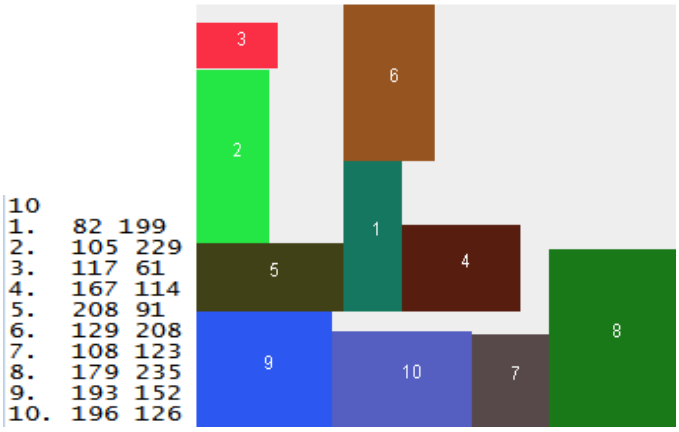
In this section, we are going to present how we have gained the capability of handling mixed signal blocks in our work. A mixed-signal integrated circuit is any integrated circuit that has both analog circuits and digital circuits on a single semiconductor die. There could be some restrictions regarding the placement of such mixed signal blocks due to possible interference. In particular, a particular mixed signal block may create signal interference if it is placed near to another similar block. Suppose we have 10 blocks to place in a plan. It could be the case that blocks $\{1,4,8\}$ cannot be adjacent to each other.

When there are mixed signal blocks, we assume that this information will be given at the end of the input file as follows: an integer n is given, which denotes the number of interferences in the floorplan. After that, n number of interferences are given in that file. In Figure 5.33 (b), we have seen that, there are 2 interferences i.e., $\{1, 2\}$ and $\{5, 9\}$ in the input file. So, in the floorplan block 1 and block 2 cannot be adjacent to each other. Similarly, block 5 and



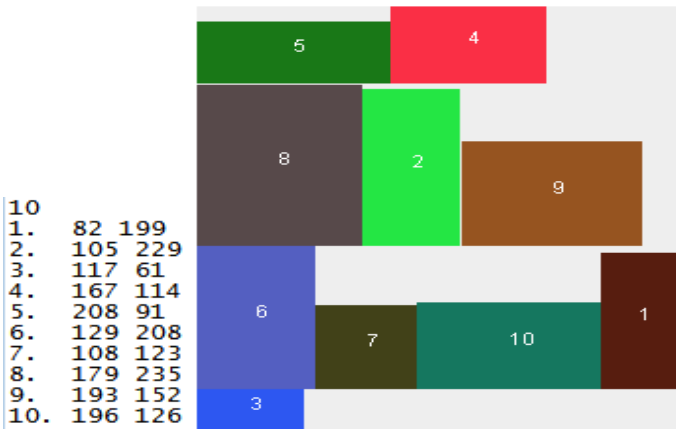
(a) Input (b) Output

Figure 5.27: Simulation result of VLSI floorplanning by HC Algorithm on GSRC (n10) benchmark. After 30 runs, this is the final solution. Dead space is 42% of the total floorplan.



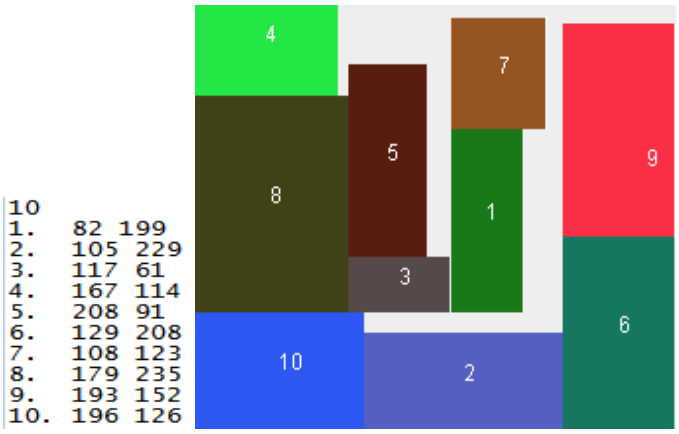
(a) Input (b) Output

Figure 5.28: Simulation result of VLSI floorplanning by SAHC Algorithm on GSRC (n10) benchmark. After 30 runs, this is the final solution. Dead space is 52% of the total floorplan.



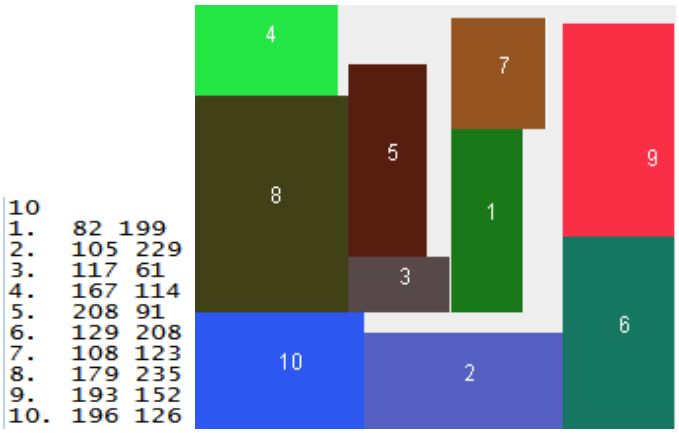
(a) Input (b) Output

Figure 5.29: Simulation result of VLSI floorplanning by RRHC Algorithm on GSRC (n10) benchmark. After 30 runs, this is the final solution. Dead space is 43.11% of the total floorplan.



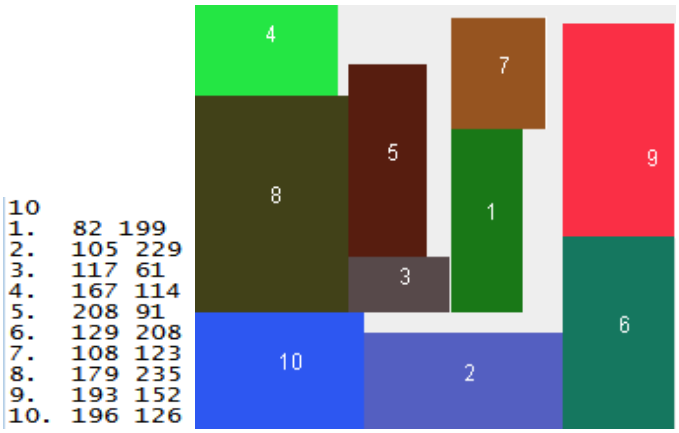
(a) Input (b) Output

Figure 5.30: Simulation result of VLSI floorplanning by VAR1 Algorithm on GSRC (n10) benchmark. After 30 runs, this is the final solution. Dead space is 10.66% of the total floorplan.



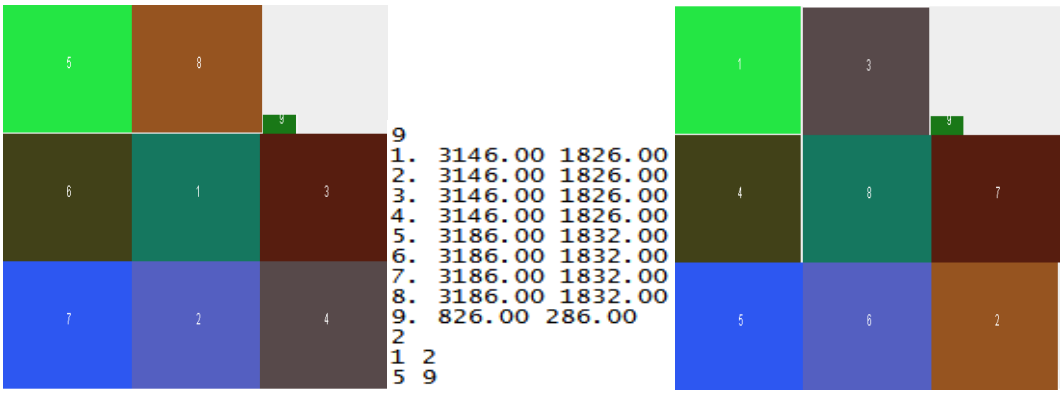
(a) Input (b) Output

Figure 5.31: Simulation result of VLSI floorplanning by VAR2 Algorithm on GSRC (n10) benchmark. After 30 runs, this is the final solution. Dead space is 10.66% of the total floorplan.



(a) Input (b) Output

Figure 5.32: Simulation result of VLSI floorplanning by VAR3 Algorithm on GSRC (n10) benchmark. After 30 runs, this is the final solution. Dead space is 10.66% of the total floorplan.



(a) Output without considering interference

(b) Input

(c) Output considering interference

Figure 5.33: Simulation result of VLSI floorplanning by VAR1 Algorithm on apte benchmark. Outputs with and without considering interference.

block 9 cannot be adjacent to each other too.

Without considering interference, we have got the output as shown in Figure 5.33 (a), where blocks 1 and 2 are adjacent to each other. When we have considered interference, we have got output as shown in Figure 5.33 (b), and blocks 1 and 2 are not adjacent to each other. The process to avoid adjacency between two blocks has been discussed as follows. When a block with interference has come to be placed in a particular position in the plane, we have selected that position for that block, if the parent of the stated block is not same as the parent of its interfering block. Algorithm 11 finds a position for the block that avoids interference.

Algorithm 11 FindPosition(*block*)

1: **for** each unused available positions **do**
2: **if** $parent(block) \neq parent(interfere(block))$ **then**
3: **return** *position*
4: **end if**
5: **end for**

5.5 Discussion of output

We have also designed a visualization software. With the help of that software, we can show any floorplan which is saved in a previous run. For a specific variant of our algorithms and

for a specific benchmark from the available benchmarks, our software saves output, which is the best in a particular run. Then we can visualize our output from this software. Now the outputs by our visualization software for Hill-Climbing, Steepest Ascent Hill-Climbing, Random Restart Hill-Climbing, VAR1, VAR2 and VAR3 algorithms and ami33, ami49, apte and Xerox benchmarks are shown in Figure 5.1 to Figure 5.24 respectively.

The outputs by our visualization software for Hill-Climbing, Steepest Ascent Hill-Climbing, Random Restart Hill-Climbing, VAR1, VAR2 and VAR3 algorithms using GSRC (n=10) benchmarks are shown in Figure 5.27 to Figure 5.32 respectively.

This thesis has shown the experimental results on different benchmark data, using a number of metaheuristic algorithms for VLSI floorplanning. The main impact of our work is on the quality of the total area. Minimizing the total area is the principal objective of most existing floorplanners. Minimization of the total area is helpful to minimize chip size, and thus cost. This work represents our first effort toward efficient algorithms for VLSI floorplanning.

Chapter 6
Visualization Software

We have designed a visualization software in JAVA, IntelliJ IDEA Community Edition 13.1.4. With the help of this software, we can view floorplans, run all the variants of our algorithm. Then we can see the comparison between different algorithms which is saved in a excel file.

6.1 Interface

Figure 6.1 represents the interface of our visualization software. Here we provides three options to the users.

- 1- Run: to run the program to allow all of our variants to execute,
- 2- View: to view floorplan,
- 3- Compare: to compare results.

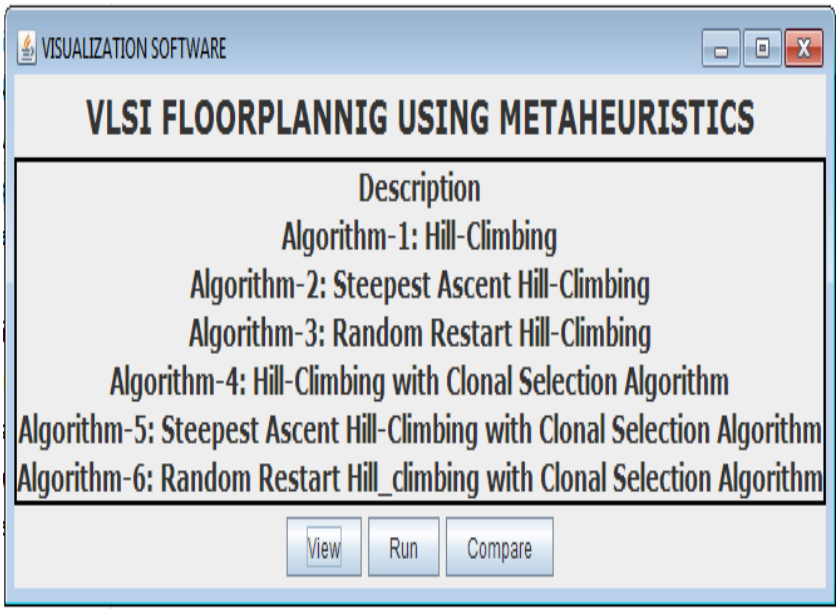


Figure 6.1: Interface of our visualization software

6.2 Run

Figure 6.3 shows the action after running the program from the interface. When an user click 'run' button, then an option widow is opened for the user. Option window is shown in Figure 6.2. The user selects his desired algorithm to execute and then the selected algorithms run using all the benchmarks and save the outputs into the appropriate folders. The output file is a text file and its name is auto generated as date-time format as shown in Figure 6.4, so that user can find the output file so easily.

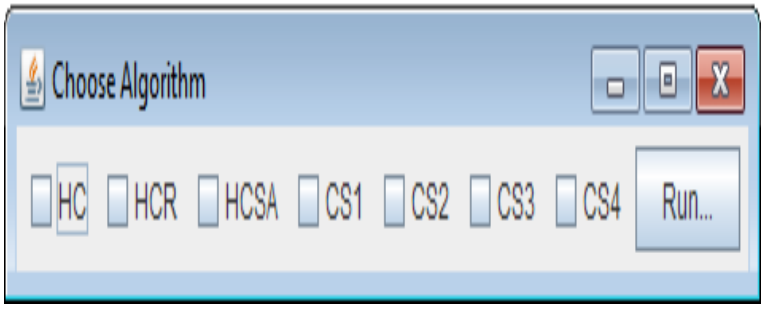


Figure 6.2: The option window to select algorithms to run

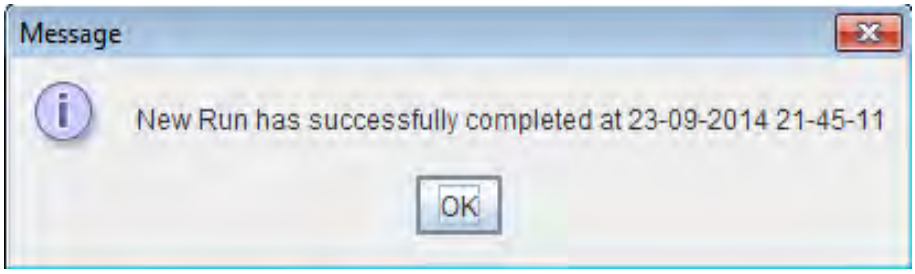


Figure 6.3: The action after running the program

6.3 View

When an user click 'View' button, then they get a window to choose a file to view its floorplan. Figure 6.5 shows that how an user will get his option to view a specific floorplan. Figure 6.6 presents a sample floorplan from our software.

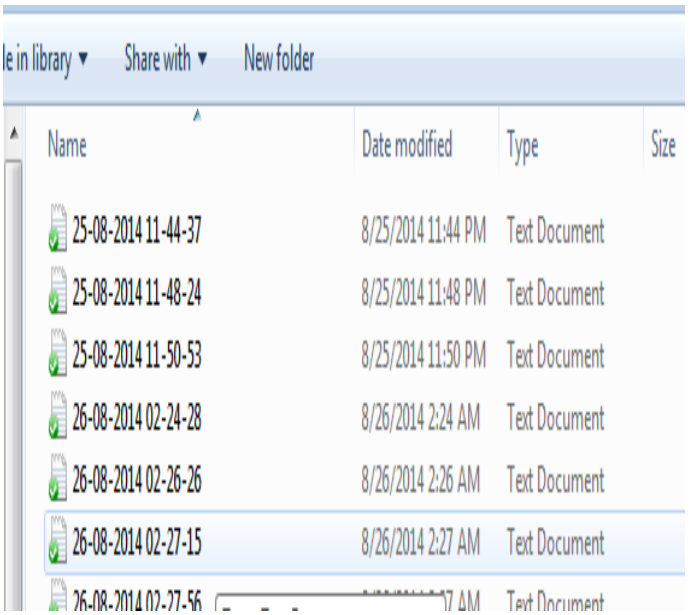


Figure 6.4: The format of the output file

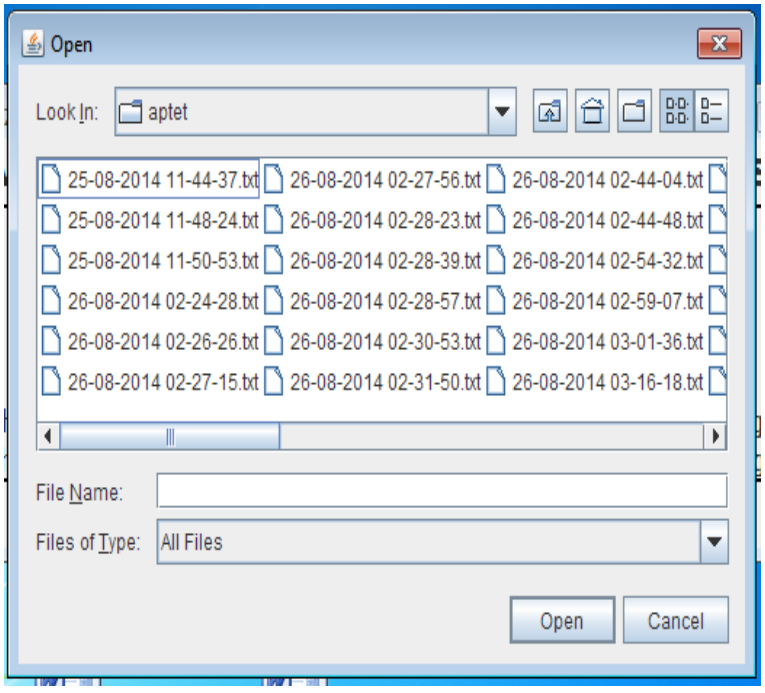


Figure 6.5: The file choosing window to view floorplan

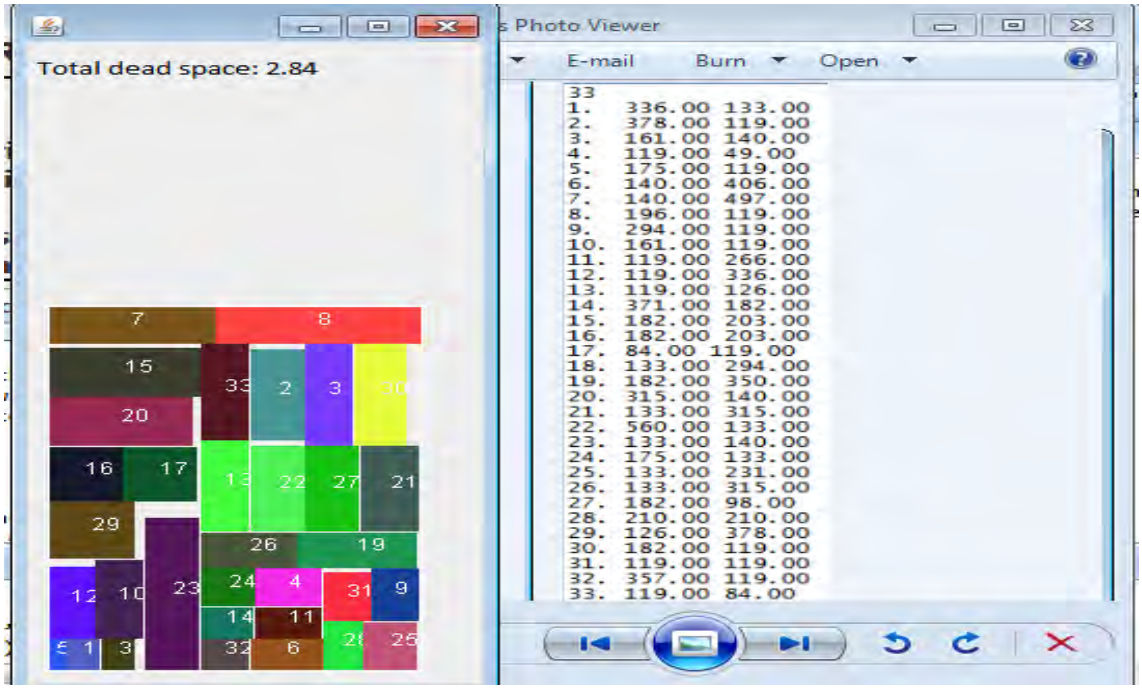
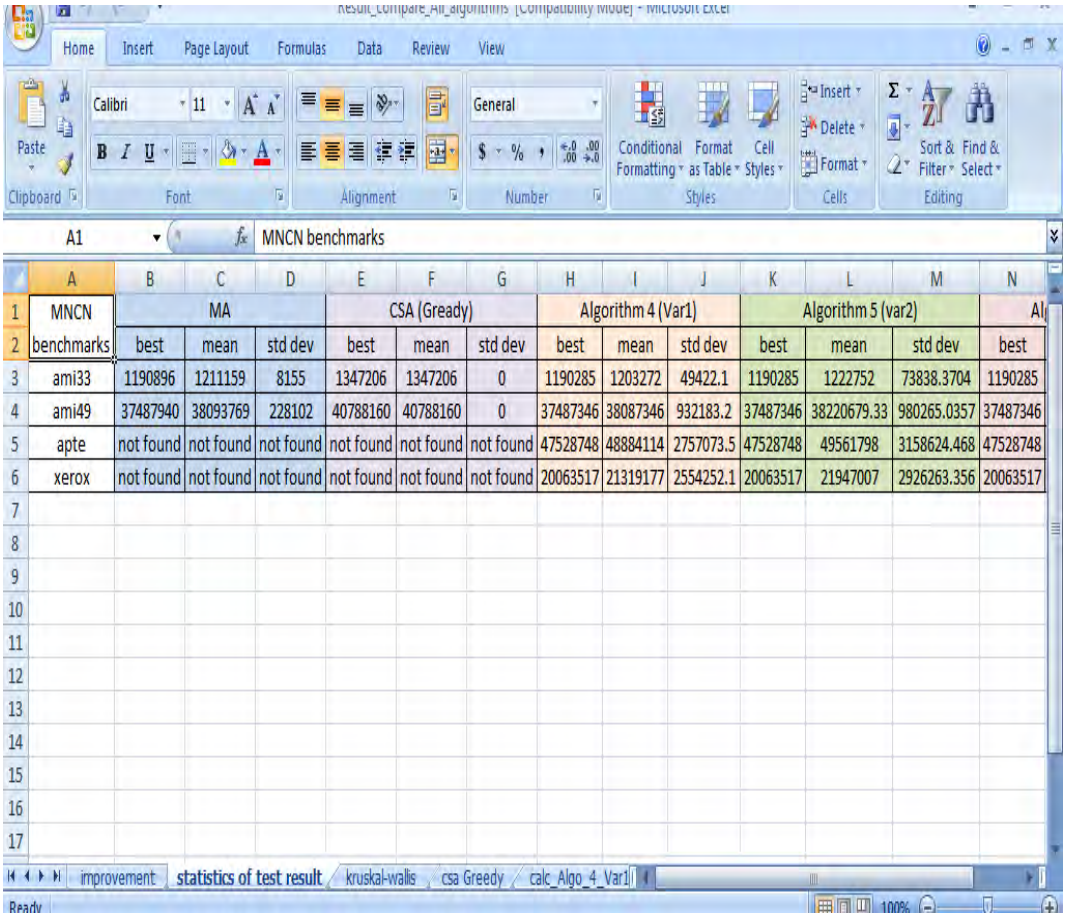


Figure 6.6: A sample floorplan drawn by our software

6.4 Compare

We have done statistical test and Kruskal-Wallis test for all comparing algorithms as well as for all the variants of our algorithms. By this option user can view all the comparison result by opening that excel file. Figure 6.7 shows the comparing data which is shown after selecting this option by a user.

This software is produced as an initial step by us to manipulate VLSI floorplan. We have a plan to develop this software in future to compute more results, show floorplan in a better way and compare different results dynamically in this software. There is no online manipulation software for VLSI design. Our first step for VLSI floorplanning visualization software can help the researchers to give importance in this section and produce more useful software for the users.



MNCN benchmarks	MA			CSA (Gready)			Algorithm 4 (Var1)			Algorithm 5 (var2)			Al
	best	mean	std dev	best	mean	std dev	best	mean	std dev	best	mean	std dev	best
ami33	1190896	1211159	8155	1347206	1347206	0	1190285	1203272	49422.1	1190285	1222752	73838.3704	1190285
ami49	37487940	38093769	228102	40788160	40788160	0	37487346	38087346	932183.2	37487346	38220679.33	980265.0357	37487346
apte	not found	not found	not found	not found	not found	not found	47528748	48884114	2757073.5	47528748	49561798	3158624.468	47528748
xerox	not found	not found	not found	not found	not found	not found	20063517	21319177	2554252.1	20063517	21947007	2926263.356	20063517

Figure 6.7: Comparison result between all algorithms

Chapter 7
Conclusion

7.1 Conclusion

Floorplanning is an essential design step for hierarchical, building-module design methodology. Floorplanning provides early feedback that evaluates architectural decisions, estimates chip areas, and estimates delay and congestion caused by wiring. As technology advances, design complexity is increasing and the circuit size is getting larger. To cope with the increasing design complexity, hierarchical design and intellectual property (IP) modules are widely used. This trend makes floorplanning much more critical to the quality of a very large-scale integration (VLSI) design than ever.

A number of metaheuristic algorithms have been presented to solve nonslicing and hard-module VLSI floorplanning problem. An effective method (i.e., CSA) with mutation has been used by these algorithms to explore the search space and a number of single state methods separately to exploit the search region. Mutation has been applied as it was applied in [56]. The algorithms have been implemented and tested on the available benchmark data-sets in the literature to empirically study the performance of the algorithms. These benchmarks include the two popular MCNC benchmark problems for the VLSI floorplanning prooblem: ami33, ami49, apte and xerox [2]. GSRC is another popular benchmark which has also been used in our experiment. Experimental results have shown that Clonal Selection Algorithm with Hill-Climbing can produce better solutions for all the benchmarks than other algorithms. We have compared our results with previous works. To compare results Kruskal-Wallis test and statistical test have been used, which have been presented as tabular form in this thesis.

Visualization software has also been developed to solve VLSI floorplanning. An algorithm from several algorithms can be selected by this software for a particular run. With the help of this software, any floorplan which is saved in a previous run can be visualized. For a specific variant of our algorithm and for a specific benchmark from the available benchmarks, output is saved by this software, which is the best in a particular run. Output from this software can be visualized and compared with all other results.

7.2 Suggestion for future work

In our thesis, we limit our objective function to the construction of a floorplan of minimum area. Other elements, such as the minimization of the total wire length, will be included in the cost function at a later date and extend to much larger problems.

We have implemented our algorithm for non-slicing hard module floorplanning problem. But, further research can include these algorithms for slicing and / or soft module floorplanning problem. In recent years, $3D$ VLSI floorplanning problems have become very popular. So, our algorithms can be applied and tested for those new problems too.

To complement our algorithms, we have written a supporting software that checks the final floorplan for correctness and draws a picture of the layout. Our visualization software is produced to manipulate VLSI floorplan and it is very new idea in this sector. We have a plan to develop this software in future to compute more results, show floorplans in a better way and compare different results dynamically from the software. There is no online manipulation software for VLSI design, so in future we will try to make a online visualization software. Our initial step for VLSI floorplanning visualization software can help the researchers to give importance in this sector and produce more useful software for the users.

REFERENCES

[1] The analyse-it general statistics software for microsoft excel. Last accessed on August 04, 2014, at 08:12:00PM. [Online]. Available: http://www.analyse-it.com/.

[2] The MCNC benchmark problems for VLSI floorplanning. Last accessed on July 21, 2014, at 02:08:00PM. [Online]. Available: https://www.mcnc.org/.

[3] Deen Md Abdullah, Wali Md Abdullah, Noor Mohammad Babu, Md Bhuiyan, Momenul Islam, Kazi Munshimun Nabi, and M Sohel Rahman. VLSI floorplanning design using clonal selection algorithm. In *Informatics, Electronics & Vision (ICIEV), 2013 International Conference on*, pages 1–6. IEEE, 2013.

[4] Hakim Salah Adiche. *Fuzzy genetic algorithm for VLSI floorplan design.* PhD thesis, King Fahd University of Petroleum and Minerals, 1997.

[5] Uwe Aickelin and Dipankar Dasgupta. Artificial immune systems. In *Search Methodologies*, pages 375–399. Springer, 2005.

[6] Jamal R Al-Enezi, Maysam F Abbod, and Salah Al Sharhan. Artificial immune systems-models, algorithms and applications. *International Journal*, 3(2), 2010.

[7] Mauro Birattari, Thomas Stutzle, Luis Paquete, Klaus Varrentrapp, et al. A racing algorithm for configuring metaheuristics. In *GECCO*, volume 2, pages 11–18. Citeseer, 2002.

[8] Jason Brownlee. Clonal selection theory & clonalg–the clonal selection classification algorithm (CSCA). *Swinburne University of Technology*, 2005.

[9] Jason Brownlee. Clonal selection algorithms. Technical report, Complex Intelligent Systems Laboratory, Swinburne University of Technology, Australia, 2007.

[10] Leandro Nunes De Castro and Jonathan Timmis. *Artificial immune systems - a new computational intelligence paradigm*. Springer, 2002.

[11] Hayward H Chan, Saurabh N Adya, and Igor L Markov. Are floorplan representations important in digital design? In *Proceedings of the 2005 international symposium on Physical design*, pages 129–136. ACM, 2005.

[12] Yun-Chih Chang, Yao-Wen Chang, Guang-Ming Wu, and Shu-Wei Wu. B*-Trees: a new representation for non-slicing floorplans. In *Proceedings of the 37th Annual Design Automation Conference*, pages 458–463. ACM, 2000.

[13] Guolong Chen, Wenzhong Guo, and Yuzhong Chen. A PSO–based intelligent decision algorithm for VLSI floorplanning. *Soft Computing*, 14(12):1329–1337, 2010.

[14] Carlos A Coello Coello, Daniel Cortes Rivera, and Nareli Cruz Cortes. Use of an artificial immune system for job shop scheduling. In *Artificial Immune Systems*, pages 1–10. Springer, 2003.

[15] James P Cohoon, Shailesh U Hegde, Worthy N Martin, and Dana S Richards. Distributed genetic algorithms for the floorplan design problem. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 10(4):483–492, 1991.

[16] Thelma Elita Colanzi, Wesley Klewerton Guez Assuncao, Aurora Trinidad Ramirez Pozo, Ana Cristina B Kochem Vendramin, Diogo Augusto Barros Pereira, Carlos Alberto Zorzo, Alto Vale do Rio do Peixe, and Pedro Luiz De Paula Filho. Application of bio-inspired metaheuristics in the data clustering problem authors and affiliations. *CLEI Electronic Journal*, 14(3), 2011.

[17] Dipankar Dasgupta. *Artificial immune systems and their applications*, volume 1. Springer, 1999.

[18] Dipankar Dasgupta. An overview of artificial immune systems and their applications. In *Artificial immune systems and their applications*, pages 3–21. Springer, 1999.

[19] Dipankar Dasgupta. Advances in artificial immune systems. *Computational Intelligence Magazine, IEEE*, 1(4):40–49, 2006.

[20] Dipankar Dasgupta, Nivedita Sumi Majumdar, and Fernando Nino. Artificial immune systems: A bibliography. *Computer Science Division, University of Memphis, Technical Report*, 2007.

[21] Dipankar Dasgupta and Fernando Nino. *Immunological computation: theory and applications*. CRC Press, 2008.

[22] Leandro Nunes De Castro and Fernando Jose Von Zuben. Artificial immune systems: Part I - basic theory and applications. Technical report, Universidade Estadual de Campinas, Dezembro de, 1999.

[23] Leandro Nunes De Castro and Fernando Jose Von Zuben. Artificial immune systems: Part ii–a survey of applications. Technical report, RTDCA 02, 2000a.

[24] Leandro Nunes De Castro and Fernando Jose Von Zuben. Artificial immune systems: Part ii–a survey of applications. Technical report, RTDCA 02, 2000b.

[25] Leandro Nunes De Castro and Fernando Jose Von Zuben. Learning and optimization using the clonal selection principle. *Evolutionary Computation, IEEE Transactions on*, 6(3):239–251, 2002.

[26] Alex Dickinson. Floyd: A knowledge based floor plan designer. In *Proceedings of IEEE International Conference on Computer Design*, pages 176–179. IEEE, 1986.

[27] Pradeep Fernando and Srinivas Katkoori. An elitist non-dominated sorting based genetic algorithm for simultaneous area and wirelength minimization in VLSI floorplanning. In *VLSI Design, 2008. VLSID 2008. 21st International Conference on*, pages 337–342. IEEE, 2008.

[28] Luca Maria Gambardella, Fabio De Luigi, and Vittorio Maniezzo. Ant colony optimization. *Istituto Dalle Molle di Studi sullIntelligenza Artificiale Galleria*, 2, 1998.

[29] Simon M Garrett. How do we evaluate artificial immune systems? *Evolutionary Computation*, 13(2):145–177, 2005.

[30] Maoguo Gong, Licheng Jiao, Lining Zhang, and Wenping Ma. Improved real-valued clonal selection algorithm based on a novel mutation method. In *Intelligent Signal Processing and Communication Systems, 2007. ISPACS 2007. International Symposium on*, pages 662–665. IEEE, 2007.

[31] Pei-Ning Guo, Chung-Kuan Cheng, and Takeshi Yoshimura. An O-tree representation of non-slicing floorplan and its applications. In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, pages 268–273. ACM, 1999.

[32] Pei-Ning Guo, Toshihiko Takahashi, Chung-Kuan Cheng, and Takeshi Yoshimura. Floorplanning using a tree representation. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 20(2):281–289, 2001.

[33] Bah-Hwee Gwee and Meng-Hiot Lim. A GA with heuristic-based decoder for IC floorplanning. *Integration, the VLSI journal*, 28(2):157–172, 1999.

[34] Emma Hart and Jon Timmis. Application areas of AIS: The past, the present and the future. *Applied soft computing*, 8(1):191–201, 2008.

[35] Steven A Hofmeyr and Stephanie Forrest. Architecture for an artificial immune system. *Evolutionary computation*, 8(4):443–473, 2000.

[36] John Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[37] Chyi-Shiang Hoo, Jeevan Kanesan, and Harikrishnan Ramiah. Enumeration technique in very large-scale integration fixed-outline floorplanning. *IET Circuits, Devices & Systems*, 8(1):47–57, 2014.

[38] John E Hunt and Denise E Cooke. Learning using an artificial immune system. *Journal of network and computer applications*, 19(2):189–212, 1996.

[39] Ajoy Kumar Khan, Rahul Vatsa, Sudipta Roy, and Bhaskar Das. A new efficient topological structure for floorplanning in 3d VLSI physical design. In *Advance Computing Conference (IACC), 2014 IEEE International*, pages 696–701. IEEE, 2014.

[40] Minghorng Lai and D Wong. Slicing tree is a complete floorplan representation. In *Proceedings of the conference on Design, automation and test in Europe*, pages 228–232. IEEE Press, 2001.

[41] Chang-Tzu Lin, De-Sheng Chen, and Yi-Wen Wang. An efficient genetic algorithm for slicing floorplan area optimization. In *Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium on*, volume 2, pages II–879. IEEE, 2002.

[42] Chang-Tzu Lin, De-Sheng Chen, and Yi-Wen Wang. Robust fixed-outline floorplanning through evolutionary search. In *Proceedings of the 2004 Asia and South Pacific Design Automation Conference*, pages 42–44. IEEE Press, 2004.

[43] Sean Luke. *Essentials of Metaheuristics*. Lulu, 2009.

[44] Chaomin Luo. *Novel Convex Optimization Approaches for VLSI Floorplanning*. PhD thesis, Citeseer, 2008.

[45] Yuchun Ma, Sheqin Dong, Xianiong Hong, Yici Cai, Chung-Kuan Cheng, and Jun Gu. VLSI floorplanning with boundary constraints based on corner block list. In *Proceedings of the 2001 Asia and South Pacific Design Automation Conference*, pages 509–514. ACM, 2001.

[46] John H McDonald. *Handbook of biological statistics*, volume 2. Sparky House Publishing Baltimore, MD, 2009.

[47] Carver Mead and Mohammed Ismail. *Analog VLSI implementation of neural systems*. Springer, 1989.

[48] Dinesh P Mehta and Sartaj Sahni. *Handbook Of Data Structures and Applications*. CRC Press, 2004.

[49] Gotaro Odawara, Kazuhiko Iijima, and Kazutoshi Wakabayashi. Knowledge-based placement technique for printed wiring boards. In *Proceedings of the 22nd ACM/IEEE Design Automation Conference*, pages 616–622. IEEE Press, 1985.

[50] Ibrahim H Osman and James P Kelly. *Meta-heuristics: theory and applications*. Springer, 1996.

[51] Maurizio Rebaudengo and Matteo Sonza Reorda. GALLO: A genetic algorithm for floorplan area optimization. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 15(8):943–951, 1996.

[52] Rob A Rutenbar. Simulated annealing algorithms: An overview. *Circuits and Devices Magazine, IEEE*, 5(1):19–26, 1989.

[53] Sadiq M Sait and Habib Youssef. *VISI Physical Design Automation: Theory and Practice*. McGraw-Hill, Inc., 1994.

[54] P Sivaranjani and KK Kawya. Performance analysis of VLSI floor planning using evolutionary algorithm. In *IJCA Proceedings on International Conference on Innovations In Intelligent Instrumentation, Optimization and Electrical Sciences*, volume ICIIIOES 9, pages 42–46. Foundation of Computer Science (FCS), 2013.

[55] Susmita Sur-Kolay and Bhargab B Bhattacharya. Inherent nonslicibility of rectangular duals in VLSI floorplanning. In *Foundations of Software Technology and Theoretical Computer Science*, pages 88–107. Springer, 1988.

[56] Maolin Tang and Xin Yao. A memetic algorithm for VLSI floorplanning. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 37(1):62–69, 2007.

[57] Isao Tazawa, Seiichi Koakutsu, and Hironori Hirata. An immunity based genetic algorithm and its application to the VLSI floorplan design problem. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 417–421. IEEE, 1996.

[58] Jon Timmis. Artificial immune systemstoday and tomorrow. *Natural computing*, 6(1):1–18, 2007.

[59] Jon Timmis, Mark Neal, and John Hunt. An artificial immune system for data analysis. *Biosystems*, 55(1):143–150, 2000.

[60] Jonathan Timmis, Andrew Hone, Thomas Stibor, and Edward Clark. Theoretical advances in artificial immune systems. *Theoretical Computer Science*, 403(1):11–32, 2008.

[61] Christine L Valenzuela and Pearl Y Wang. A genetic algorithm for vlsi floorplanning. In *Parallel Problem Solving from Nature PPSN VI*, pages 671–680. Springer, 2000.

[62] Christine L Valenzuela and Pearl Y Wang. VLSI placement and area optimization using a genetic algorithm to breed normalized postfix expressions. *Evolutionary Computation, IEEE Transactions on*, 6(4):390–401, 2002.

[63] Laung-Terng Wang, Yao-Wen Chang, and Kwang-Ting Tim Cheng. *Electronic design automation: synthesis, verification, and test*. Morgan Kaufmann, 2009.

[64] Andrew Watkins, Jon Timmis, and Lois Boggess. Artificial immune recognition system (AIRS): An immune-inspired supervised learning algorithm. *Genetic Programming and Evolvable Machines*, 5(3):291–317, 2004.

[65] Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

[66] Wayne Wolf. *Modern VLSI design: system-on-chip design*. Pearson Education, 2002.

[67] Eugene Y. C. Wong and Henry Y. K. Lau. Advancement in the twentieth century in artificial immune systems for optimization: review and future outlook. In *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, pages 4195–4202. IEEE, 2009.

[68] Zheng Shiqin Wang Xiufeng. An improved clonal selection algorithm for multi-modal function optimization. *Computer Engineering and Applications*, 3:004, 2006.

[69] Evangeline F. Y. Young, Chris C. N. Chu, and M. L. Ho. Placement constraints in floorplan design. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 12(7):735–745, 2004.

[70] Jieqiong Zheng, Yunfang Chen, and Wei Zhang. A survey of artificial immune applications. *Artificial Intelligence Review*, 34(1):19–34, 2010.