Evolutionary Approach for Solving Standard Cell Placement In VLSI Physical Design Automation

Thesis submitted in partial fulfillment of the requirements of the award of degree of

Master of Technology

in

Computer Science and Applications

Submitted By

Jobanpreet Kaur

Roll No: 601103010

Supervised By

Ms. Maninder Kaur

Assistant Professor (SMCA)

SCHOOL OF MATHEMATICS AND COMPUTER APPLICATIONS

THAPAR UNIVERSITY

PATIALA – 147004

July 2013

Certificate

I hereby certify that the work being presented in the thesis entitled, *"Evolutionary Approach For Standard Cell Placement in VLSI Physical Design Automation"*, in partial fulfillment of the requirements for the award of degree of Master of Technology in Computer Science and Applications submitted to the School of Mathematics and Computer Applications of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of ***Ms. Maninder Kaur*** and refers other researcher's works which are duly listed in the reference section.

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

Johanpreet Kaur

(Johanpreet Kaur)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Maninder Kaur

(Ms. Maninder Kaur)

Assistant Professor,
School of Mathematics and Computer Applications,
Thapar University, Patiala

Countersigned By

Dr. Rajesh Kumar

Head,
School of Mathematics and Computer Applications,
Thapar University, Patiala

Dr. S.K. Mohapatra

Dean (Academic Affairs)
Thapar University, Patiala

Acknowledgment

First and foremost I would like to thank God for providing me with faith, self-confidence and ability to complete this thesis.

It would be a great pleasure for me to acknowledge the guidance, assistance and help received from **Ms. Maninder Kaur**, Assistant professor, SMCA, Thapar University, Patiala. I am very grateful for her continual support, encouragement and invaluable suggestions.

I extend my deep gratitude towards **Dr. Rajesh Kumar**, Head, SMCA, Thapar University, Patiala for providing the timely facilities and infrastructure that helped in completion of the thesis.

I would like to take this opportunity to express my sincere appreciation and thanks to my friends and family for their strong moral support and constant encouragement throughout this research. I am also thankful to all staff members of SMCA for their kind cooperation and sincere help.

Jobanpreet Kaur
Roll No: 601103010
M.Tech (CSA)

Abstract

The increased complexity in VLSI cell placement due to increase in number of transistors requires the application of various heuristic approaches including evolutionary approaches for obtaining better results. Evolutionary approaches are able to produce high quality placement solutions for standard cell circuits in comparison to other sophisticated algorithms. This work proposes an algorithm named, Evolutionary Approach for Standard Cell Placement problem (EASCP). The proposed algorithm follows a variant of genetic paradigm where a trial value is associated with each candidate solution of the population to improve the quality of solution by replacing the abandoned solutions. With a key feature of exploring the solution space, the proposed EASCP uses a repository for keeping track of the best solutions per generation, thereby, exploiting the solution space. The algorithm was simulated on test circuits. The experimental results obtained a layout with better wirelength in comparison to simple genetic algorithm at the cost of fractional increase in runtime.

Table of Contents

List of Abbreviations

	Abbreviation	Details
1.	EA	Evolutionary Algorithms
2.	SCP	Standard Cell Placement
3.	GA	Genetic Algorithm
4.	VLSI	Very Large Scale Integration
5.	EASCP	Evolutionary Approach for Standard Cell Placement
6.	GEA	Genetic-Evolutionary Algorithm
7.	TS	Tabu Search
8.	SA	Simulated Annealing
9.	HPWL	Half Perimeter WireLength
10.	PMX	Partially Mapped Crossover

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION

VLSI refers to a technology through which it is possible to implement large circuits in silicon i.e. circuits consisting of large number of transistors. The VLSI technology has been successfully used to build microprocessors, signal processors, systolic arrays, large capacity memories, memory controllers and interconnection networks. The complexity of VLSI being designed and used nowadays has made the manual approach of design impractical hence making design automation crucial. With the rapid development of technology in the last two decades, the status of VLSI technology is described by the following characteristics [8]:

- There has been an increase in the size and functionality of the integrated circuits.
- The feature size is reduced and the speed of operation is increased.
- The probability of predicting the circuit behavior is improved.

With the increase in the complexity of the circuit, it is not very essential to save on transistors; the cost reduction through logic minimization is likely to be insignificant when the total number of transistors is in order of a million. On the other hand, it is essential to save on interconnection costs, since wires are more expensive in VLSI than transistors. This whole technology of VLSI goes through many phases.

1.1 Phases of VLSI Design Process

The various phases of VLSI design are shown in figure 1.1. *Architectural Design* is carried out by expert engineers and it affects the cost and design of the chip considerably. In *logical design*, the architectural description is mapped into logic components generally gates (OR, AND, NOT etc.) that are connected to each other. *Physical design* of a circuit is the phase that precedes the fabrication of a circuit. Based on the logical design, a physical design description is generated. Physical design is the process of converting the specification of a circuit into the geometric description of a layout [6]. Due to the large number of components, physical design is an exceptionally complex process [7]. It is usually broken down into a number of stages, such as partitioning, placement and routing. In general, partitioning divides a large system into a set of small subsystems. Placement is done by placing the modules on the chip by satisfying certain constraints and finally routing determines how the wires will connect the modules.

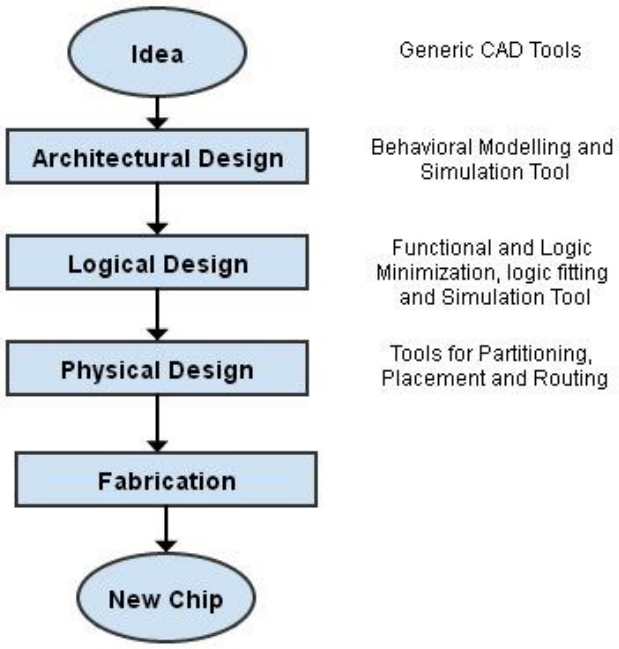


Figure 1.1 [8] Various Phases of VLSI Design Process

The proposed research work focusses on the problem of *Standard Cell Placement* in VLSI physical design. The cell placement problem is to position the modules on the real-estate such that some form of objective function is minimized. The objective function for the problem is generally one or several of the following: minimize estimated length of longest wire, minimize estimated sum of wirelength, minimize time delay or minimize layout area.

1.2 Basic Terminology

Some of the basic terminology [8] used in the work is discussed in this section.

A *cell* is a basic building block that is used to build larger circuits. A cell is generally a combination of multiple input gates and flip flops. The term *standard cell* refers to those cells whose design has been standardized in some manner. The cells may have a standard height or a standard pin structure. A *net* is defined by a collection of pins that must be electrically connected. A *netlist* is a list of all nets in the circuit. There may be *weighted nets* in the circuit. The weighted nets are indicated in the net list of the circuit where a positive *weight* is associated with each net that defines the magnitude of criticality of the net. The circuit specifications of a netlist have to be converted into a connectivity matrix. Consider the following circuit in figure 1.2 as an example circuit.

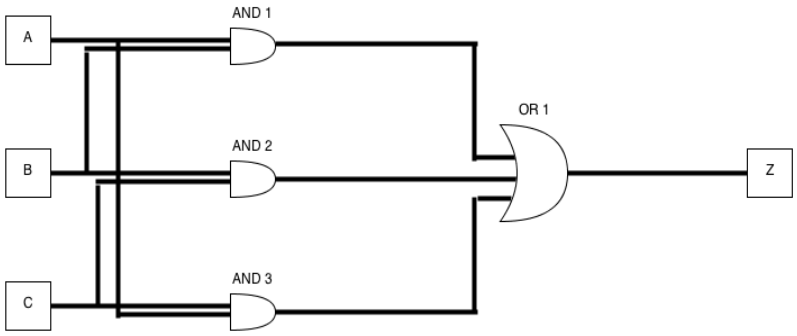


Figure 1.2 Example Circuit

This circuit has three inputs as A, B and C. There are three AND gates namely AND1, AND2 and AND3. The output of three AND gates is passed to one OR gate (OR1) that generates the final output Z. For this circuit, a *connectivity graph* is generated as shown in figure 1.3. The connectivity graph is more simple and convenient to work with than the original netlist.

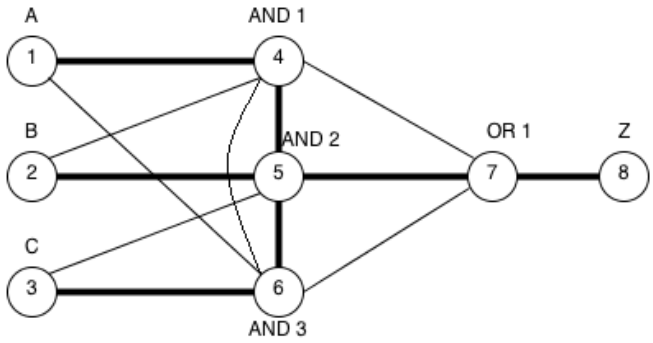


Figure 1.3 Connectivity Graph for the Circuit

The nodes represent the modules, input pads and output pads and the edges represent the connections among them. After the connectivity graph, a *connectivity matrix* is constructed.

Nodes/ Nodes	1	2	3	4	5	6	7	8
1	0	0	0	1	0	1	0	0
2	0	0	0	1	1	0	0	0
3	0	0	0	0	1	1	0	0
4	1	1	0	0	1	1	1	0
5	0	1	1	1	0	1	1	0
6	1	0	1	1	1	0	1	0
7	0	0	0	1	1	1	0	1
8	0	0	0	0	0	0	1	0

Figure 1.4 Connectivity Matrix

This matrix is an n x n matrix where n is the total number of nodes in the graph. The connectivity matrix is symmetric such that

$$c_{ij} = c_{ji} \qquad where \;\; i, j = 1, 2, \ldots, n$$

The matrix contains the values 0 and 1. The value 1 indicates that there exists a connection between the corresponding modules and the value 0 indicates that there is no connection. The connectivity matrix is shown in figure 1.4.

1.3 Need for Heuristic Methods for Standard Cell Placement Problem

The work focusses on solving the problem of Standard Cell Placement in VLSI physical design process. The placement of cells such that the total wirelength is minimized is an NP-complete problem [11]. NP-complete problems are very hard to solve and could not be solved in polynomial time [2]. The number of cells to be placed is very large in number. Therefore, it is not feasible in this case to take into account the basic brute force method of considering all the placements and then selecting the best one. For such problems that would take years to complete when exact methods are applied, metaheuristics are needed. For combinatorial optimization problems (for which it is known that a polynomial time solution exists but is not practical), metaheuristics are required. It means some advanced heuristic techniques are required that could be applied on such problems to evolve towards a better solution. Though these techniques do not promise the best solution to the problem, yet they generate an optimal good solution that could evolve to a better solution.

The problem of Standard Cell Placement has been widely studied by the researchers since the last few decades and a number of metaheuristic techniques have been developed for solving the cell placement problem [3], [4], [5]. One of the approaches in metaheuristics for solving Standard Cell Placement (SCP) problem is an evolutionary approach. Evolutionary Algorithms (EA) are a class of stochastic optimization methods that simulate the process of natural evolution [9]. Though these algorithms have a simple underlying principle, yet they are robust, efficient and powerful for solving combinatorial optimization problems. Some of the popular evolutionary algorithms are Artificial Bee Colony (ABC) algorithm, Bat algorithm, Genetic Algorithm (GA), Particle Swarm Optimization (PSO). These algorithms have shown good results and some of the hybrid approaches using these algorithms have shown better results.

The most common approach used by the researchers is genetic algorithms for solving SCP. Various versions of genetic approach have been used to solve the placement problem that has shown

improved results. However, the main problem with evolutionary approaches is that the solution space is not explored fully and the solution may get stuck in local optima. The proposed work focusses on this problem issue to explore the solution space fully.

1.4 Outline of the Thesis

The further organization of the thesis is as follows. There are four remaining chapters:

Chapter 2. Literature Survey

In this chapter various techniques to solve standard cell placement problem using evolutionary approaches have been presented discussing their limitations and literature survey with relevance of the research work has been highlighted. Also, this chapter discusses the various research papers related to the work that have been studied and research gaps in the previous work. This chapter is the motivation of this research work.

Chapter 3. Problem Statement

This chapter gives the problem definition for standard cell placement discussing the various wirelength estimation methods.

Chapter 4. Work Done

This chapter is the backbone of this research that contains all the functions and algorithms that have been developed for this study. The proposed evolutionary algorithm with the fitness function is explained in this chapter. The chapter presents the data sets that include sample circuits with results. Various tables and graphs have been included in this chapter that gives the proof of this research work.

Chapter 5. Conclusion and Future Scope

This chapter is devoted for the discussion with conclusion and the extension of this research work.

1.5 Summary

The chapter gives the introduction about the background of the topic and the also discusses the basic terminology related to the topic. The intensity and depth of the problem was presented and the need to develop the heuristic approaches for solving SCP was discussed. The outline of the thesis was presented that briefly describes the organization of the thesis.

CHAPTER 2

LITERATURE SURVEY

2.1 Introduction

The standard cell placement problem is to place the cell modules on a fixed size chip such that certain constraints are fulfilled. Cell placement has been a crucial stage in VLSI physical design process and has been studied since many decades. Cell placement is an NP-complete problem, and thus, it cannot be solved in polynomial time [13]. Trying to get an exact solution by brute force techniques in order to find the best solution would take time proportional to the factorial of the number of cells. Hence, it is not possible to use this method for circuits with any reasonable number of cells. Searching through a large number of candidate placement configurations efficiently requires heuristic algorithms. The heuristic techniques find a good placement with wirelength close to the minimum, but with no guarantee of achieving the absolute minimum [12]. Various heuristic techniques including the evolutionary approaches have been developed for such combinatorial optimization problems.

2.2 VLSI Standard Cell Placement

Standard cells are logical modules having fixed heights and varied widths. These cells are placed in rows with routing channels between rows used for interconnects [1]. For a given circuit, consisting of a set of cell modules and a net list giving the interconnections between these modules, the standard cell placement problem is providing a layout giving the positions of the cell modules in parallel rows such that all the nets are interconnected using wires and the total layout area is minimized [10].

An acceptable placement is the one that is physically feasible i.e. the cells do not overlap with each other, they are placed within the boundaries and the cells should be restricted to rows in predetermined locations. The cost function in SCP is the sum of the total wire length (which is a widely used measure of the quality of placement) and other constraints such as number of crossings of sensitive nets, total unused space at the end of each row, total chip area, time delay etc. However, the main objective of a placement algorithm is to determine a feasible placement with the minimum possible cost. To illustrate the above points, an example circuit is shown in figure 2.1.

The figure shows an electrical circuit containing 8 functional standard cells. The functional area for the cells in a placement remains same; it is the wiring area that changes with placement.

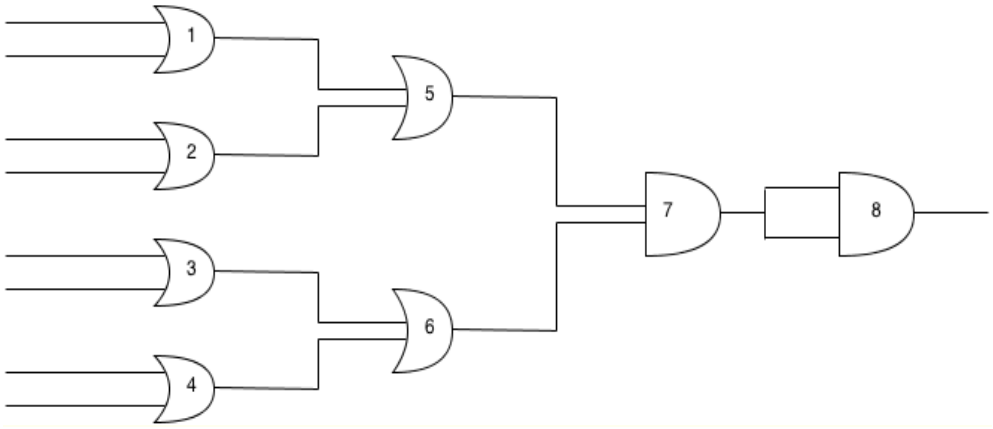


Figure 2.1 Example of a Simple Circuit

The input for the SCP problem consists of a set of cell modules and a netlist. The output is a layout giving the cell positions. Figure 2.2 and 2.3 shows two placements for the circuit shown in figure 2.1 with wirelength 10 and 12 respectively. The optimal placement would be the one having minimum wirelength.

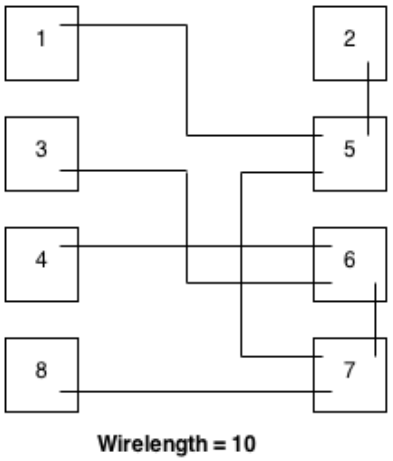


Figure 2.2 2-D Placement (Wirelength = 10)

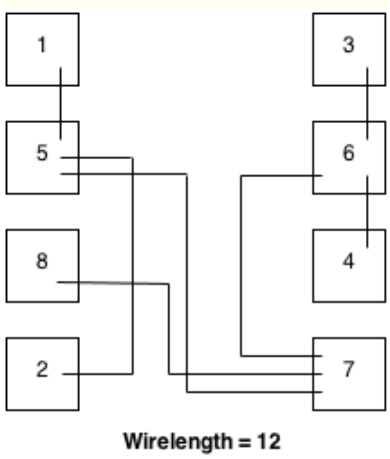


Figure 2.3 2-D Placement (Wirelength = 12)

2.2.1 Cell Placement Algorithms

There are a number of placement algorithms. These algorithms have been broadly classified as follows:

- *Constructive Algorithms:* In these algorithms, a placement is built from the scratch and once the cell position is fixed, it is not modified anymore. Typically, a random seed module is selected and placed in the chip layout area. Then other modules are selected one by one in order

of their connectivity to the placed modules and are placed close to the already placed modules such that the total estimated wirelength is minimized. Such algorithms are generally fast in nature but they produce poor layouts. These algorithms are therefore used for generating an initial placement for iterative improvement algorithms [1]. The main reason for their use is their speed. Min-Cut Placement algorithm is an example of constructive algorithm.

- *Iterative Improvement Algorithms:* These algorithms are used to change intermediate placements in order to improve the cost function. Iterative improvement algorithms generally produce good placements but require large computation time. The basic iterative improvement approach interchanges randomly selected pairs of modules and accepts the interchange if it results in a reduction in cost [23]. The algorithm halts when there is no further improvement of placements in terms of cost function during a given large number of trials. Some of the popular iterative improvement algorithms include simulated annealing and genetic algorithms [1].

2.2.2 Related Study

The research work in [1] discussed the various issues in VLSI cell placement techniques. The authors presented a comprehensive survey of the various cell placement techniques emphasizing on standard ce11 and macro placement. Five major algorithms for placement were discussed namely, simulated annealing, force-directed placement, placement by numerical optimization, min-cut placement and evolution-based placement.

The work presented in [14] was focused on modern VLSI design challenges and complex placement constraints. The authors presented example techniques to handle the challenges coming from large-scale mixed-size circuit designs with the wirelength optimization.

The authors in [15] discussed the research studies that address various aspects of global and detailed placement in VLSI design. The survey of the history of placement research, the progress achieved so far in the field and the outstanding challenges in VLSI was also described in the work. The authors addressed various issues in cell placement including wirelength estimation methods (HPWL), timing and power-driven placement etc. in the work.

2.3 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are a population-based type of optimization algorithm that mimic the image of natural biological evolution and are used to solve complex problems. These algorithms use biological principles namely selection, crossover, mutation and survival of the fittest [16].

Evolutionary algorithms operate on a population of probable solutions, applying the principle of survival of the fittest to produce better approximations to a solution. In evolutionary algorithms, natural selection is simulated by a stochastic selection process. Each candidate solution is provided a chance to breed a certain number of times dependent on their quality. Quality is thus assessed by evaluating the individuals and assigning them scalar fitness values. This process leads to the evolution of populations of individuals that are better suited to their environment than their parents similar to natural evolution. In an EA, a number of artificial creatures search over the space of the problem. They compete repeatedly with each other to find out optimal areas of the search space. It is hoped that over time the most successful of these creatures will evolve to find out the best solution. The EAs make a few assumptions regarding the fitness which is an advantage in comparison to other optimization methods [17].The working of evolutionary algorithms is shown in figure 2.4.

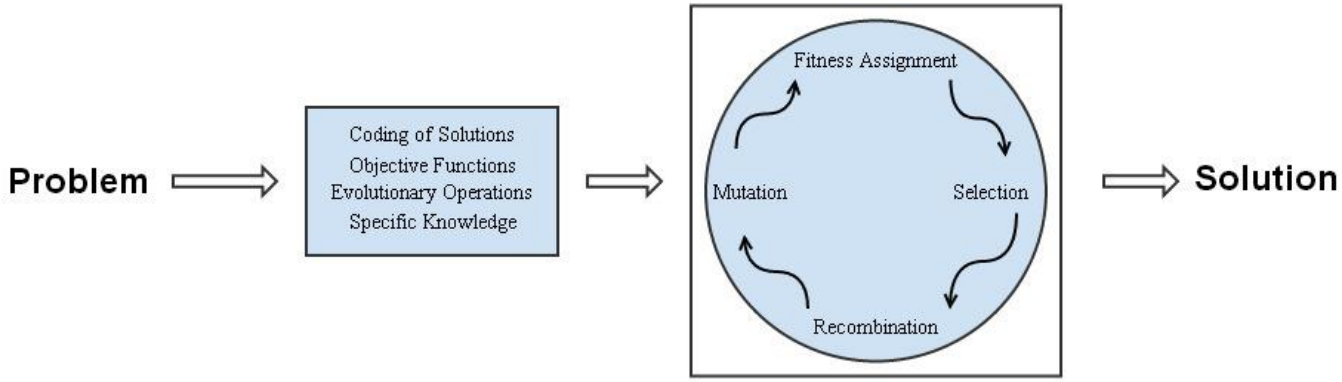


Figure 2.4 [39] Problem Solution Using Evolutionary Algorithms

2.3.1 Principles of EA

The principles of the EA are discussed as follows:

- *Initialize population*

  A population of number of individuals is randomly initialized at the beginning of the computation producing the first generation.

- *Evaluation*

  The objective function is then evaluated for the individuals in the initial generation. Fitness values of each solution candidates are computed using the objective function.

- *Selection*

  Selection refers to choosing the individuals for reproduction and number of offsprings each selected individual produces. Parents are selected according to their fitness by means of one of the following algorithms [39]:

  - roulette-wheel selection
  - local selection
  - stochastic universal sampling
  - tournament selection
  - truncation selection

- *Reproduction*

  Reproduction produces new individuals by combining the information contained in the parents. Depending on the kind of representation of the individuals the following algorithms can be applied [39]:

  - Discrete reproduction
  - Real valued reproduction
    - intermediate reproduction
    - line reproduction
    - extended line reproduction
  - Binary valued reproduction
    - single-point / double-point /multi-point crossover
    - uniform crossover
    - Cyclic crossover
    - PMX Crossover

- *Reinsertion*

  After the offsprings are generated, they must be inserted into the population. This is especially important, if number of offsprings produced is less as compared to the size of the original population or when not all generated offspring are to be used at each generation or if more offspring are generated than needed. A reinsertion scheme determines which individuals should be inserted into the new population and which individuals of the population will be replaced by offspring.

EAs could be used for multiobjective optimization because they are able to capture multiple optimal solutions in a single simulation run and may exploit similarities of solutions by recombination [9].

2.3.2 Characteristics of EA

The various characteristics of EAs that distinguish them from other heuristic methods are as follows [39]:

- Evolutionary algorithms carry the population search of points in parallel instead of a single point.
- Evolutionary algorithms consider the objective function and corresponding fitness levels for the directions of search. They do not consider any kind of derivative information or other auxiliary knowledge unlike other heuristic methods.
- Evolutionary algorithms use probabilistic transition rules for improving the solution space instead of deterministic ones.
- Evolutionary algorithms can provide a number of possible solutions to a given problem leaving the final choice to the user.

2.3.3 Related Study

The work presented in [9] gave a short overview of the structure and basic algorithms of evolutionary approaches. Various issues in context with evolutionary approaches and principles of EA were discussed using various NP complete problems. The authors in this work compared and improved existing evolutionary approaches to multiobjective optimization.

The authors in the work [19] discussed the probabilistic optimization algorithms based on natural evolution and compared evolution strategies, evolutionary programming and genetic algorithms with respect to certain characteristic components of EAs. Also the similarities and differences of the algorithms were elaborated in the work.

The work in [20] focused on the description and detailed study of Genetic-Evolutionary Algorithms (GEA). The authors addressed various aspects on the use of GEA for real world problems and discussed the need for using GEA. The authors also suggested many of the good economic reasons beyond the design of competent genetic algorithms that would favor GEAs than other methods.

The authors in the paper [21] offered an introduction to evolutionary programming and indicated relationship to other methods of evolutionary computation, especially, genetic algorithms and evolution strategies. The authors also mentioned the optimization performance of the techniques and extensions to include mechanisms of self-adaption.

The research article [22] surveyed the history as well as the current state of the rapidly growing field of evolutionary computation. The authors described the purpose, the general structure, and the

working principles of different approaches, including genetic algorithms (GA), evolution strategies (ES) and evolutionary programming (EP) by analyzing and comparing their most important characteristics (i.e. representations, variation operators, reproduction and selection mechanism). The authors also gave a brief overview on the diversity of application domains.

2.4 Genetic Algorithms

2.4.1 Introduction

Genetic algorithm (GA) is a famous stochastic optimization algorithm founded by Holland [29]. It simulates the evolutionary process of a population of individuals over time. The main characteristics of GA is the provision of a population, selection based on survival of the fittest, reproduction using crossover operator and mutation by incorporating random incremental changes. Genetic algorithms use a similar concept of reproduction and survival of the fittest to solve various optimization problems using various genetic operators. GAs work on a population of individuals called chromosomes that represent a possible solution to a given problem. The size of a population determines the amount of information stored by the GA. A fitness function is used to evaluate each individual's fitness value that measures the goodness of a solution to the problem. In each iteration, the algorithm breeds a population of individuals by applying genetic operators such as selection, crossover, inversion, and mutation. A new generation then evolves from the existing population hopefully with better solutions. Due to the stochastic selection process, good schemata are more likely to be inherited by the individuals of new generations. The fitness of the entire population is therefore improved over a number of generations [2]. The selection operator chooses members from the current generation for reproduction. The crossover operator on the other hand combines portions from the two parents to create two new offsprings. The mutation operator makes an incremental change to an individual to form a slightly new chromosome. Finally, the replacement operator decides which offsprings are going to replace which members of the current generation to create a new generation of individuals.

2.4.2 Types of Genetic Algorithm

Genetic Algorithms can be broadly classified into types based on the replacement strategy used:

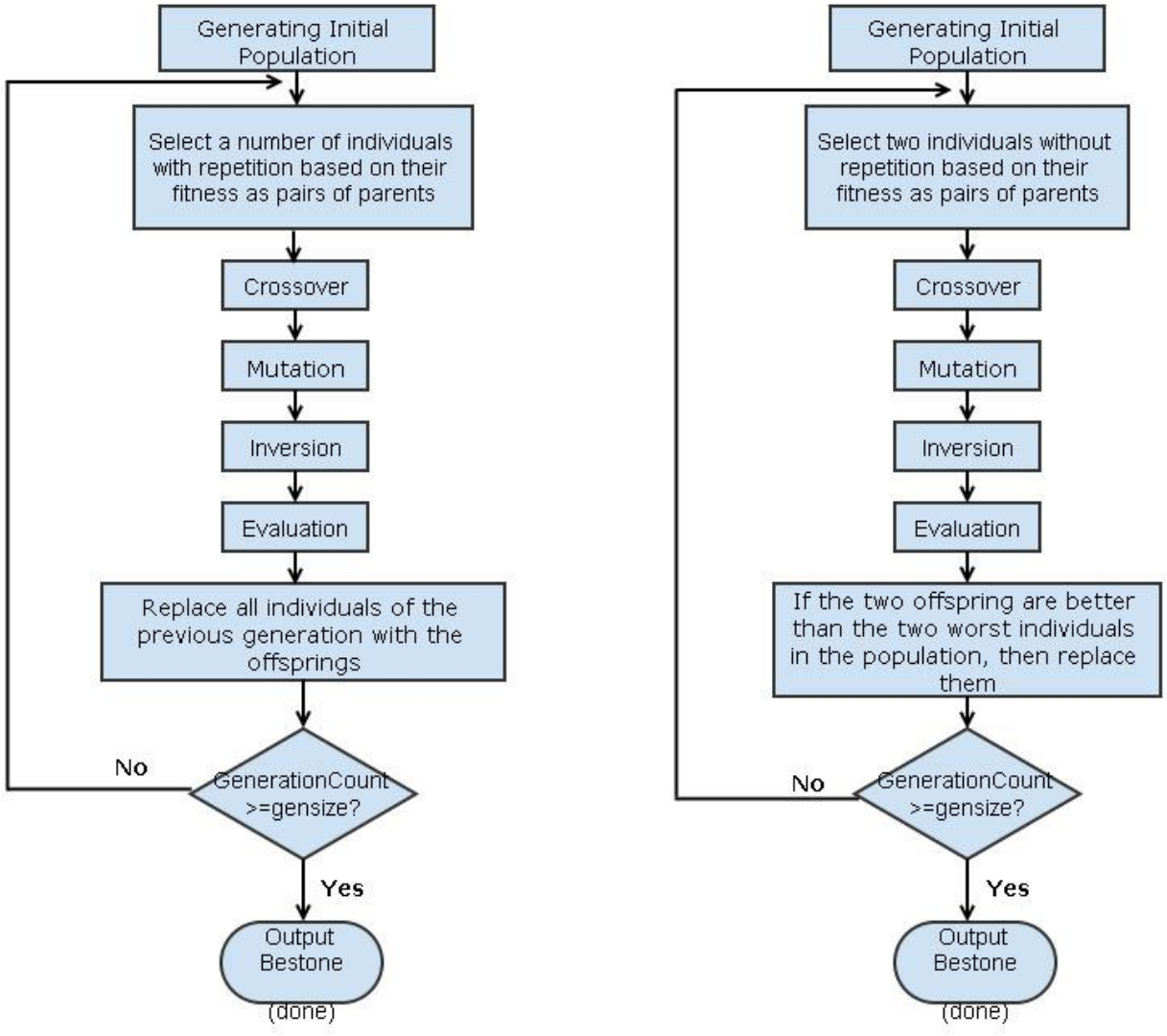
- *Generational GA*

  Generational GAs are generally used, where the new generation replaces the old one in each iteration [27].

- *Steady State GA*

  In a steady-state GA, every single iteration involves the application of one crossover or mutation operator and only one or two new individuals are added to the population by generally replacing the worst individuals [27]. The steady-state GA has a higher growth pressure on the promising individuals than that of a generational GA but it has a disadvantage as it is prone to stagnation [23].

Figure 2.5 illustrates two different types of Genetic Algorithms: (a) a generational GA, and (b) a steady-state GA where the main point of difference lies in the replacement strategy.



a) Generational GA **b) Steady State GA**

Figure 2.5 [23] Types of GA

2.4.3 Principles of GA

The basic principles of a GA are discussed as follows:

2.4.3.1 Encoding of a Chromosome

The chromosome should in some way contain information about solution which it represents [13].

- **Permutation Encoding**

  Permutation encoding is generally used in ordering problems namely, travelling salesman problem or task ordering problem. In *permutation encoding***,** every chromosome solution is encoded as a string of numbers such that numbers are represented in a sequence.

Chromosome A	9 7 5 2 6 4 3 1 8
Chromosome B	4 1 9 7 2 3 5 8 6

- **Binary Encoding**

  In *binary encoding*, every chromosome is represented in the form of strings of *bits*, *0* or *1*.

Chromosome A	010100101101101011100001
Chromosome B	100110100000110001011101

  Binary encoding gives many possible chromosomes even with a small number of alleles.

2.4.3.2 Selection

Selection determines the individuals that are chosen for reproduction and number of offsprings produced by each selected individual. The most commonly used method of selection is *Rank Selection.* In this method, first ranking of the population is done and then every chromosome receives fitness from this ranking. The worst chromosomes will be assigned fitness 1, second worst 2 etc. and the best chromosomes will be assigned fitness N (i.e. number of chromosomes in population) [31].

2.4.3.3 Basic Operators of GA

The basic genetic operators of GA are discussed as follows:

- **Crossover**

  Crossover combines genes from parent chromosomes in order to create a new offspring. The simplest way to perform a crossover is to choose randomly some crossover point; copy everything before this point from first parent and then copy everything after this crossover point from the second parent.

  "|" represents the crossover point in the following example.

Chromosome 1 *11001 | 10100110110*

Chromosome 2 *01011 | 11000011110*

Offspring 1 *11001 | 11000011110*

Offspring 2 *01011 | 10100110110*

There are other ways how to perform crossover which are far more complicated and efficient than discussed above. The type of crossover generally depends on encoding of the chromosome. Specific crossover made for a specific problem can improve performance of the genetic algorithm. There are many types of crossover which are discussed as follows:

- ***Cycle* Crossover**

  The Cycle Crossover operator identifies a number of cycles between two parent chromosomes. In order to form Child 1, cycle one is copied from parent 1, cycle 2 is copied from parent 2, cycle 3 copied from parent 1 and so on [42]. Here's an example:

  Parent 1: 5 4 7 3 6 2 8 1 9 0

  Parent 2: 0 1 2 3 4 8 6 7 5 9

  Cycle 1 Values: 5 9 0 will be marked.

  Cycle 1: Starting with the first value in Parent 1 and dropping down to the same position in Parent 2. 5 Goes to 0. Then looking for 0 in Parent 1 and it is found at the 10th position where it is dropped down to 9. Looking for this value in Parent 1 and it is found in the 9th position which further drops down to 5. Since it was started with 5, therefore, cycle is completed.

  Parent 1: ~~5~~ 4 7 3 6 2 8 1 ~~9 0~~

  Parent 2: ~~0~~ 1 2 3 4 8 6 7 ~~5 9~~

  Cycle 2 Values: 4 1 7 2 8 6 will be marked.

  Cycle 2: Starting with value 4 and dropping down to 1 which is found at the 8th position in Parent 1 and then dropping down to 7. 7 Drops down to 2, 2 Drops down to 8, 8 drops down to 6 and 6 drops down to 4. Thus cycle 2 is completed.

  Parent 1: ~~5~~ ~~4 7~~ 3 ~~6 2 8 1~~ ~~9 0~~

  Parent 2: ~~0~~ ~~1 2~~ 3 ~~4 8 6 7~~ ~~5 9~~

  Cycle 3 Values: 3

  Cycle 3: The only possible cycle left is of length 1 and contains the value 3.

The offsprings are filled as:

Parent 1: 5 4 7 3 6 2 8 1 9 0

Parent 2: 0 1 2 3 4 8 6 7 5 9

Copy Cycle 1: Cycle 1 values from Parent 1 are copied to Child 1 and values from Parent 2 will be copied to Child 2. Cycle 2 will be different.

Copy Cycle 2: Cycle 2 values from Parent 1 are copied to ***Child 2*** and values from Parent 1 will be copied to ***Child 1***.

Copy Cycle 3: Cycle 3 is like Cycle 1, Parent 1 goes to Child 1 and finally Parent 2 goes to Child 2.

Child 1: 5 1 2 3 4 8 6 7 9 0

Child 2: 0 4 7 3 6 2 8 1 5 9

- ***Order Point Crossover***

Order point crossover is a simple permutation crossover in which a swath of consecutive alleles from parent 1 drops down and remaining values are placed in the child in the order which they appear in parent 2 [40].

Parent 1: 5 4 7 3 6 2 8 1 9 0

Parent 2: 0 ~~1~~ ~~2~~ ~~3~~ 4 ~~8~~ ~~6~~ 7 5 9

Child 1: 0 4 7 3 6 2 8 1 5 9

Step 1: Selecting a random swath of consecutive alleles from parent 1 represented by an underlined.

Step 2: Dropping the swath down to ***Child 1*** and marking out these alleles in Parent 2.

Step 3: Starting on the right side of the swath, alleles from parent 2 are selected and inserted in ***Child 1*** at the right edge of the swath. Since 5 is in that position in Parent 2, it is inserted into ***Child 1*** first at the right edge of the swath. The alleles 1, 2 and 3 are skipped because they are marked out and 4 is inserted into the 2nd spot in ***Child 1.***

Step 4: For the second child from the two parents, Parent 1 and Parent 2 are flipped back and step 1 is repeated.

- ***Partially Mapped Crossover (PMX)***

In partially mapped crossover operator two crossover points are selected randomly from the parent chromosomes to generate the offspring. The two crossover points give a mapping

selection which is used to affect a cross through position by position exchange operations [43].

Example for PMX is as:

Parent 1: 8 4 7 3 6 2 5 1 9 0

Parent 2: 0 1 2 3 4 5 6 7 8 9

Step 1: A random swath of consecutive alleles from Parent 1 to the ***Child 1*** are copied.

Parent 1: 8 4 7 3 6 2 5 1 9 0

Parent 2: 0 1 2 3 4 5 6 7 8 9

Child 1: _ _ _ 3 6 2 5 1 _ _

Step 2: Analyzing the values, it is found that '4' is the first value in the swath of Parent 2 that isn't in the child. '6' is identified as the value in the same position in Parent 1. The value 6 in Parent 2 is located and found that it is still in the swath. This step is repeated using 6 as the value.

Parent 1: 8 4 7 3 6 2 5 1 9 0

Parent 2: 0 1 2 3 4 5 6 7 8 9

Child 1: _ _ _ 3 6 2 5 1 _ _

Repeating Step 1: '5' is found at the same position in Parent 1 and locating 5 in Parent 2. It also is in the swath so step 1 is repeated once more with value '5'.

Parent 1: 8 4 7 3 6 2 5 1 9 0

Parent 2: 0 1 2 3 4 5 6 7 8 9

Child 1: _ _ 4 3 6 2 5 1 _ _

Repeating Step 1: '2' is in the same position in Parent 1 and 2 is located in Parent 2 at the 3rd position. Finally, a position in the Child for the value 4 from Step 2.

Parent 1: 8 4 7 3 6 2 5 1 9 0

Parent 2: 0 1 2 3 4 5 6 7 8 9

Child 1: _ 7 4 3 6 2 5 1 _ _

Step 3: '7' is the next value in the swath in Parent 2 that isn't already included in the Child. So, checking the same index in Parent 1 and '1' is found in that position. Next, checking for '1' in Parent 2 and it is found in the 2nd position. Since the 2nd position is not part of the swath, location for the value '7' is found.

Parent 1: 8 4 7 3 6 2 5 1 9 0

Parent 2: 0 1 2 3 4 5 6 7 8 9

Child 1: 0 7 4 3 6 2 5 1 8 9

Everything else from Parent 2 drops down to the child.

- **Mutation**

  After a crossover is performed, mutation is performed on solutions to prevent all solutions in population getting trapped in local convergence of solved problem. Mutation changes the new offspring randomly. For binary encoding, a few randomly chosen bits from 1 to 0 or from 0 to 1 could be swapped for generating a mutated offspring. Mutation can then be following:

Original offspring 1	0101111000011110
Original offspring 2	1101100100110111
Mutated offspring 1	0100111000011110
Mutated offspring 2	1101101100110111

2.4.4 Basic Parameters of GA

There are two basic parameters of GA namely, crossover probability and mutation probability.

- **Crossover Probability:** This parameter defines *'How often the crossover will be performed?'* Crossover is done with a goal that new chromosomes will have good parts of old chromosomes and hopefully the new chromosomes will be better.
- **Mutation Probability:** This parameter defines *'How often the parts of chromosome will be mutated?'* If there is no mutation, offspring is taken after crossover (or copy) without any change.

Other Parameter

Population Size: This parameter defines *'How many chromosomes are in population (in one generation)?'* If the population is very less, GA has a few possibilities to perform crossover and only a small part of search space is explored. On the other hand, if the population is too large, GA slows down.

2.4.5 Need for Hybrid Approaches with GA

GAs are domain-independent stochastic search techniques [26]. The power of GAs comes from the fact that the technique is robust and can deal successfully with a wide range of problem areas.

Genetic Algorithms are fast as compared to other techniques that make them most widely held for solving standard cell placement problem.

However, one disadvantage of GAs is that even though the solution space is explored properly, the solution may get stuck in local optimum. The solution space is not explored fully and the algorithms may give a solution that is not fairly good. However, for such problems, the genetic algorithms are combined with other heuristic techniques to form hybrid approaches that overcome the limitations of exploration capabilities of GA and thus form more efficient approaches giving better results. Besides, hybrid approaches, memetic algorithms have been developed that have improved the results. The hybrid approaches are generally developed with the algorithms that would improve the local search. The algorithms that are generally combined with GA to form hybrid algorithms include Simulated Annealing and Tabu Search.

In order to speed up the search procedures for SCP, various parallel and island based genetic algorithms have been developed that have reduced the time required to produce efficient results than basic GAs.

2.4.6 Related Study

The work on standard cell placement using genetic algorithms started in 1990. Shahookar et al in [33] proposed a genetic algorithm to solve the standard cell placement problem. The proposed algorithm worked on a set of solutions constituting a constant size population. The authors implemented the algorithm by applying various crossover operators namely PMX crossover, cyclic crossover and order crossover. The experimental results showed that cyclic crossover outer performed in comparison to PMX and order crossover methods. The authors also incorporated mutation and inversion operator in the proposed algorithm.

In the paper [34], the proposed algorithm applied transformations on the chromosomal representation of the physical layout instead of directly applying the transformations on physical layout. The work presented in this paper incorporated a Selection method that would select the cell having maximum area first and so on, and the crossover operator applied on every cell with cells having next higher, next lower and equal area. The algorithm used a Random Point crossover.

The authors in [35] further extended their previous approach by incorporating meta-genetic algorithm. The meta-genetic algorithm is itself a genetic optimization process that executes the genetic algorithm to solve a placement problem and manipulates its parameters to optimize its fitness. The proposed work used the genetic operators with different rates and probabilities which led to the execution of GA with different parameters giving more efficient results.

The authors in the [36] proposed an optimization of hybrid and local search algorithms for standard cell placement problem. The work focused on investigating in detail the hybrid systems based on heuristic techniques including hybrids of HNN (Hopfield Neural Network) and GA (Genetic Algorithm) with SA (Simulated Annealing) & GA based PRSA algorithms. The authors first suggested HNN & GA hybrid system and then extended to SA & GA hybrid systems that used either a coupling mechanism or integrating the methods with their key features to develop a new method, Parallel Recombinative SA (PRSA) and proposed a solution representation and development of cross-over and neighborhood operators.

The authors in the work [37] addressed the optimization of cell placement phase in VLSI design process and proposed a novel hybrid algorithm for performance and low power driven VLSI standard cell placement. The authors in the proposed work suggested the incorporation of fuzzy logic in the design of aggregating function rather than a single objective. Further the authors combined the better searching features of Tabu Search and parallel exploration capabilities of Genetic Algorithm to give an efficient hybrid algorithm.

The authors in [38] proposed an approach that dealt with different constraints and objectives in one optimization step. The proposed algorithm used genetic algorithms with tree-structured genotype representation. The authors suggested hybrid algorithms for two constrained placement problems namely Facility layout Generation and VLSI Macro Cell Layout Generation. The suggested work used genetic algorithm with non-standard genotype representation for bottom up construction slicing tree for individual that considered all constraints.

The author in paper [24] proposed a hybrid cell placement approach for low power VLSI standard cell placement based on two evolutionary approaches namely Tabu Search and Genetic Algorithm. The proposed approach and GA was compared and the author concluded that the proposed approach outperformed genetic algorithm in terms of quality of final solution and CPU time. The proposed algorithm used the solution encoding in the form of 2-D grid.

In the work [12], the authors proposed a memetic algorithm for standard cell placement. The suggested algorithm is a pure GA combined with tile-based local search in three different ways i.e. before crossover, after crossover and before and after crossover. As stated experimentally, the authors concluded that the amount of improvement of the quality of solution and decreased CPU time is enhanced by integrating GA with local search method.

The authors in the work [32] presented the partial mapped crossover and cells exchange mutation operators in the genetic algorithm for standard cell placement problem. In the proposed algorithm, a heuristic initial placement approach along with methods of timely updating the coordinates of cells

were used in order to eliminate overlaps between the cells. The experimental results revealed improvement in placement results.

The summary of all the papers related to genetic approaches applied on SCP is shown in table 2.1.

2.5 Research Gaps

The major drawback with work presented in [35] is the runtime of the algorithm. The time required for the evaluation of new configurations is very large that leads to less efficiency of the algorithm when the circuit size is very large. The hybrid approaches discussed [36], [37] , [38] include complex crossover operators and complex encoding schemes but these approaches still face the same problem as the basic GA; the population's diversity drops strongly and the GA gets stuck by creating almost only duplicates of a small set of leading candidate solutions called super-individuals. The proposed algorithm in [34] incorporated the crossover operator that does not introduce much innovation thus leading to the formation of same individuals in every generation leading to premature convergence. Thus the population diversity is lowered and much precious CPU time is wasted.

Besides the low efficiency of searching mechanism in genetic approaches, one common limitation that has been observed in the previous research work is that the algorithms employed do not keep track of the history of best solutions encountered so far in the search space. The solution in the search space is not exploited efficiently thus sometimes leading to the elimination of the best solution during genetic operations.

2.6 Objectives of the Study

In a situation of premature convergence and the increased runtime, it becomes very obvious that the heuristic search is not performing well, and something must be changed in the GA's setup. Keeping these points in mind, the following three objectives of the research work are set:

- Detailed study of various genetic approaches that have been applied on standard cell placement problem for VLSI design.
- Development of an algorithm using evolutionary approach for standard cell placement problem.
- Comparison of the proposed algorithm with simple genetic algorithm approach in terms of wirelength and running time of algorithms.

Table 2.1 Summary of Genetic Approaches For SCP

Author Name	Year	Proposed Technique	Key Features
K. Shahookar, P. Mazumder	1990	Genetic Algorithm	- Comparison of three crossover operators i.e. Order crossover, PMX and Cyclic - Cyclic crossover operator used. - Inversion operator used
K. Shahookar, P. Mazumder	1990	Genetic Algorithm using Meta-Genetic Parameter Optimization	- Parameters of GA optimized - Genetic operators used at different rates in each execution of GA
Volker Schnecke, Oliver Vornberger	1997	Hybrid Genetic Algorithm	- GA with tree-structured representation used - Bottom up construction of slicing tree for an individual used - Crossover operator is Gene Pool Recombination - Three mutation operators used
Sadiq M. Sait, Mahmood R. Minhas	2002	Novel Hybrid Algorithm	- Fuzzy logic incorporated in aggregating function - TS + GA
Mahmood R. Minhas	2003	Evolutionary Cell Placement Technique	- Tabu Search + Genetic Algorithm - Solution encoding in form of 2D grid
Shawki Areibi, Zhen Yang	2004	Memetic Algorithm	- Pure GA + Tile Based Local Search - Local search called after crossover, before crossover and both
Guofang Nan, Minqiang Li, Wenlan Shi, Jisong Kou	2006	Improved Genetic Algorithm for Cell Placement	- PMX crossover and cells exchange mutation operators - Elimination of overlaps between cells using initial heuristic placement approach - Objective Function simplified
Aaquil Bunglowala, Dr. B. M. Singhi, Dr. Ajay Verma	2009	Hybrid and Local Search Algorithms	- Hopfield Neural Network + GA - SA + GA - PRSA included crossover and neighborhood operators
Rini Mahajan, Amit Saxena, Baljit Singh Khera	2010	Genetic Algorithm using various genetic operators	- Selection Method: cell having maximum area selected first - Random point Crossover

2.7 Summary

This chapter focusses on the standard cell placement problem and the various approaches developed by the researchers to solve this problem. This chapter serves the purpose of motivation of the research work. The detailed explanation of the background of VLSI cell placement techniques, evolutionary algorithms and genetic algorithms was discussed in this chapter along with a brief survey on the evolutionary approaches applied on SCP. The work discussed in the survey focused on various aspects of SCP discussing different genetic operator applied on different approaches; various objective functions of SCP were described not taking into account the orientation of cells, time delay and performance of circuits etc. The research gaps were highlighted and the objectives of the study were made clear in the chapter. The paper published for this chapter was:

> Jobanpreet Kaur and Maninder Kaur, "A Survey On Various Genetic Approaches for Standard Cell Placement," *International Journal of Computer Technology and Applications*, Vol. 4, No. 3, pp. 533-536, June, 2013.

CHAPTER 3

PROBLEM STATEMENT

The cell placement problem in VLSI design is a crucial stage in the physical design process. For a given circuit consisting of a set of cell modules and a net list giving the interconnections between these modules, the standard cell placement problem is to arrange a layout that would indicate the positions of the modules in parallel rows such that all the nets are interconnected using wires and the total layout area is minimized. The total chip area includes two parts: the area required for the cell rows and the area required for routing of wires. Although there are a number of objectives in standard cell placement problem such as routability, low power, time delay, performance, the proposed research work is focused on the main objective i.e. minimizing wire length.

3.1 Problem Description

Given an electrical circuit consisting of cells, with predefined input and output terminals, interconnected in a predefined way, the Standard Cell Placement problem is constructing a layout indicating the positions of the cells, so that the estimated wire length and the layout area are minimized and other given constraints are satisfied. The inputs to the problem are cell description with sizes and terminal locations and the netlist defines the interconnections between the cells. The output list contains a list of x- and y- coordinates of the cells [23].

Graph theory is used to model VLSI physical design problems including Standard Cell Placement [6]. A circuit can be represented by a hypergraph G (V,E) where the vertex set V = $\{v_1,v_2,\ldots,v_n\}$ represent the set of cells to be placed and the edge set E = $\{e_1,e_2,\ldots,e_n\}$ represent the set of nets connecting the cells. Each edge e_j is an ordered pair of vertices having a non-negative weight w_j assigned to it. The placement problem is to assign all cells of the circuit to the predetermined positions on the chip such that cells do not overlap. Each cell i is assigned to a location (x_i, y_i) on XY plane. Minimizing the wire-length is approximately equivalent to minimizing the total chip area for the standard cell layout [33],therefore, the total cost of a placement layout, denoted f (x, y), can be estimated by the sum of wire length over all nets [11].

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

where (x_i , y_i) denotes the location of cell *i*; w_{ij} is a non-negative weight of the edge connecting cell *i* and cell *j*. The above formulation can be rewritten in the form of matrix as:

$$\emptyset(x,y) = \frac{1}{2}\, x^T C x + d_x^T x + \frac{1}{2}\, y^T C y + d_y^T y + t$$

Vectors *x* and *y* represent the coordinates of the N cells; matrix *C* is the Hessian matrix; vectors d_x^T and d_y^T and the constant term *t* result from the contributions of the fixed cells. The solutions are evaluated based on the fitness value using a fitness function F:

$$F = \frac{1}{\sum_{i=1}^{n} HPWLi}$$

where $HPWLi$ is the estimated wire-length of the net *i* and *n* is the number of nets. HPWL is Half Perimeter Wire Length method used to estimate the total wire length.

3.2 Wirelength Estimation Methods

The wirelength can be estimated using any of the following methods: Half perimeter Wirelength Method (Semi Perimeter Method), Complete Graph, Minimum Chain and Steiner Tree Approximation, Source to Sink Connection method and Minimum Spanning Tree method [1]. Since at the placement stage, detailed routing is not available and the exact total wire length of the layout cannot be obtained, therefore, the Half-perimeter method is commonly used to estimate the total wire-length. HPWL is an efficient and widely used approximation to estimate the wirelength of a net. The method consists of finding the smallest bounding rectangle that encloses all the pins of the net to be connected. The estimated wirelength of the interconnections is half the perimeter of this bounding rectangle. That is,

$$\sum_{i \in Nets} HPWL_i = \sum_{i \in Nets} \frac{p_i}{2} = \sum_{i \in Nets} [span_{xi} + span_{yi}]$$

where p_i denotes the perimeter of net *i*; $span_{xi}$ and $span_{yi}$ denotes the vertical and horizontal spans of net *i*'s bounding box respectively.

3.3 Summary

This chapter describes the problem statement for standard cell placement in context with the graph theory. The fitness function for the problem was defined and the wirelength estimation method (Half Perimeter Wire Length) was discussed.

CHAPTER 4

WORK DONE

The standard cell placement problem has been widely studied by the researchers and various algorithms have been developed. One of the most efficient approaches for solving SCP is evolutionary approach that is a population-based type of optimization algorithm and mimics the behavior of natural biological evolution. The work proposes a variant of an evolutionary approach for solving standard cell placement (EASCP) that explores the solution space and memorizes the best solutions of every generation thus exploiting the best solutions. A detailed explanation of this approach is given below. The results obtained from experimental analysis are also presented in form of tables and graphs.

4.1 Methodology

The proposed work follows a genetic paradigm for solving standard cell placement (EASCP) problem where a trial value is associated with every solution in the population. The trial value of an individual is modified based on its capability to generate a better offspring thus giving a chance to the individual solutions to generate better offsprings.

The initial population is randomly generated and all the individuals are paired to form parents for crossover. After the application of order point crossover, if the offspring generated has worse fitness value than its parents, the trial value of the parent solution is incremented by 1 but for the vice-versa case, the trial value for the parents remain same and the trial value of newly generated offspring is set to 0. The best solution in every generation is memorized and stored in a repository thus keeping track of the best solution generated so far. The process is repeated for a set of generations and all the individuals whose trial value exceeded a certain limit are discarded and the population is flushed with new distinct solutions. The block diagram for the working of EASCP is shown in the figure 4.1. The proposed algorithm EASCP has a number of advantages over other existing techniques. The problem of premature convergence is dealt with by allowing the parent solutions to reproduce better solutions. If the parent solutions are not able to generate better solutions, they are discarded and new distinct solutions are allowed to enter the process. Thus the probability of GA getting trapped in local optima is reduced. Also the running time of algorithm is improved as same solutions are not evaluated again and again.

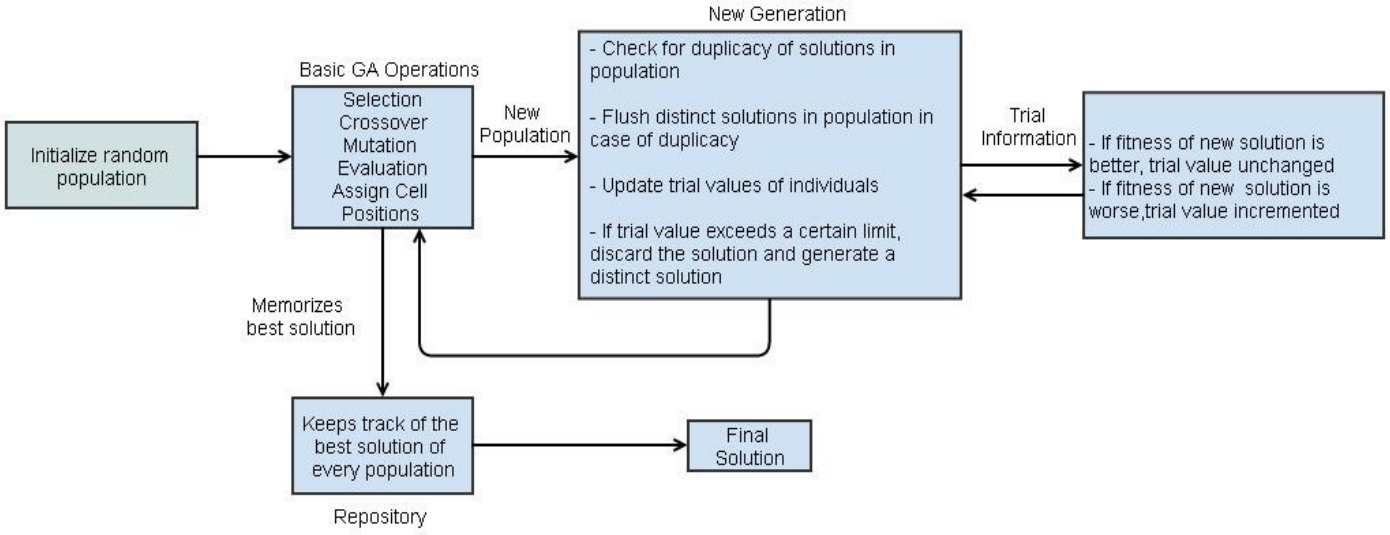


Figure 4.1 Block Diagram Showing Working of EASCP

4.1.1 Description of EASCP Algorithm:

The basic components of the algorithm are described in table 4.1.

Table 4.1 Basic Components of Algorithm

Symbol Representation	Meaning of Symbols
N	Total number of cell modules in the circuit
Nnets	Total number of nets in the circuit
Gen	Current generation
P	Total number of population
MAXGEN	Maximum number of generations

Pseudo Code for the algorithm is as follows:

Step 1. Generate an initial population of size P randomly with a set of feasible solutions.

Step 2. Read the input files (netlist files and the cell library files) and assign the parameter values to the variables. The cell positions for the initial population are assigned by considering the parameters of cell library files.

Step 3. Determine the fitness value of each solution in the population using the Half Perimeter Wirelength (HPWL) method since detailed routing is not available at the placement stage. The perimeter (p_i) of a net *i* is calculated using the sum of horizontal ($span_{xi}$) and vertical

spans ($span_{yi}$) of net *i*'s bounding box [11]. The fitness function is the inverse wirelength. More the fitness value less is the wirelength. The fitness value is maximized in the algorithm.

$$\sum_{i \in Nets} HPWL_i = \sum_{i \in Nets} \frac{p_i}{2} = \sum_{i \in Nets} [span_{xi} + span_{yi}]$$

$$F = \frac{1}{HPWL_i}$$

Step 4. Pair the chromosomes in the population with each other such that order crossover operation is applied to generate two offsprings.

Step 5. Perform mutation on the generated offsprings by inverting the cell positions at random points. The crossover and mutation operations are applied on the whole population at once to generate a new population from the existing population.

Step 6. Assign the fitness value and the cell positions for the generated offsprings using the parameters cell height and cell width of the cells. The two most fitted individuals among the parent and the child solutions form a part of the population whereas the rest two are discarded.

Step 7. At the end of every generation, the best solution in the generation is memorized and stored in the repository and before incrementing the generation, a check for duplicate individuals in the population is done. In case of duplicity, solutions are randomly generated iteratively until population is filled with distinct solutions.

Step 8. After a population is generated, the trial values will be changed for a set of solutions. If a parent solution produces an offspring with a worse fitness value, its trial value will be incremented by 1 and the parent solution will become the part of the next generation population. For the vice-versa case, the trial value will remain unchanged giving a chance to the child solution to be inserted in the next population.

Step 9. The trial value of the individuals is evaluated after every generation. If the trial value of a particular individual exceeds a certain limit, then that individual is removed from the population and is replaced by a randomly generated distinct solution in the population.

Step10. After a repeated number of generations, the best solution amongst the best solutions of every generation is chosen as the final solution from the repository.

4.2 Experimental Setup

This section illustrates the sample circuit used for simulating results of the algorithm, the basic configurations of the input file of the circuit and some general aspects of the setup.

4.2.1 Problem Instance

The experimental results presented in subsequent section were generated using a test circuit. This test circuit contains the configurations of 10 cell modules that need to be placed on a fixed size chip (dimensions provided) such that the wirelength is minimized. The input file of the test circuit is shown in figure 4.2.

```
100  10  10  25  10
N1  7
N2  5
N3  4
N4  3
N5  2
N6  8
N7  2
N8  5
N9  9
N10 7
C1  10  N1  N6  N9
C2  25  N3  N5  N7
C3  12  N3  N4
C4  5   N2  N8
C5  30  N5  N6  N10
C6  17  N1  N3  N4
C7  62  N4  N7
C8  36  N5  N8
C9  40  N1  N2  N6  N8  N10
C10 27  N7  N9
```

Figure 4.2 Input File for the Test Circuit

The input file of the test circuit is interpreted as follows. The first element in the file in file i.e. 100 represents maximum row length. The next elements in the first row correspond to number of cells, number of nets, cell height and channel height respectively. The data from 2nd row to 11th row assigns the weights to corresponding nets. The data from 12th row to 21st row provides the information regarding the interconnections between the cells and the cell widths of each cell (represented by a unique cell number).

4.2.2 Initialization of Parameters

The parameters namely Crossover Probability (Rc), Mutation Probability (Rm) and Population Size (P) need to be assigned fixed values as their values determine the efficient execution of the algorithm. The algorithm was implemented using crossover rate, Rc = 0.66, mutation rate, Rm = 0.01 and P = 10.

4.3 Simulation Results and Discussions

The work is carried out by writing code in Turbo C on Windows 7 platform and running on an Intel® Core™ 2Duo (2.10 GHz) machine with 2 GB memory. The EASCP algorithm was implemented in C language using test sample circuits. The simulation was carried on six test circuits with a maximum nodes of 15 and maximum nets 13. The number of generations for which the algorithm was executed was 20.

The results of the experiments are reported in Table 4.2. The optimum values of the fitness values (inverse of wirelength) in table 4.2 shows a variation between the two algorithms. The EASCP shows better results in terms of the quality of solutions. The CPU time for the two algorithms is comparable for circuits having larger nodes. The decrease in time is due to the reason that the individuals with limited trial values are discarded and are not explored repeatedly.

Table 4.2 Comparison of Basic genetic Algorithm and the Proposed Evolutionary Algorithm based on the minimum and average fitness and CPU time (in seconds)

Circuit	No. of Nodes	No. of Nets	Basic Genetic Algorithm			Proposed Evolutionary Algorithm (EASCP)		
			Maximum Fitness	Average Fitness	CPU	Maximum Fitness	Average Fitness	CPU
Sample1	10	10	0.000315	0.000275	0.22	0.000560	0.000370	0.28
Sample2	14	10	0.000564	0.000448	0.20	0.000782	0.000540	0.21
Sample3	12	7	0.001091	0.001000	0.17	0.001182	0.001046	0.19
Sample4	15	13	0.000724	0.000587	0.26	0.000965	0.000787	0.26
Sample5	10	6	0.000659	0.000597	0.25	0.000861	0.000688	0.27
Sample6	12	10	0.000491	0.000443	0.18	0.000516	0.000494	0.21

The wirelength can be calculated using the fitness value as wirelength is the inverse of fitness value. Thus for sample 1, the minimum wirelength for the basic GA and EASCP comes out to be 2857 and 1786 respectively.

The comparison of the performance of the two algorithms in terms of average fitness values and maximum fitness values is shown in the form of column charts in figure 4.3 and figure 4.4.

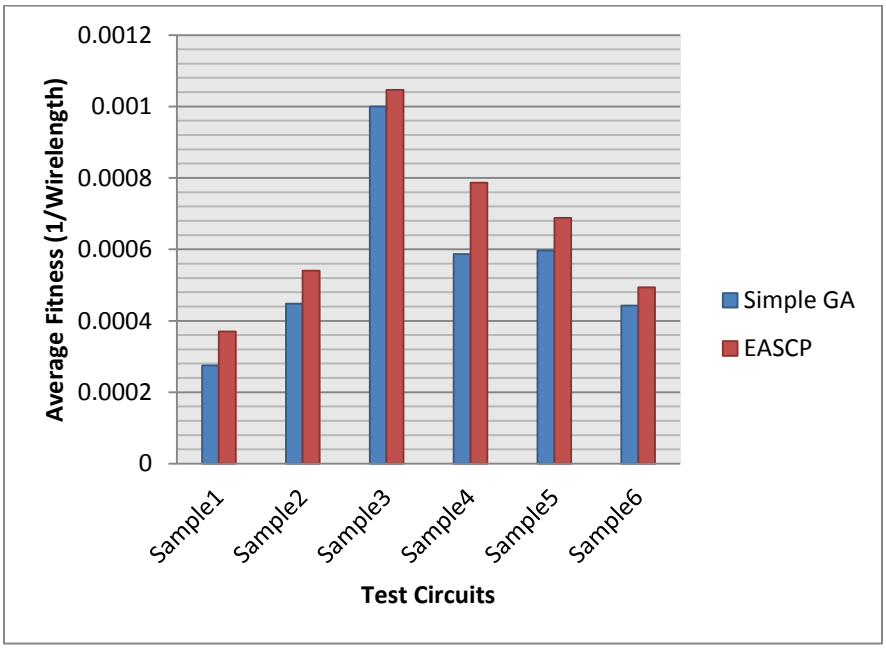


Figure 4.3 Comparative Results of Simple GA and EASCP Based on Average Fitness Value (1/Wirelength)

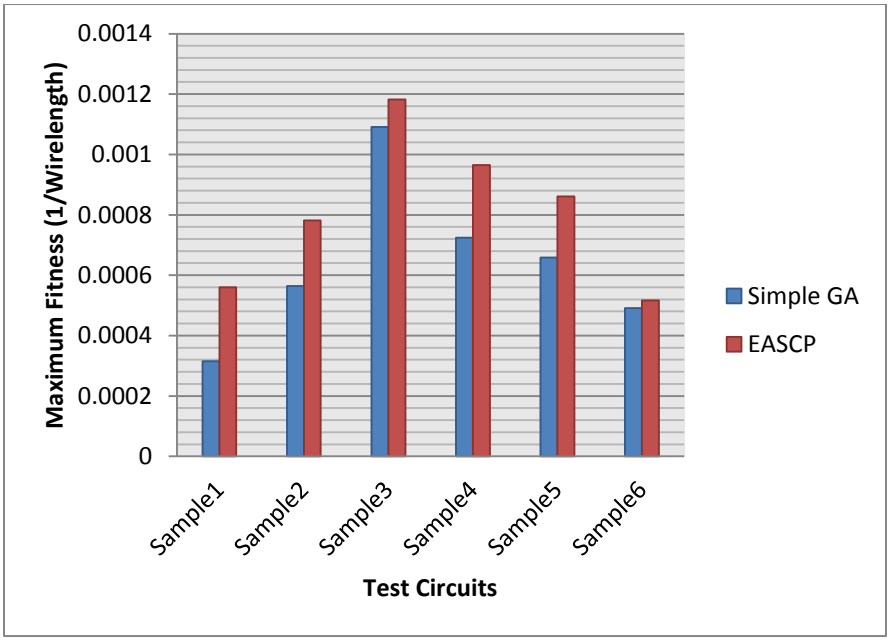


Figure 4.4 Comparative Results of Simple GA and EASCP Based on Maximum Fitness Value (1/Wirelength)

The graphs depict clearly that the EASCP gives better average wirelength in comparison with the basic GA producing better placement results and the figure 4.5 depicts that the CPU time for EASCP is comparable for circuits with large number of nodes.

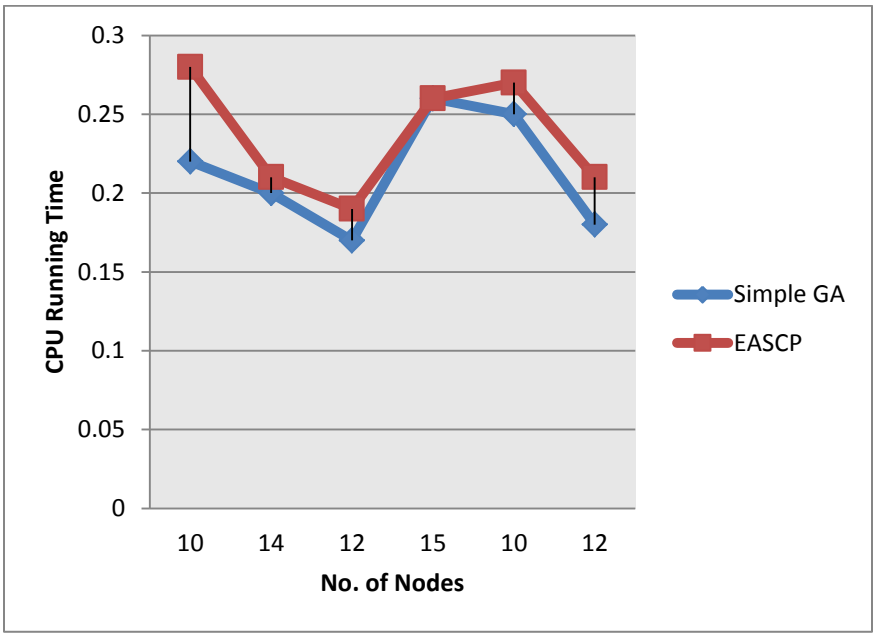
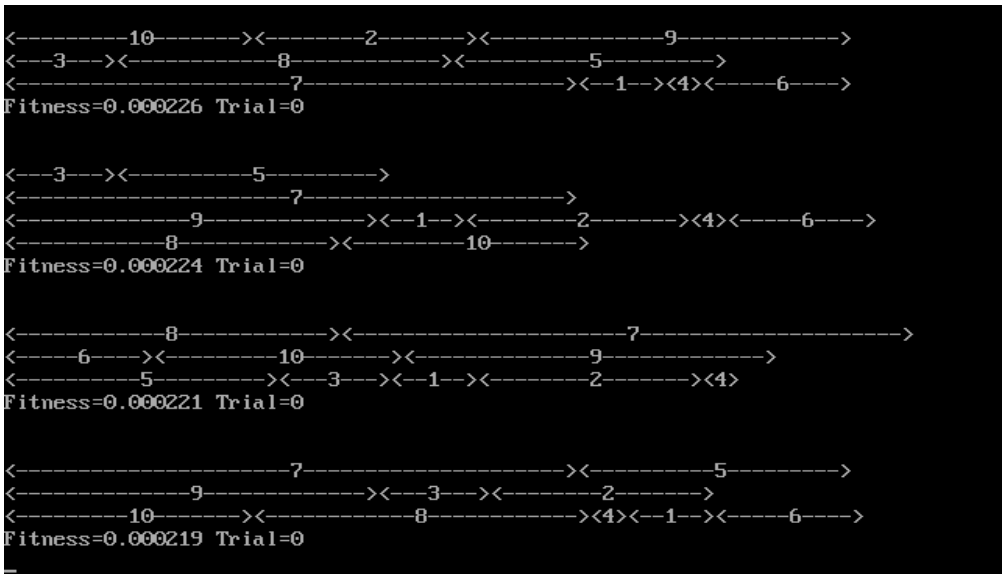


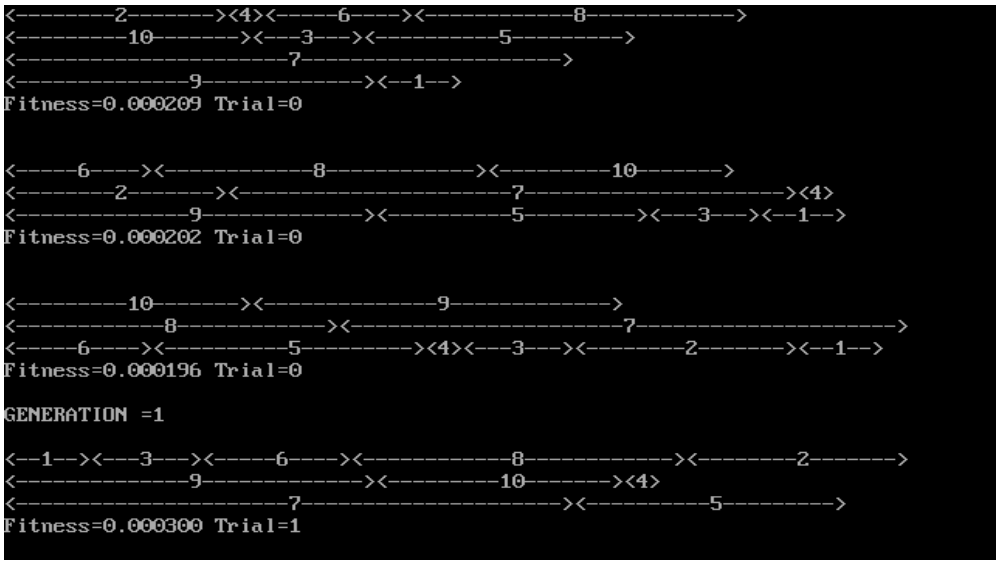
Figure 4.5 Comparative Results of Simple GA and EASCP Based on CPU Time and No. of Nodes

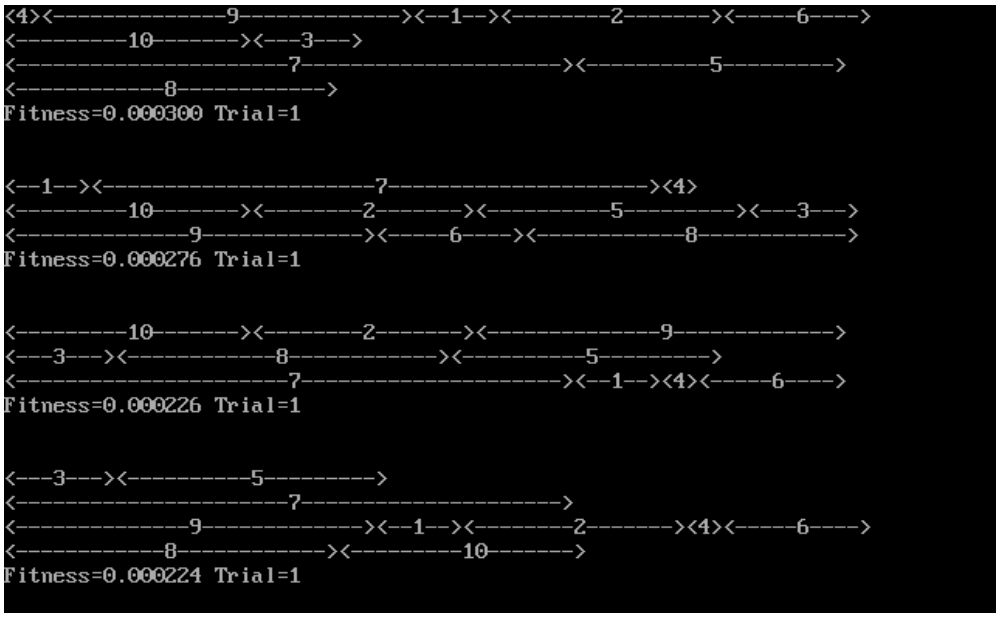
4.4 Snapshots

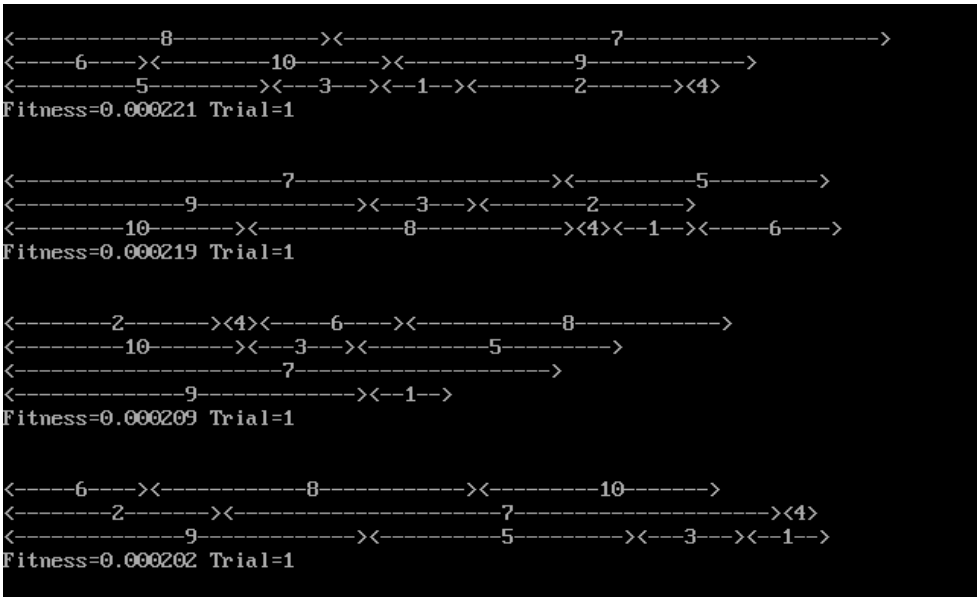
The EASCP algorithm is implemented in Turbo C, the snapshots of which are shown below:

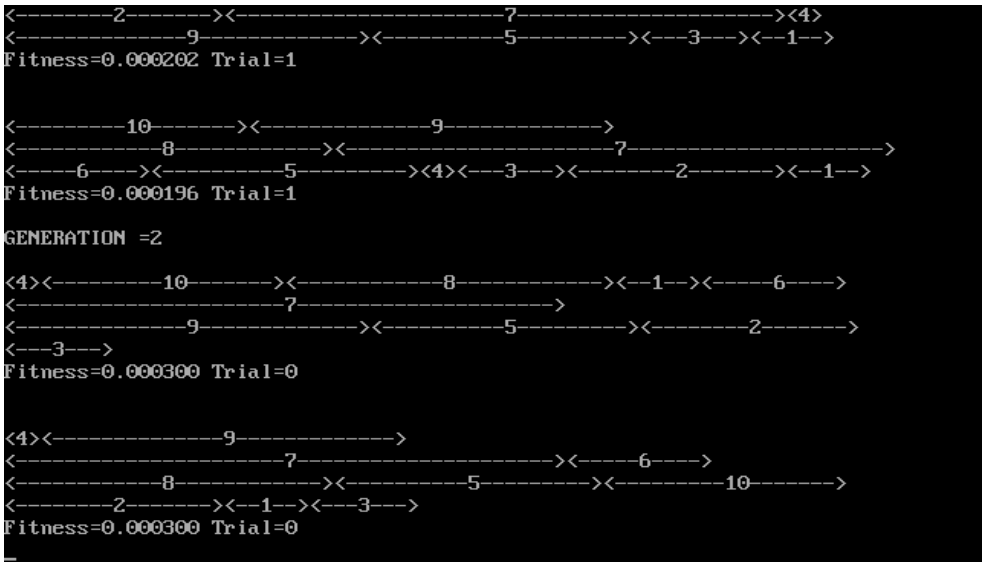
```
The LAYOUT and FITNESS VALUES of Generations is as:

GENERATION =0

<--1--><---3---><-----6----><------------8------------><--------2-------->
<--------------9-------------><----------10-------><4>
<----------------------7---------------------><----------5--------->
Fitness=0.000300 Trial=0


<4><--------------9-------------><--1--><--------2-------><-----6---->
<---------10-------><---3--->
<----------------------7---------------------><----------5--------->
<------------8------------>
Fitness=0.000300 Trial=0


<--1--><----------------------7---------------------><4>
<---------10-------><--------2-------><----------5---------><---3--->
<--------------9-------------><-----6----><------------8------------>
Fitness=0.000276 Trial=0
```


<---------10-------><--------2-------><--------------9-------------->
<---3---><------------8------------><----------5---------->
<----------------------7---------------------><--1--><4><-----6---->
Fitness=0.000226 Trial=0
<---3---><----------5---------->
<----------------------7---------------------->
<--------------9-------------><--1--><---------2-------><4><-----6---->
<------------8------------><---------10------->
Fitness=0.000224 Trial=0
<------------8------------><----------------------7---------------------->
<-----6----><----------10------><--------------9------------->
<----------5---------><---3---><--1--><--------2------->
Fitness=0.000221 Trial=0
<----------------------7---------------------><----------5---------->
<--------------9-------------><---3---><--------2------->
<---------10------><------------8------------><4><--1--><-----6---->
Fitness=0.000219 Trial=0


<--------2-------><4><-----6----><------------8------------>
<---------10-------><---3---><----------5---------->
<----------------------7---------------------->
<--------------9-------------><--1-->
Fitness=0.000209 Trial=0
<-----6----><------------8------------><---------10------->
<--------2-------><----------------------7----------------------><4>
<--------------9-------------><----------5---------><---3---><--1-->
Fitness=0.000202 Trial=0
<---------10-------><---------------9-------------->
<------------8------------><----------------------7---------------------->
<-----6----><----------5---------><4><---3---><--------2-------><--1-->
Fitness=0.000196 Trial=0
GENERATION =1
<--1--><---3---><-----6----><------------8------------><--------2------->
<--------------9-------------><---------10-------><4>
<----------------------7---------------------><----------5---------->
Fitness=0.000300 Trial=1

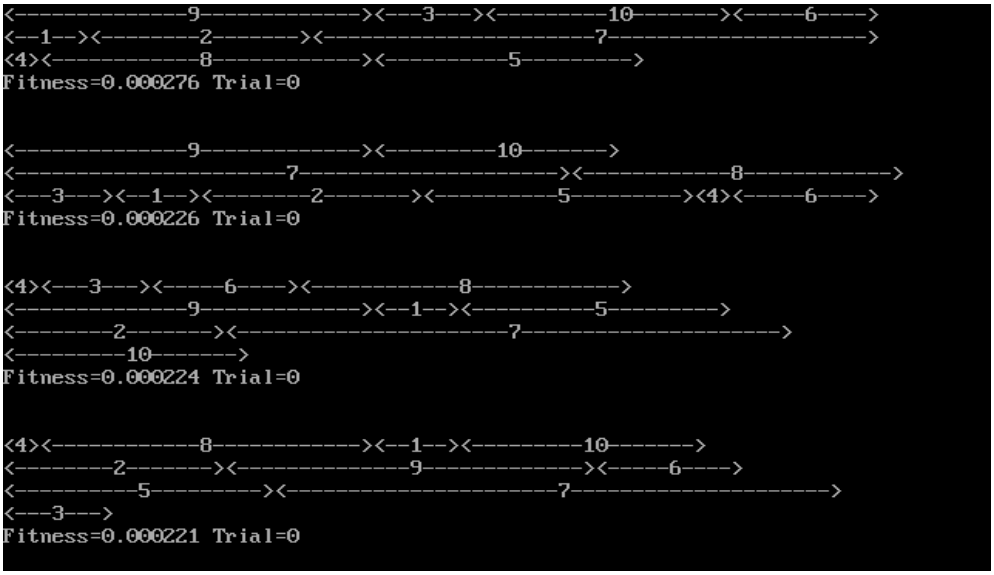

<4><--------------9-------------><--1--><--------2-------><-----6---->
<---------10-------><---3--->
<----------------------7---------------------><----------5---------->
<------------8------------>
Fitness=0.000300 Trial=1
<--1--><----------------------7----------------------><4>
<---------10-------><--------2-------><----------5---------><---3--->
<--------------9-------------><-----6----><------------8------------>
Fitness=0.000276 Trial=1
<---------10-------><--------2-------><--------------9-------------->
<---3---><------------8------------><----------5---------->
<----------------------7---------------------><--1--><4><-----6---->
Fitness=0.000226 Trial=1
<---3---><----------5---------->
<----------------------7---------------------->
<--------------9-------------><--1--><---------2-------><4><-----6---->
<------------8------------><---------10------->
Fitness=0.000224 Trial=1

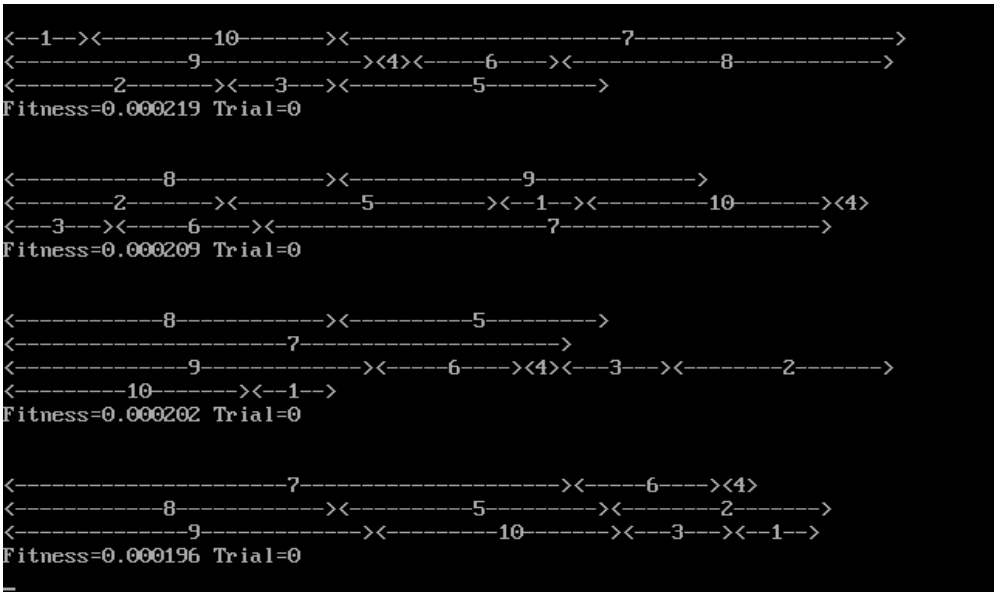


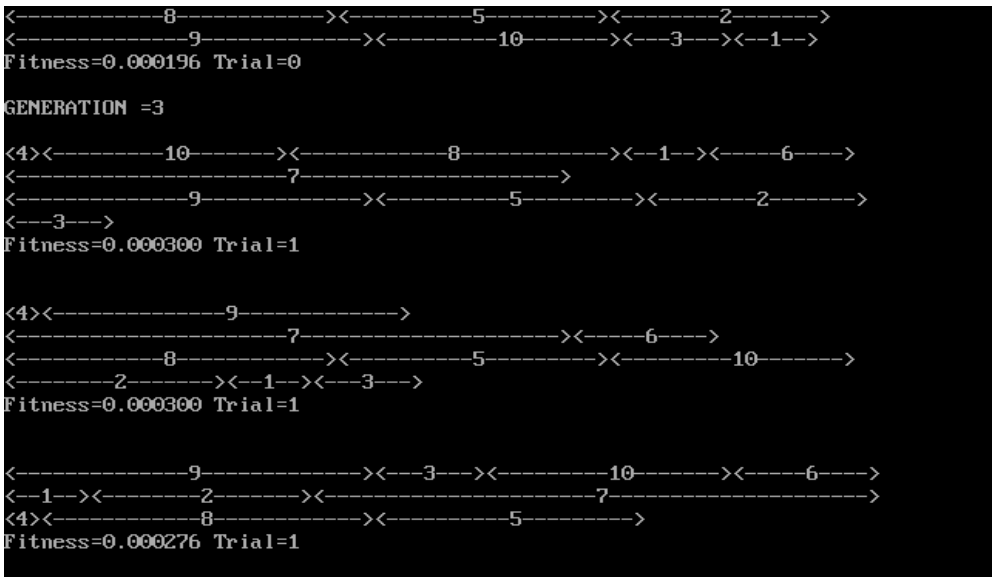


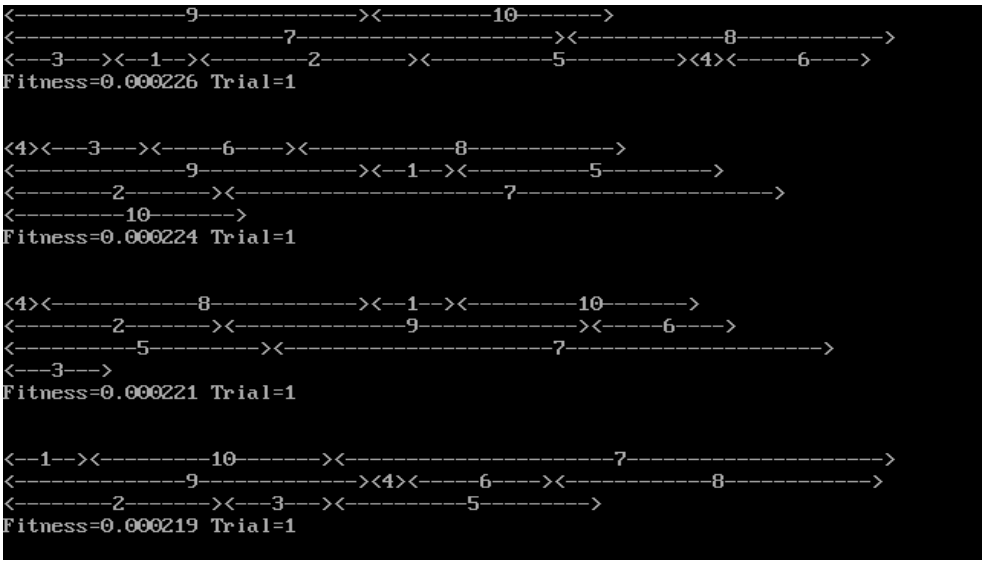
GENERATION =2

GENERATION =2

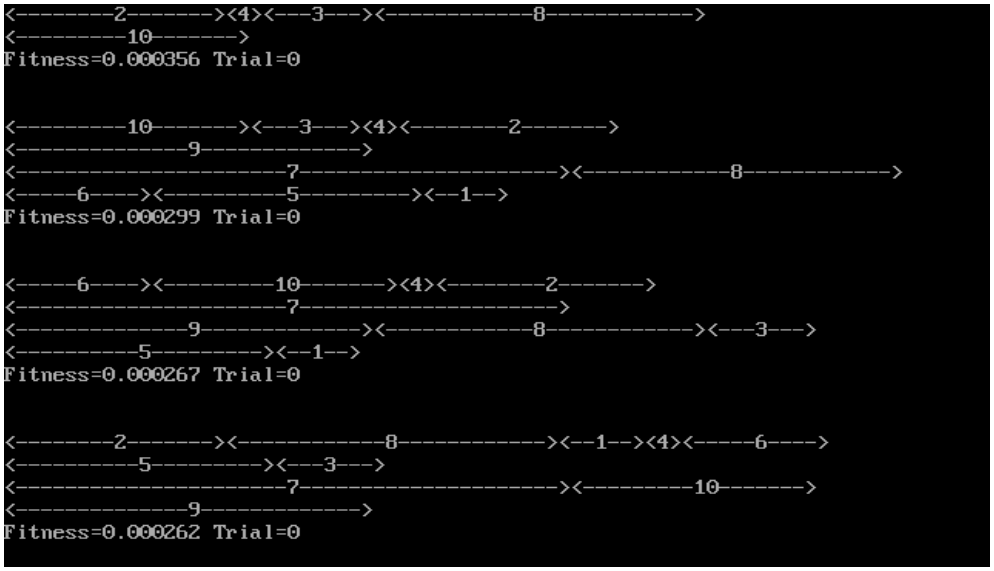


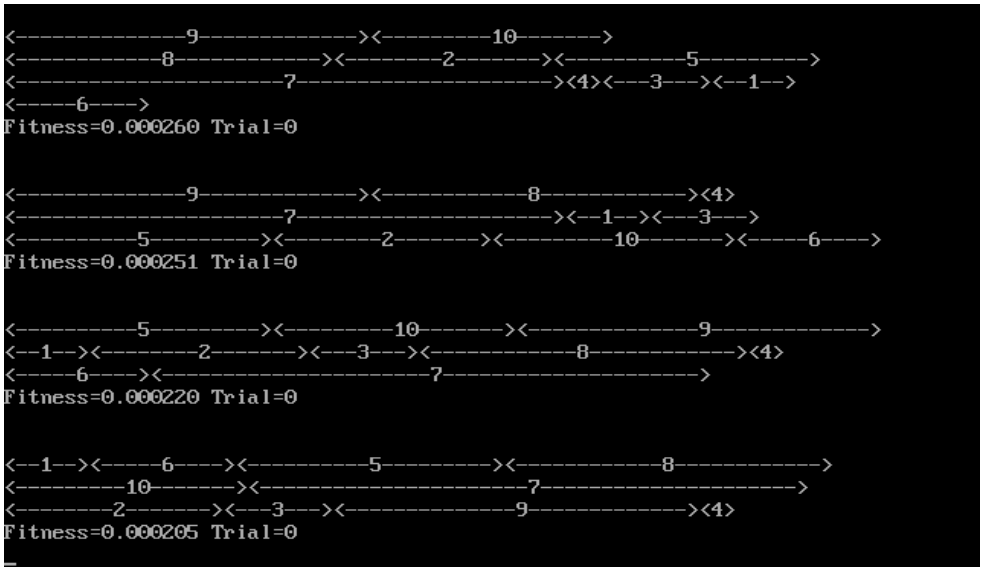

<--1--><---------10-------><----------------------7---------------------->
<--------------9-------------><4><-----6----><------------8------------>
<--------2-------><---3---><----------5---------->
Fitness=0.000219 Trial=0
<------------8------------><--------------9------------->
<--------2-------><----------5---------><--1--><---------10-------><4>
<---3---><-----6----><----------------------7---------------------->
Fitness=0.000209 Trial=0
<------------8------------><----------5--------->
<----------------------7---------------------->
<--------------9-------------><-----6----><4><---3---><--------2------->
<---------10-------><--1-->
Fitness=0.000202 Trial=0
<----------------------7---------------------><-----6----><4>
<------------8------------><----------5---------><--------2------->
<--------------9-------------><---------10------><---3---><--1-->
Fitness=0.000196 Trial=0

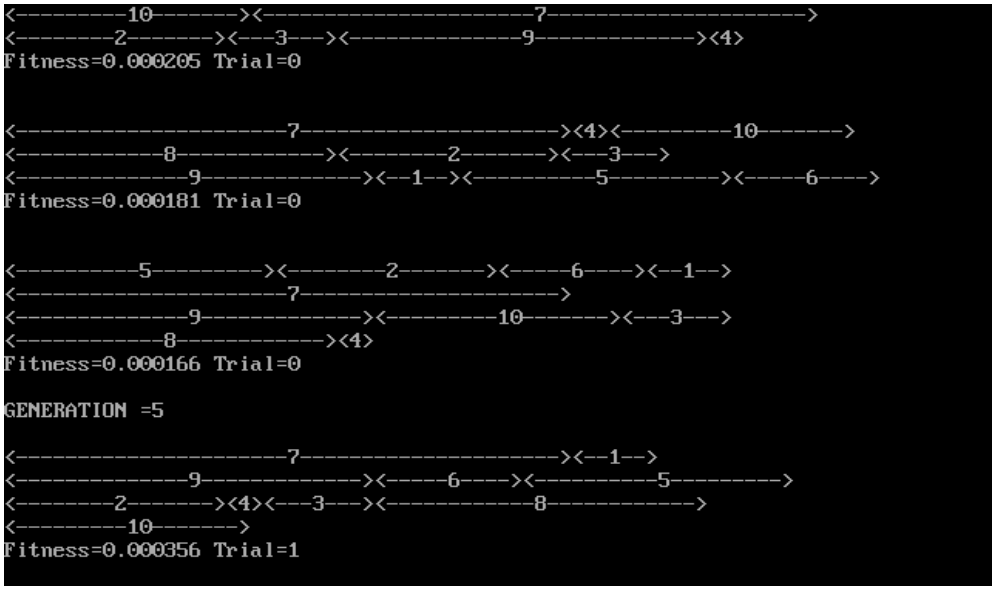

<------------8------------><----------5---------><--------2------->
<--------------9-------------><---------10------><---3---><--1-->
Fitness=0.000196 Trial=0
GENERATION =3
<4><---------10-------><------------8------------><--1--><-----6---->
<----------------------7---------------------->
<--------------9-------------><----------5---------><--------2------->
<---3--->
Fitness=0.000300 Trial=1
<4><--------------9------------->
<----------------------7---------------------><-----6---->
<------------8------------><----------5---------><---------10------->
<--------2-------><--1--><---3--->
Fitness=0.000300 Trial=1
<--------------9-------------><---3---><---------10------><-----6---->
<--1--><--------2-------><----------------------7---------------------->
<4><------------8------------><----------5--------->
Fitness=0.000276 Trial=1

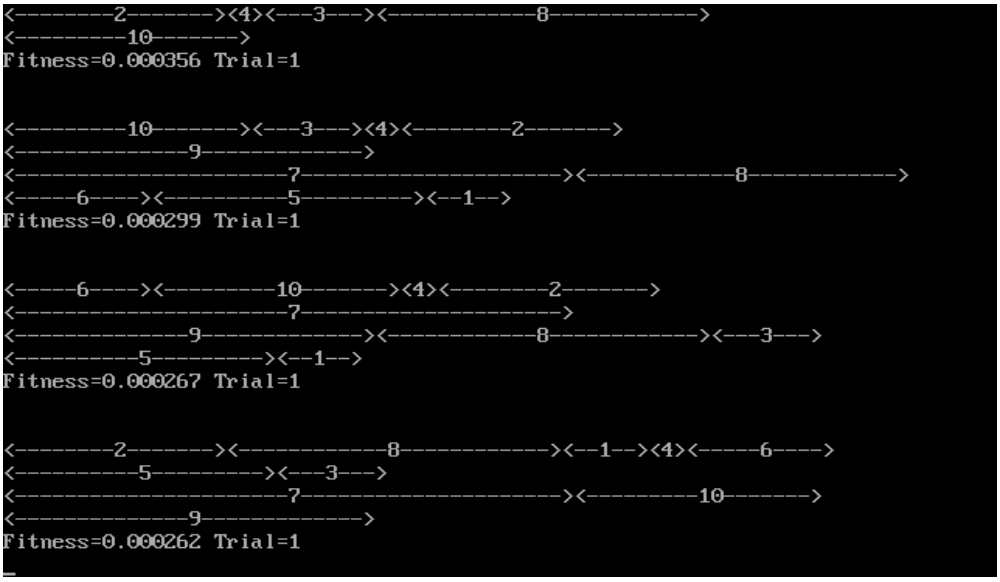

<--------------9-------------><---------10------->
<----------------------7---------------------><------------8------------>
<---3---><--1--><--------2-------><----------5---------><4><-----6---->
Fitness=0.000226 Trial=1
<4><---3---><-----6----><------------8------------>
<--------------9-------------><--1--><----------5--------->
<--------2-------><----------------------7---------------------->
<---------10------->
Fitness=0.000224 Trial=1
<4><------------8------------><--1--><---------10------->
<--------2-------><--------------9-------------><-----6---->
<----------5---------><----------------------7---------------------->
<---3--->
Fitness=0.000221 Trial=1
<--1--><---------10------><----------------------7---------------------->
<--------------9-------------><4><-----6----><------------8------------>
<--------2-------><---3---><----------5---------->
Fitness=0.000219 Trial=1

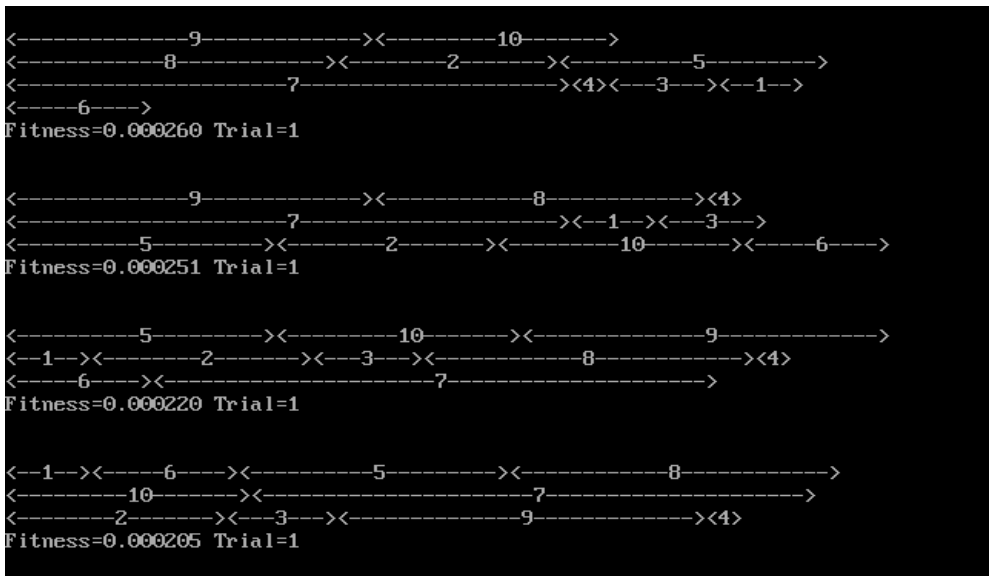

<--------2-------><---------5---------><--1--><---------10------><4>
<---3---><-----6----><----------------------7---------------------->
Fitness=0.000209 Trial=1
<------------8------------><----------5---------->
<----------------------7---------------------->
<--------------9-------------><-----6----><4><---3---><--------2------->
<---------10-------><--1-->
Fitness=0.000202 Trial=1
<----------------------7---------------------><-----6----><4>
<------------8------------><----------5---------><--------2------->
<--------------9-------------><---------10------><---3---><--1-->
Fitness=0.000196 Trial=1
GENERATION =4
<----------------------7---------------------><--1-->
<--------------9-------------><-----6----><----------5---------->
<--------2-------><4><---3---><------------8------------>
<---------10------->
Fitness=0.000356 Trial=0

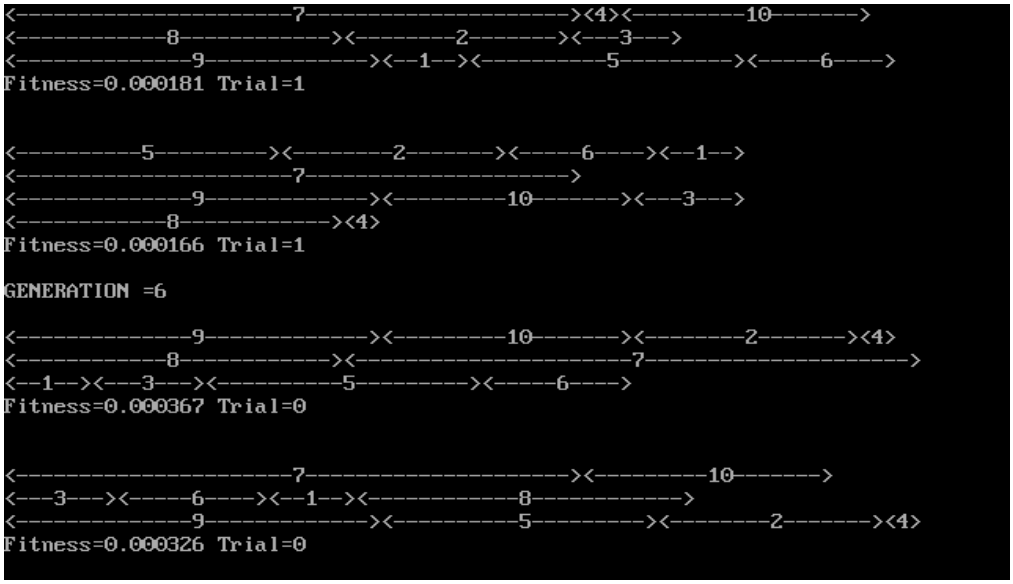

<--------2-------><4><---3---><------------8------------>
<---------10------->
Fitness=0.000356 Trial=0
<---------10-------><---3---><4><--------2------->
<--------------9-------------->
<----------------------7---------------------><------------8------------>
<-----6----><----------5---------><--1-->
Fitness=0.000299 Trial=0
<-----6----><----------10-------><4><--------2------->
<----------------------7---------------------->
<--------------9-------------><------------8------------><---3--->
<----------5---------><--1-->
Fitness=0.000267 Trial=0
<--------2-------><------------8------------><--1--><4><-----6---->
<----------5---------><---3--->
<----------------------7---------------------><---------10------->
<--------------9------------->
Fitness=0.000262 Trial=0

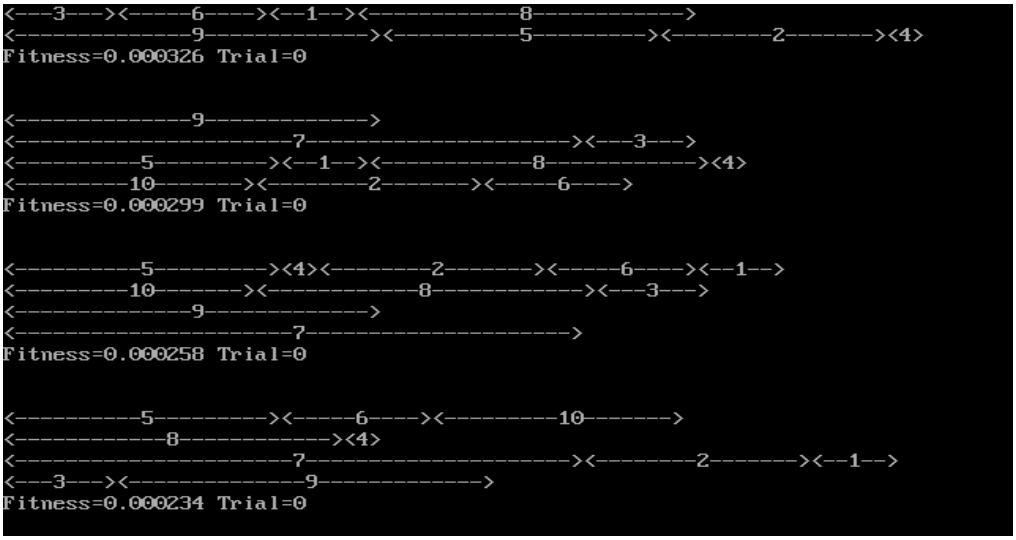

<--------------9-------------><----------10------->
<------------8------------><--------2-------><----------5--------->
<----------------------7---------------------><4><---3---><--1-->
<-----6---->
Fitness=0.000260 Trial=0
<--------------9-------------><------------8------------><4>
<----------------------7---------------------><--1--><---3--->
<----------5---------><--------2-------><---------10------><-----6---->
Fitness=0.000251 Trial=0
<----------5---------><---------10------><--------------9------------->
<--1--><--------2-------><---3---><------------8------------><4>
<-----6----><----------------------7---------------------->
Fitness=0.000220 Trial=0
<--1--><-----6----><----------5---------><------------8------------>
<---------10-------><----------------------7---------------------->
<--------2-------><---3---><--------------9------------><4>
Fitness=0.000205 Trial=0


<---------10-------><----------------------7---------------------->
<--------2-------><---3---><--------------9-------------><4>
Fitness=0.000205 Trial=0
<----------------------7---------------------><4><---------10------->
<------------8------------><--------2-------><---3--->
<--------------9-------------><--1--><----------5---------><-----6---->
Fitness=0.000181 Trial=0
<----------5---------><--------2-------><-----6----><--1-->
<----------------------7---------------------->
<--------------9-------------><---------10------><---3--->
<------------8------------><4>
Fitness=0.000166 Trial=0
GENERATION =5
<----------------------7---------------------><--1-->
<--------------9-------------><-----6----><----------5--------->
<--------2-------><4><---3---><------------8------------>
<---------10------->
Fitness=0.000356 Trial=1

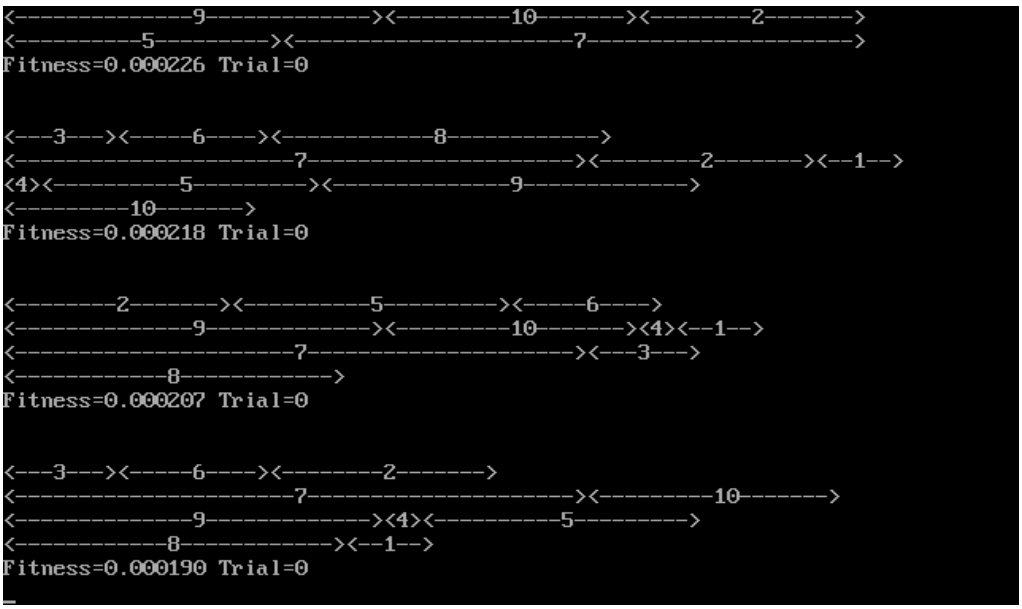

<--------2-------><4><---3---><------------8------------>
<---------10------->
Fitness=0.000356 Trial=1
<---------10-------><---3---><4><--------2------->
<--------------9------------->
<----------------------7---------------------><------------8------------>
<-----6----><----------5---------><--1-->
Fitness=0.000299 Trial=1
<-----6----><----------10-------><4><--------2------->
<----------------------7---------------------->
<--------------9-------------><------------8------------><---3--->
<----------5---------><--1-->
Fitness=0.000267 Trial=1
<--------2-------><------------8------------><--1--><4><-----6---->
<----------5---------><---3--->
<----------------------7---------------------><---------10------->
<--------------9------------->
Fitness=0.000262 Trial=1


<--------------9-------------><----------10------->
<------------8------------><--------2-------><----------5--------->
<----------------------7---------------------><4><---3---><--1-->
<-----6---->
Fitness=0.000260 Trial=1
<--------------9-------------><------------8------------><4>
<----------------------7---------------------><--1--><---3--->
<----------5---------><---------2-------><---------10------><-----6---->
Fitness=0.000251 Trial=1
<----------5---------><----------10-------><--------------9------------->
<--1--><--------2-------><---3---><------------8------------><4>
<-----6----><----------------------7---------------------->
Fitness=0.000220 Trial=1
<--1--><-----6----><----------5---------><------------8------------>
<---------10-------><----------------------7---------------------->
<--------2-------><---3---><--------------9------------><4>
Fitness=0.000205 Trial=1





Fitness=0.000258 Trial=0
Fitness=0.000234 Trial=0

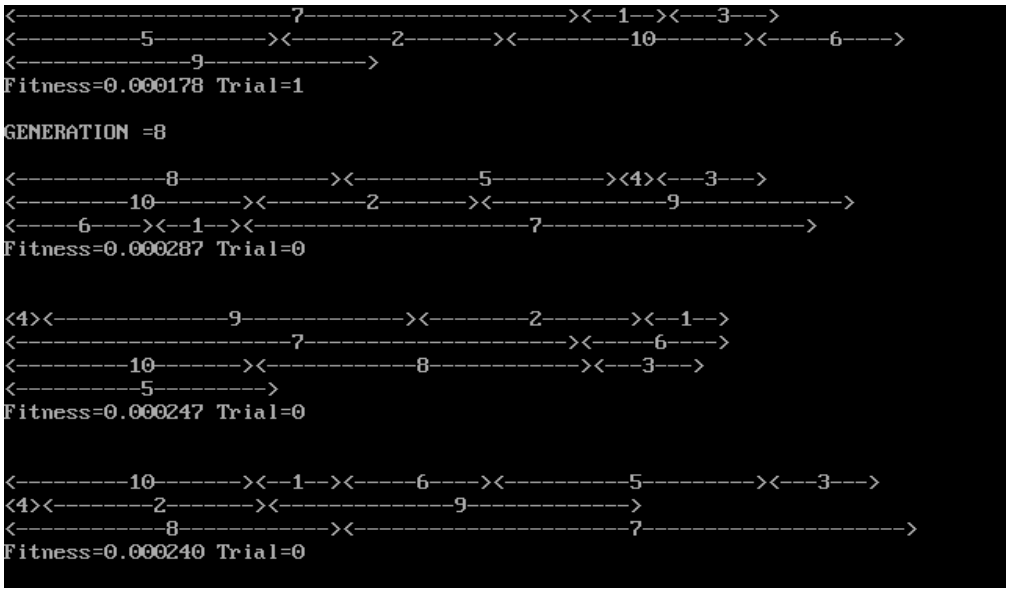


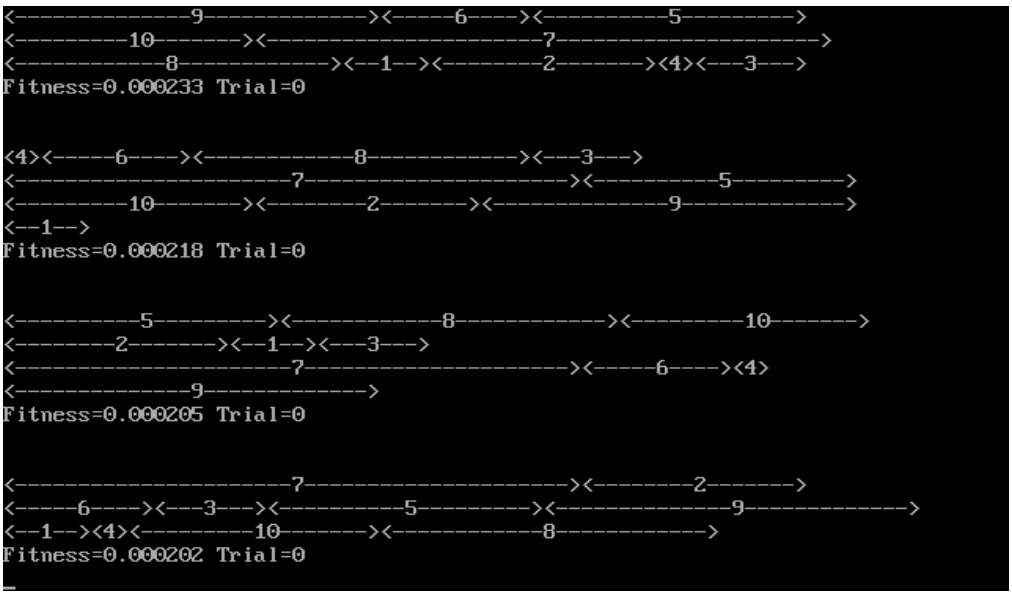
Fitness=0.000226 Trial=0
Fitness=0.000218 Trial=0
Fitness=0.000207 Trial=0
Fitness=0.000190 Trial=0

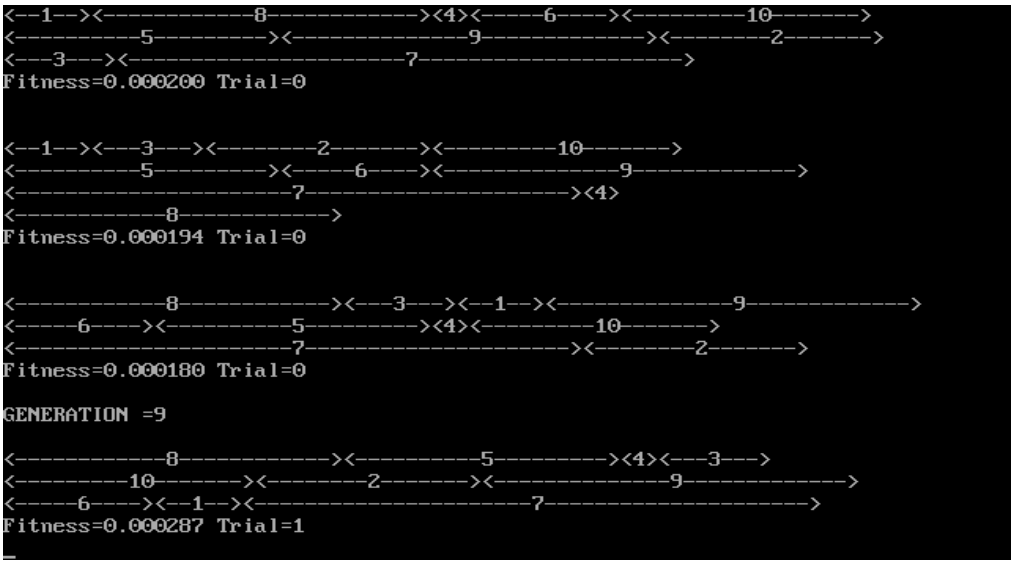
```
<----------------------7---------------------><--1--><---3--->
<----------5---------><--------2-------><---------10-------><-----6---->
<--------------9------------->
Fitness=0.000178 Trial=0

GENERATION =7

<--------------9------------><---------10-------><--------2-------><4>
<------------8------------><----------------------7---------------------->
<--1--><---3---><----------5---------><-----6---->
Fitness=0.000367 Trial=1


<----------------------7---------------------><---------10------->
<---3---><-----6----><--1--><------------8------------>
<--------------9-------------><----------5---------><--------2-------><4>
Fitness=0.000326 Trial=1


<--------------9------------->
<----------------------7---------------------><---3--->
<----------5---------><--1--><------------8------------><4>
<---------10-------><--------2-------><-----6---->
Fitness=0.000299 Trial=1
_
```

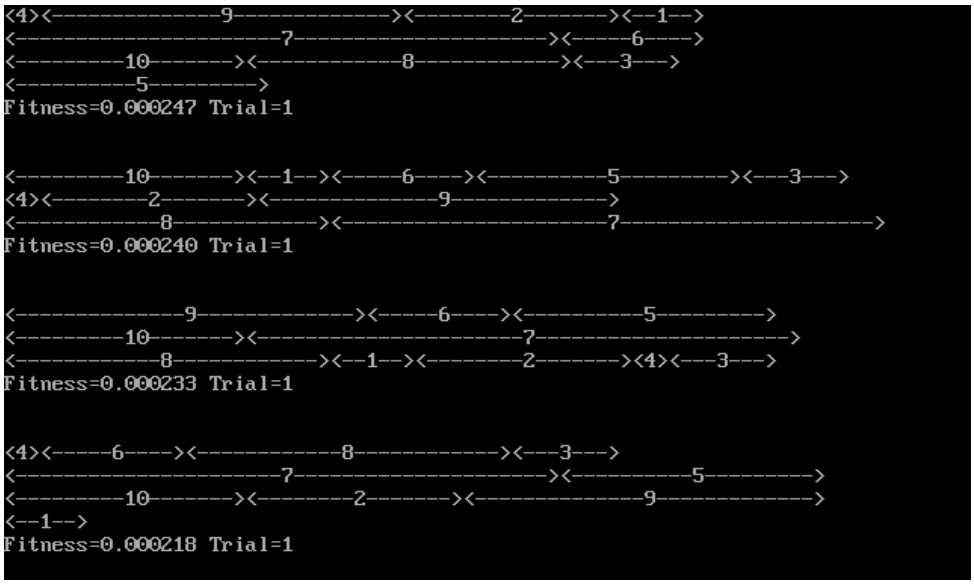
```
<---------10-------><------------8------------><---3--->
<--------------9------------->
<----------------------7---------------------->
Fitness=0.000258 Trial=1


<----------5---------><-----6----><---------10------->
<------------8------------><4>
<----------------------7---------------------><--------2-------><--1-->
<---3---><--------------9------------->
Fitness=0.000234 Trial=1


<---3---><------------8------------><-----6----><4><--1-->
<--------------9-------------><---------10-------><--------2------->
<----------5---------><----------------------7---------------------->
Fitness=0.000226 Trial=1


<---3---><-----6----><------------8------------>
<----------------------7---------------------><--------2-------><--1-->
<4><----------5---------><--------------9------------->
<---------10------->
Fitness=0.000218 Trial=1
```

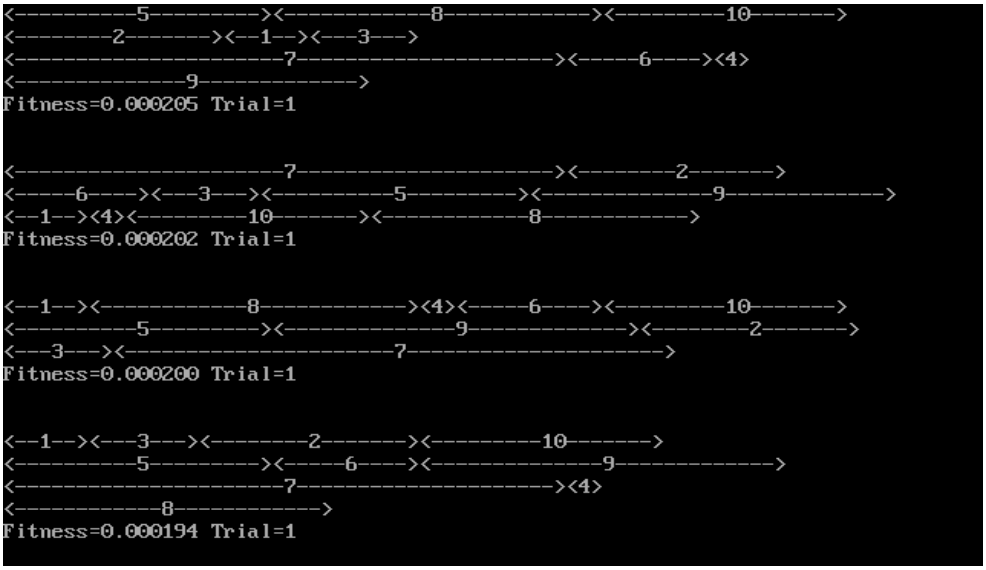
```
<4><----------5---------><--------------9------------->
<---------10------->
Fitness=0.000218 Trial=1


<--------2-------><----------5---------><-----6---->
<--------------9-------------><---------10-------><4><--1-->
<----------------------7---------------------><---3--->
<------------8------------>
Fitness=0.000207 Trial=1


<---3---><-----6----><--------2------->
<----------------------7---------------------><---------10------->
<--------------9-------------><4><----------5--------->
<------------8------------><--1-->
Fitness=0.000190 Trial=1


<4><------------8------------>
<----------------------7---------------------><--1--><---3--->
<----------5---------><--------2-------><---------10-------><-----6---->
<--------------9------------->
Fitness=0.000178 Trial=1
_
```

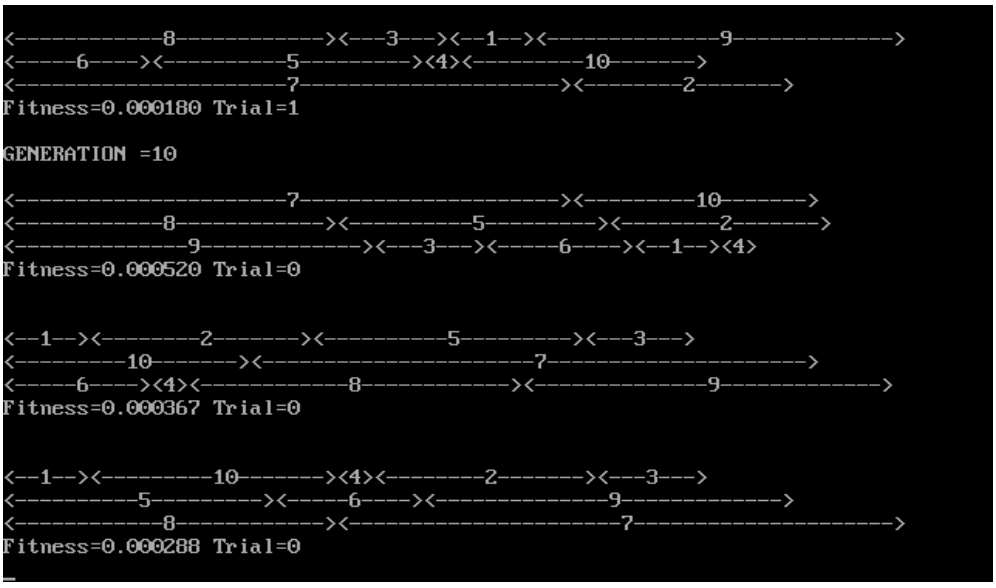

<----------7----------><--1--><---3--->
<----------5---------><--------2-------><---------10-------><-----6---->
<--------------9------------->
Fitness=0.000178 Trial=1
GENERATION =8
<------------8------------><----------5---------><4><---3--->
<---------10-------><--------2-------><--------------9------------->
<-----6----><--1--><----------7---------->
Fitness=0.000287 Trial=0
<4><--------------9------------><--------2-------><--1-->
<----------7---------><-----6---->
<---------10-------><------------8------------><---3--->
<----------5--------->
Fitness=0.000247 Trial=0
<---------10-------><--1--><-----6----><----------5---------><---3--->
<4><--------2-------><--------------9------------->
<------------8------------><----------7---------->
Fitness=0.000240 Trial=0

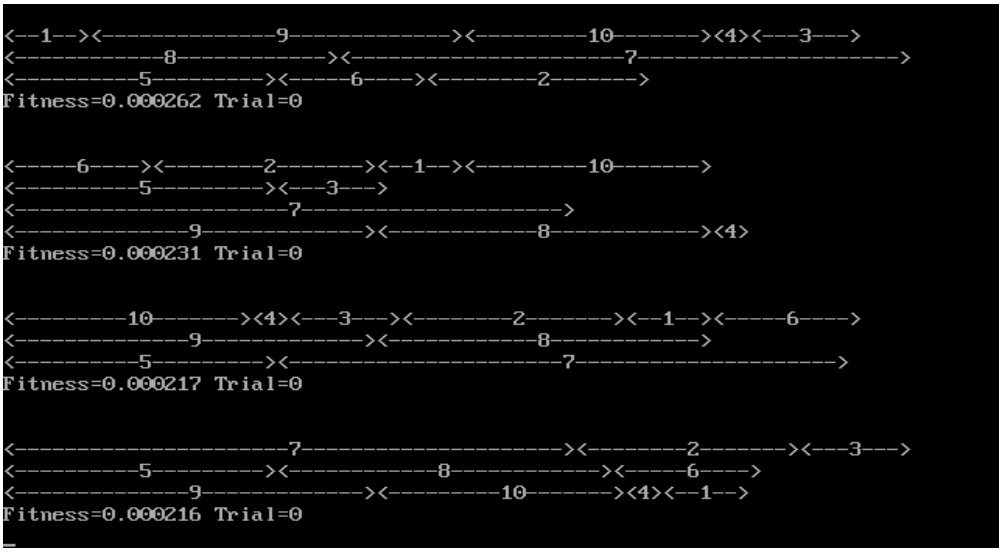

<--------------9------------><-----6----><----------5--------->
<---------10-------><----------7---------->
<------------8------------><--1--><--------2-------><4><---3--->
Fitness=0.000233 Trial=0
<4><-----6----><------------8------------><---3--->
<----------7---------><----------5--------->
<---------10-------><--------2-------><--------------9------------->
<--1-->
Fitness=0.000218 Trial=0
<----------5--------><------------8------------><---------10------->
<--------2-------><--1--><---3--->
<----------7---------><-----6----><4>
<--------------9------------->
Fitness=0.000205 Trial=0
<----------7---------><--------2------->
<-----6----><---3---><----------5--------><--------------9------------->
<--1--><4><---------10-------><------------8------------>
Fitness=0.000202 Trial=0

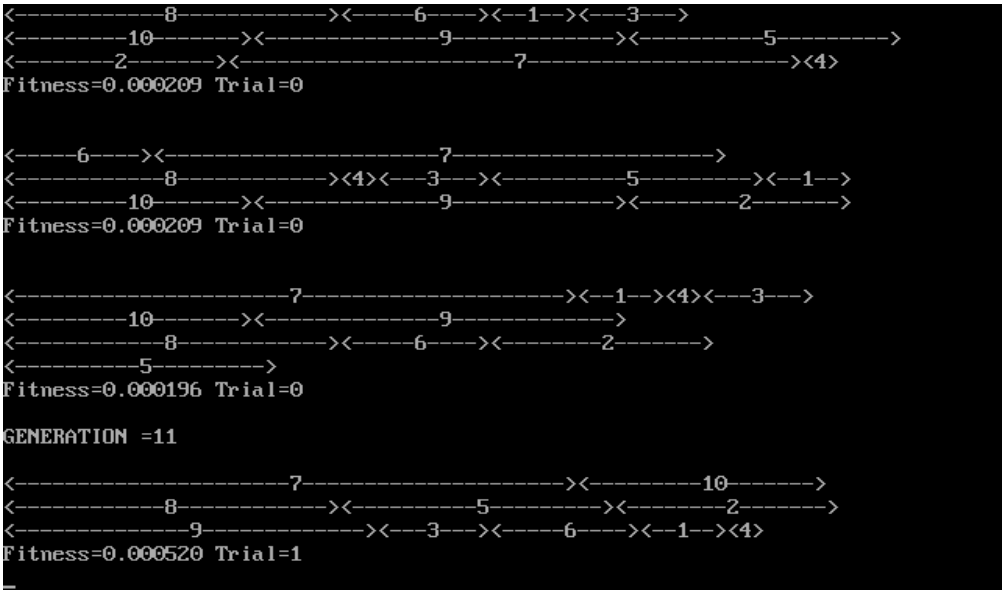

<--1--><------------8------------><4><-----6----><---------10------->
<----------5---------><--------------9------------><--------2------->
<---3---><----------7---------->
Fitness=0.000200 Trial=0
<--1--><---3---><--------2-------><---------10------->
<----------5---------><-----6----><--------------9------------->
<----------7---------><4>
<------------8------------>
Fitness=0.000194 Trial=0
<------------8------------><---3---><--1--><--------------9------------->
<-----6----><----------5---------><4><---------10------->
<----------7---------><--------2------->
Fitness=0.000180 Trial=0
GENERATION =9
<------------8------------><----------5---------><4><---3--->
<---------10-------><--------2-------><--------------9------------->
<-----6----><--1--><----------7---------->
Fitness=0.000287 Trial=1

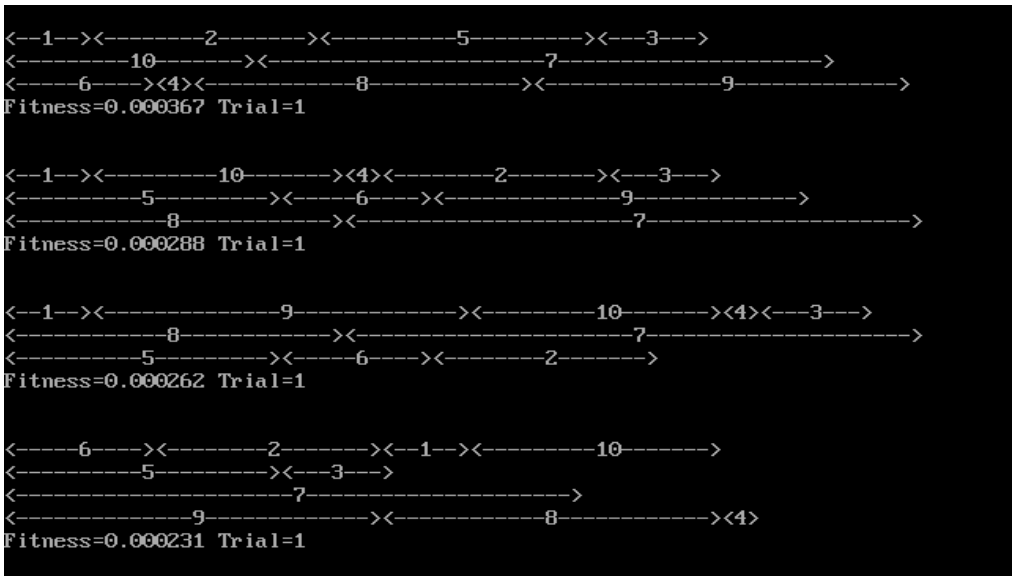

<4><----------9----------><--------2-------><--1-->
<----------------------7---------------------><-----6---->
<---------10-------><------------8------------><---3--->
<----------5--------->
Fitness=0.000247 Trial=1
<---------10-------><--1--><-----6----><----------5---------><---3--->
<4><--------2-------><--------------9------------->
<------------8------------><----------------------7---------------------->
Fitness=0.000240 Trial=1
<--------------9-------------><-----6----><----------5--------->
<---------10-------><----------------------7---------------------->
<------------8------------><--1--><--------2-------><4><---3--->
Fitness=0.000233 Trial=1
<4><-----6----><------------8------------><---3--->
<----------------------7---------------------><----------5--------->
<---------10-------><--------2-------><--------------9------------->
<--1-->
Fitness=0.000218 Trial=1

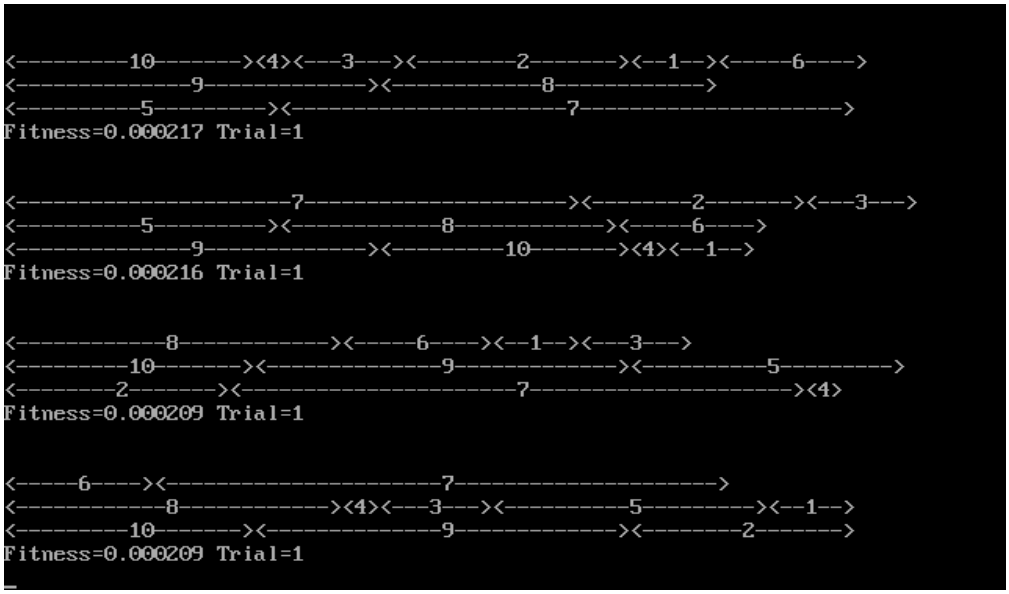

<----------5---------><------------8------------><---------10------->
<--------2-------><--1--><---3--->
<----------------------7---------------------><-----6----><4>
<--------------9------------->
Fitness=0.000205 Trial=1
<----------------------7---------------------><--------2------->
<-----6----><---3---><----------5---------><--------------9------------->
<--1--><4><----------10-------><------------8------------>
Fitness=0.000202 Trial=1
<--1--><------------8------------><4><-----6----><---------10------->
<----------5---------><--------------9------------><--------2------->
<---3---><----------------------7---------------------->
Fitness=0.000200 Trial=1
<--1--><---3---><--------2-------><---------10------->
<----------5---------><-----6----><--------------9------------->
<----------------------7---------------------><4>
<------------8------------>
Fitness=0.000194 Trial=1

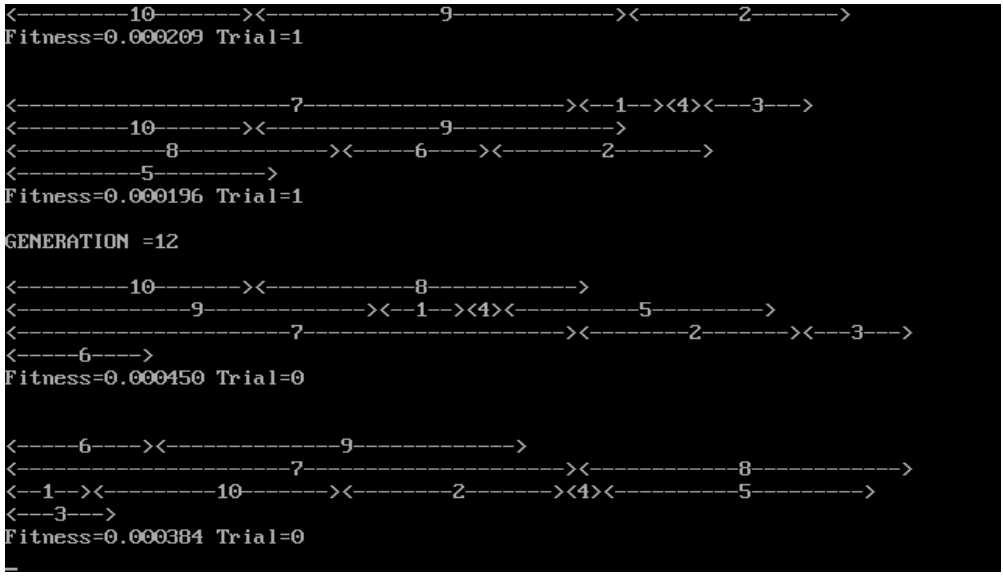

<------------8------------><---3---><--1--><--------------9------------->
<-----6----><----------5---------><4><---------10------->
<----------------------7---------------------><--------2------->
Fitness=0.000180 Trial=1
GENERATION =10
<----------------------7---------------------><---------10------->
<------------8------------><----------5---------><--------2------->
<--------------9------------><---3---><-----6----><--1--><4>
Fitness=0.000520 Trial=0
<--1--><--------2-------><----------5---------><---3--->
<---------10-------><----------------------7---------------------->
<-----6----><4><------------8------------><--------------9------------->
Fitness=0.000367 Trial=0
<--1--><---------10-------><4><--------2-------><---3--->
<----------5---------><-----6----><--------------9------------->
<------------8------------><----------------------7---------------------->
Fitness=0.000288 Trial=0

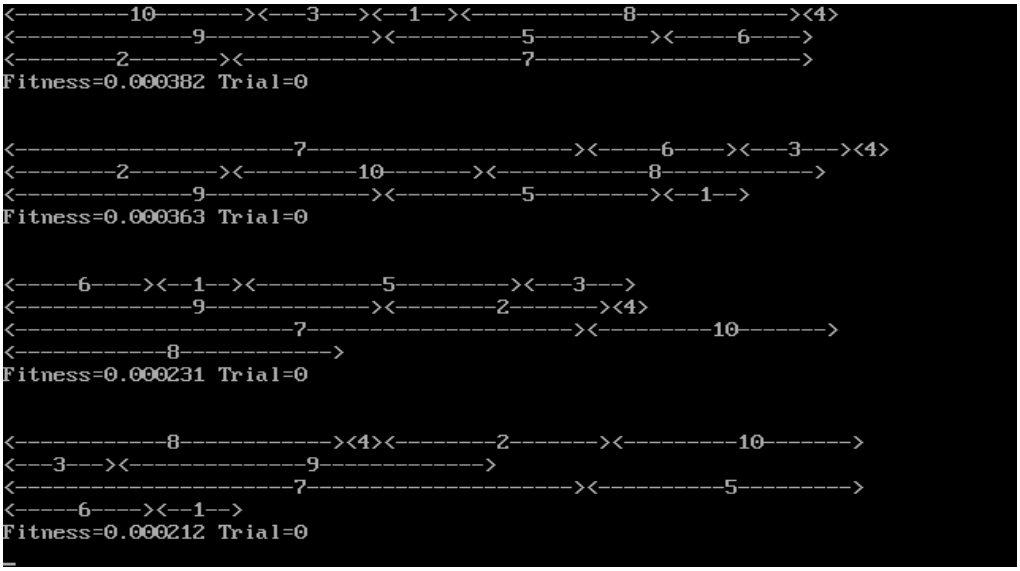

<--1--><----------9----------><---------10-------><4><---3--->
<----------8----------><----------------------7---------------------->
<----------5---------><-----6----><--------2------->
Fitness=0.000262 Trial=0
<-----6----><--------2-------><--1--><---------10------->
<----------5---------><---3--->
<----------------------7---------------------->
<----------9----------><----------8----------><4>
Fitness=0.000231 Trial=0
<---------10-------><4><---3---><--------2-------><--1--><-----6---->
<----------9----------><----------8---------->
<----------5---------><----------------------7---------------------->
Fitness=0.000217 Trial=0
<----------------------7---------------------><--------2-------><---3--->
<----------5---------><----------8----------><-----6---->
<----------9----------><---------10-------><4><--1-->
Fitness=0.000216 Trial=0

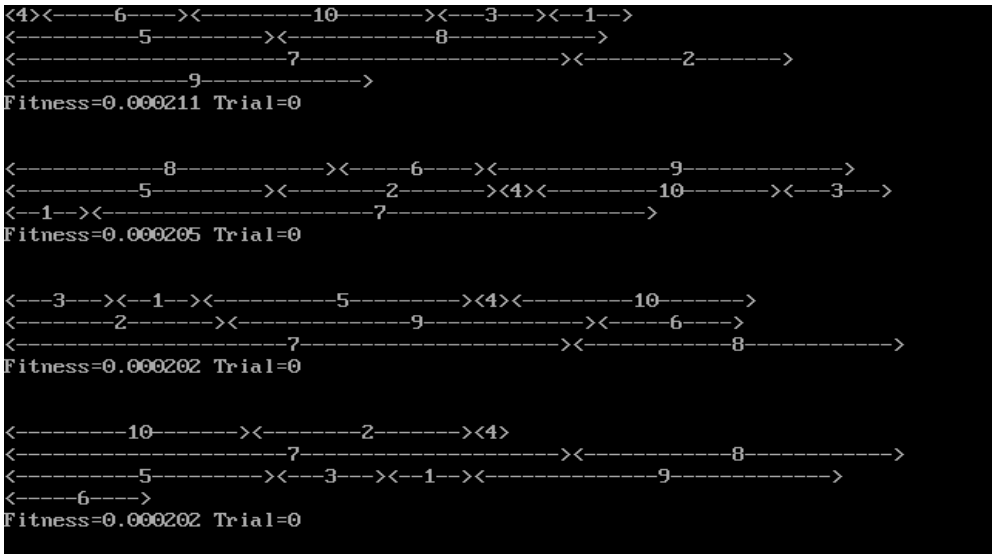

<----------8----------><-----6----><--1--><---3--->
<---------10-------><----------9----------><----------5--------->
<--------2-------><----------------------7---------------------><4>
Fitness=0.000209 Trial=0
<-----6----><----------------------7---------------------->
<----------8----------><4><---3---><----------5---------><--1-->
<---------10-------><----------9----------><--------2------->
Fitness=0.000209 Trial=0
<----------------------7---------------------><--1--><4><---3--->
<---------10-------><----------9---------->
<----------8----------><-----6----><--------2------->
<----------5--------->
Fitness=0.000196 Trial=0
GENERATION =11
<----------------------7---------------------><---------10------->
<----------8----------><----------5---------><--------2------->
<----------9----------><---3---><-----6----><--1--><4>
Fitness=0.000520 Trial=1

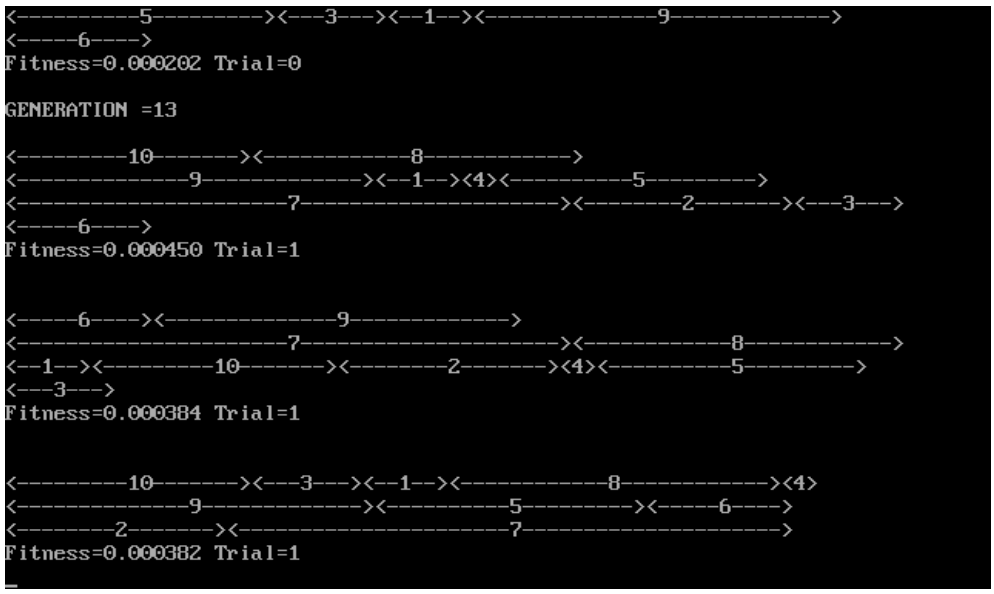

<--1--><--------2-------><----------5---------><---3--->
<---------10-------><----------------------7---------------------->
<-----6----><4><----------8----------><----------9---------->
Fitness=0.000367 Trial=1
<--1--><---------10-------><4><--------2-------><---3--->
<----------5---------><-----6----><----------9---------->
<----------8----------><----------------------7---------------------->
Fitness=0.000288 Trial=1
<--1--><----------9----------><---------10-------><4><---3--->
<----------8----------><----------------------7---------------------->
<----------5---------><-----6----><--------2------->
Fitness=0.000262 Trial=1
<-----6----><--------2-------><--1--><---------10------->
<----------5---------><---3--->
<----------------------7---------------------->
<----------9----------><----------8----------><4>
Fitness=0.000231 Trial=1

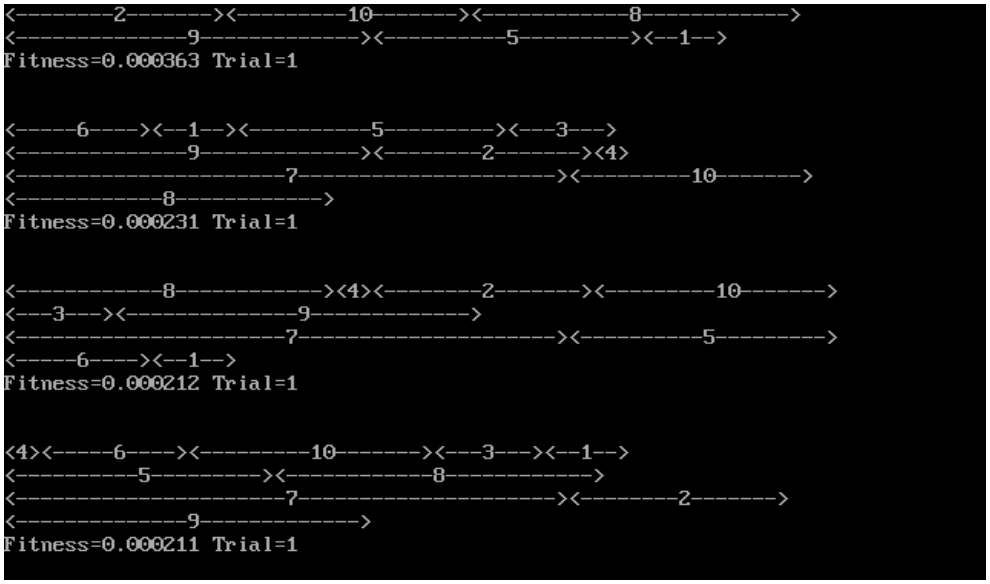

<---------10-------><4><---3---><--------2-------><--1--><-----6---->
<-------------9------------><-----------8----------->
<----------5---------><----------------------7---------------------->
Fitness=0.000217 Trial=1
<----------------------7---------------------><--------2-------><---3--->
<----------5---------><------------8----------><-----6---->
<-------------9------------><---------10-------><4><--1-->
Fitness=0.000216 Trial=1
<------------8------------><-----6----><--1--><---3--->
<---------10-------><--------------9-------------><----------5--------->
<--------2-------><----------------------7---------------------><4>
Fitness=0.000209 Trial=1
<-----6----><----------------------7---------------------->
<------------8------------><4><---3---><----------5---------><--1-->
<---------10-------><--------------9-------------><--------2------->
Fitness=0.000209 Trial=1

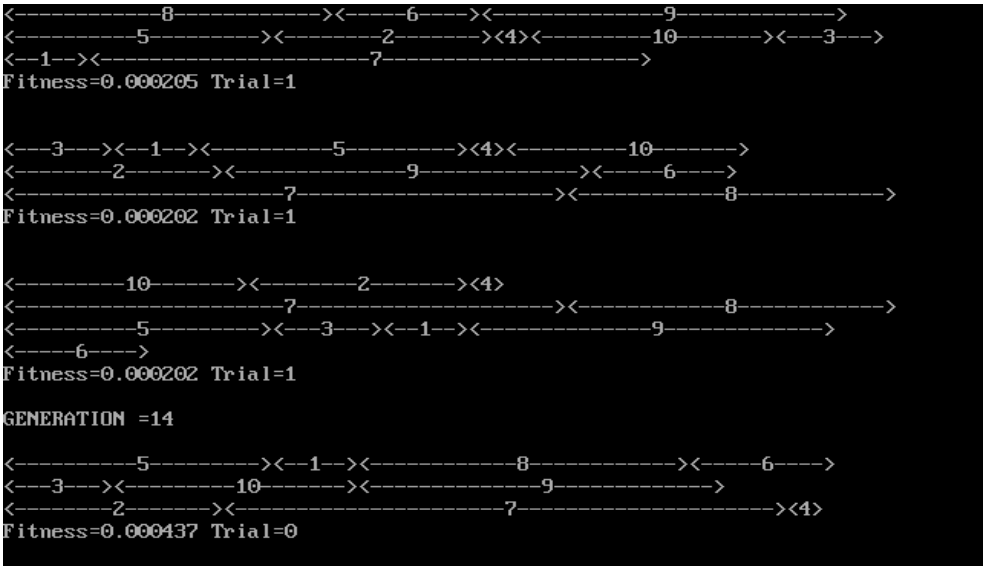

<---------10-------><--------------9-------------><--------2------->
Fitness=0.000209 Trial=1
<----------------------7---------------------><--1--><4><---3--->
<---------10-------><--------------9------------->
<------------8------------><-----6----><--------2------->
<----------5---------->
Fitness=0.000196 Trial=1
GENERATION =12
<---------10-------><------------8------------>
<---------------9-------------><--1--><4><----------5--------->
<----------------------7---------------------><--------2-------><---3--->
<-----6---->
Fitness=0.000450 Trial=0
<-----6----><--------------9------------->
<----------------------7---------------------><------------8------------>
<--1--><---------10-------><--------2-------><4><----------5--------->
<---3--->
Fitness=0.000384 Trial=0


<---------10-------><---3---><--1--><------------8------------><4>
<---------------9-------------><----------5---------><-----6---->
<--------2-------><----------------------7---------------------->
Fitness=0.000382 Trial=0
<----------------------7---------------------><-----6----><---3---><4>
<--------2-------><----------10-------><------------8------------>
<---------------9-------------><----------5---------><--1-->
Fitness=0.000363 Trial=0
<-----6----><--1--><----------5---------><---3--->
<--------------9-------------><--------2-------><4>
<----------------------7---------------------><---------10------->
<------------8------------>
Fitness=0.000231 Trial=0
<------------8------------><4><--------2-------><---------10------->
<---3---><--------------9------------->
<----------------------7---------------------><----------5--------->
<-----6----><--1-->
Fitness=0.000212 Trial=0


<4><-----6----><---------10-------><---3---><--1-->
<----------5---------><------------8------------>
<----------------------7---------------------><--------2------->
<---------------9------------->
Fitness=0.000211 Trial=0
<------------8------------><-----6----><---------------9------------->
<----------5---------><--------2-------><4><---------10-------><---3--->
<--1--><---------------------7---------------------->
Fitness=0.000205 Trial=0
<---3---><--1--><----------5---------><4><---------10------->
<--------2------->
<----------------------7---------------------><------------8------------>
Fitness=0.000202 Trial=0
<---------10------->
<----------------------7---------------------><------------8------------>
<----------5---------><---3---><--1--><---------------9------------->
<-----6---->
Fitness=0.000202 Trial=0


<----------5---------><---3---><--1--><---------------9------------->
<-----6---->
Fitness=0.000202 Trial=0
GENERATION =13
<---------10------->
<---------------9------------><--1--><4><----------5--------->
<----------------------7---------------------><--------2------><---3--->
<-----6---->
Fitness=0.000450 Trial=1
<-----6---->
<----------------------7---------------------><------------8------------>
<--1--><---------10------><--------2------><4><----------5--------->
<---3--->
Fitness=0.000384 Trial=1
<---------10------><---3---><--1--><------------8------------><4>
<---------------9------------><----------5---------><-----6---->
<--------2------><----------------------7---------------------->
Fitness=0.000382 Trial=1


<--------2-------><---------10------><------------8------------>
<---------------9------------><----------5---------><--1-->
Fitness=0.000363 Trial=1
<-----6----><--1--><----------5---------><---3--->
<---------------9------------><--------2-------><4>
<----------------------7---------------------><---------10------>
<------------8------------>
Fitness=0.000231 Trial=1
<------------8------------><4><---------2-------><---------10------>
<---3---><---------------9------------->
<----------------------7---------------------><----------5--------->
<-----6----><--1-->
Fitness=0.000212 Trial=1
<4><-----6----><---------10------><---3---><--1-->
<----------5---------><------------8------------>
<----------------------7---------------------><--------2------->
<---------------9------------->
Fitness=0.000211 Trial=1



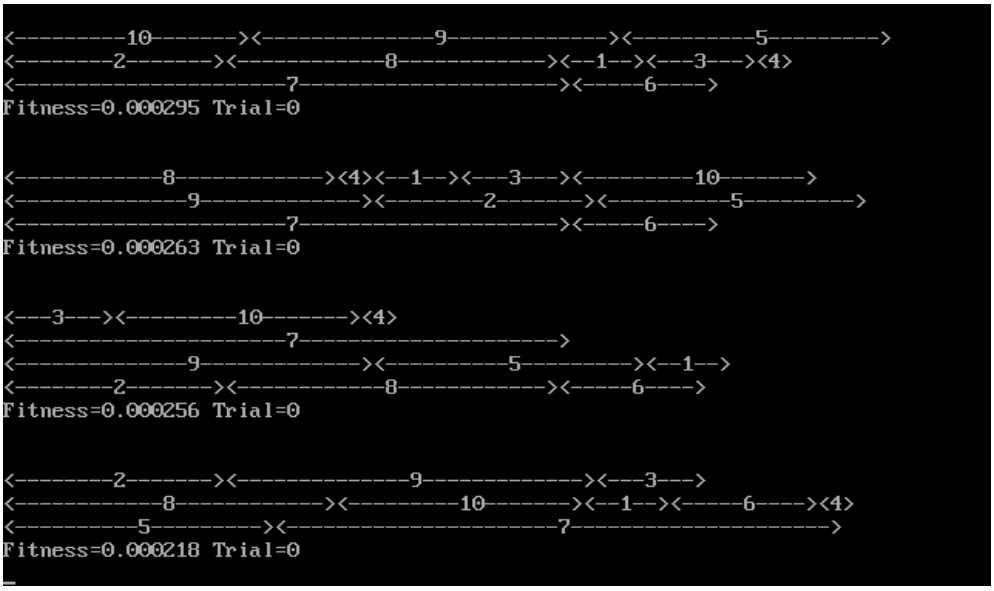
Fitness=0.000205 Trial=1

Fitness=0.000202 Trial=1

Fitness=0.000202 Trial=1

GENERATION =14

Fitness=0.000437 Trial=0

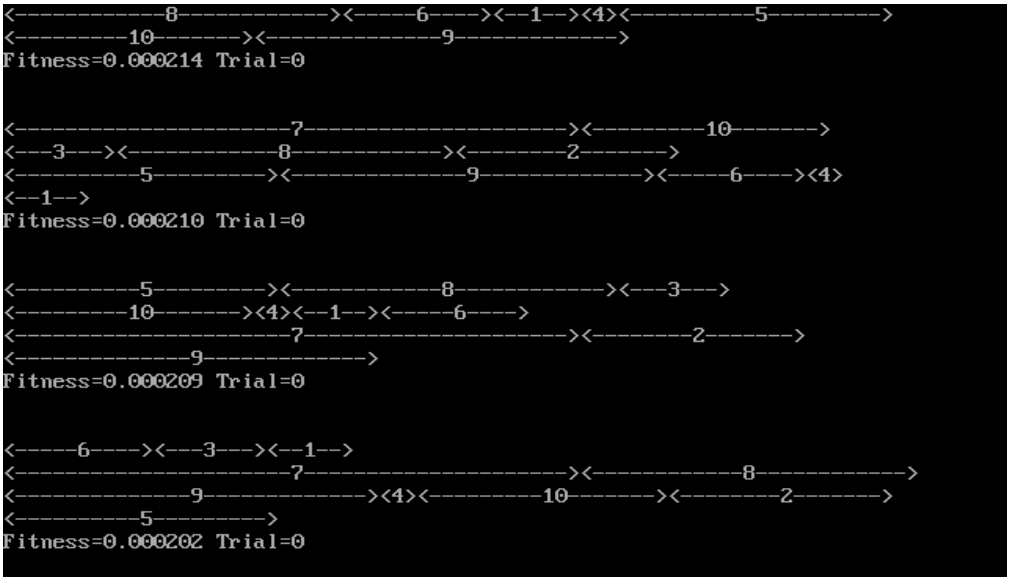


Fitness=0.000295 Trial=0

Fitness=0.000263 Trial=0

Fitness=0.000256 Trial=0

Fitness=0.000218 Trial=0



Fitness=0.000214 Trial=0

Fitness=0.000210 Trial=0

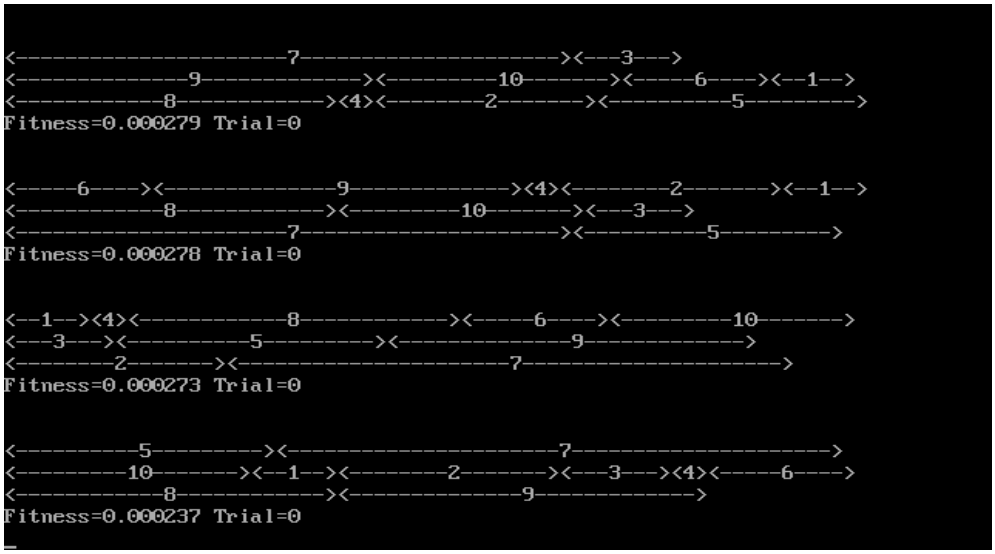
Fitness=0.000209 Trial=0

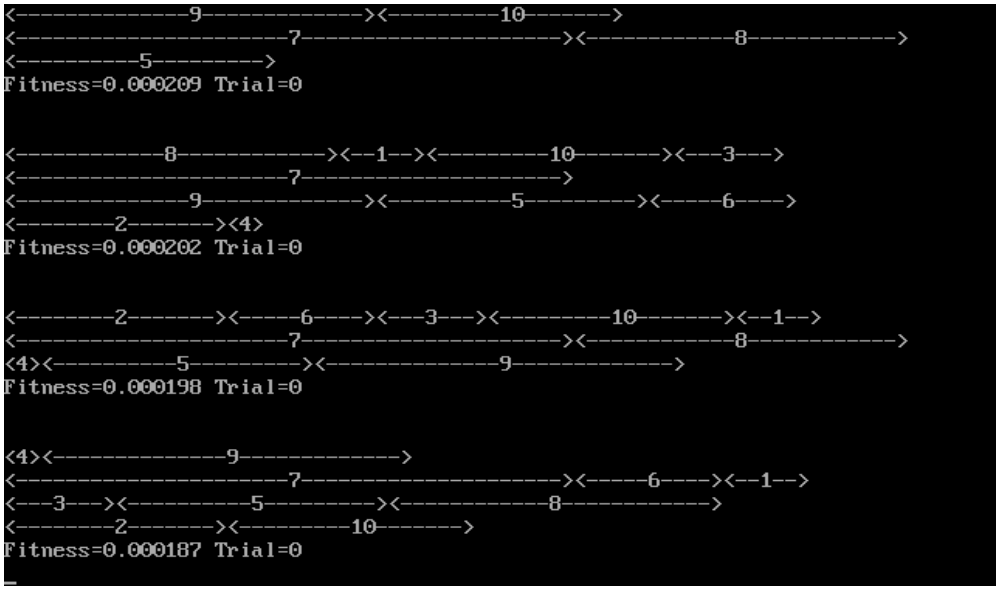
Fitness=0.000202 Trial=0

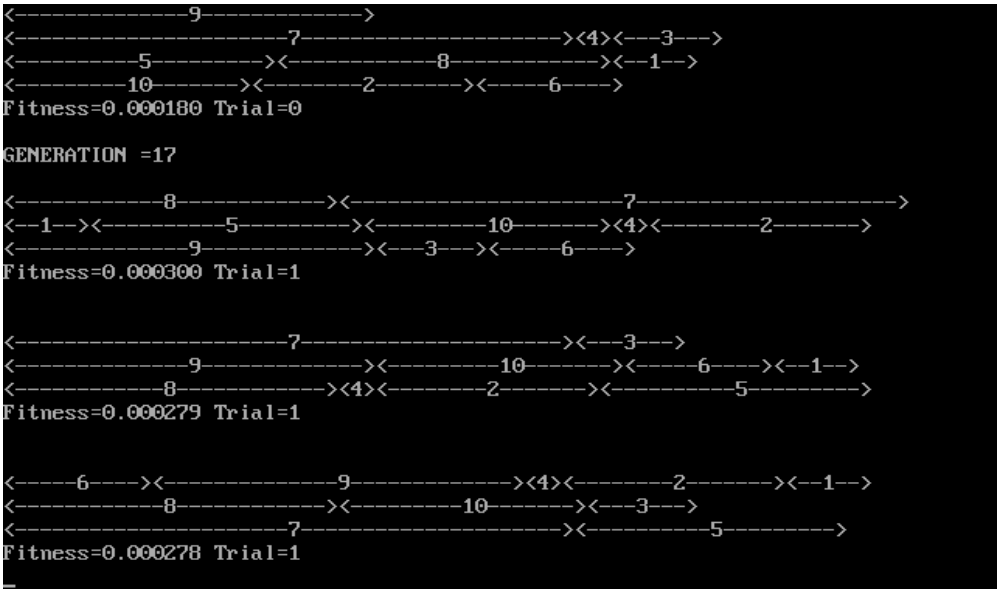
```
<--1--><----------5---------><---3---><-----6---->
<----------------------7---------------------><------------8------------>
<--------------9-------------><----------10-------><--------2-------><4>
Fitness=0.000192 Trial=0

GENERATION =15

<----------5---------><--1--><------------8------------><-----6---->
<---3---><---------10-------><--------------9------------->
<--------2-------><----------------------7---------------------><4>
Fitness=0.000437 Trial=1


<----------10-------><--------------9-------------><----------5--------->
<--------2-------><------------8------------><--1--><---3---><4>
<----------------------7---------------------><-----6---->
Fitness=0.000295 Trial=1


<------------8------------><4><--1--><---3---><----------10------->
<--------------9-------------><---------2-------><----------5--------->
<----------------------7---------------------><-----6---->
Fitness=0.000263 Trial=1
_
```

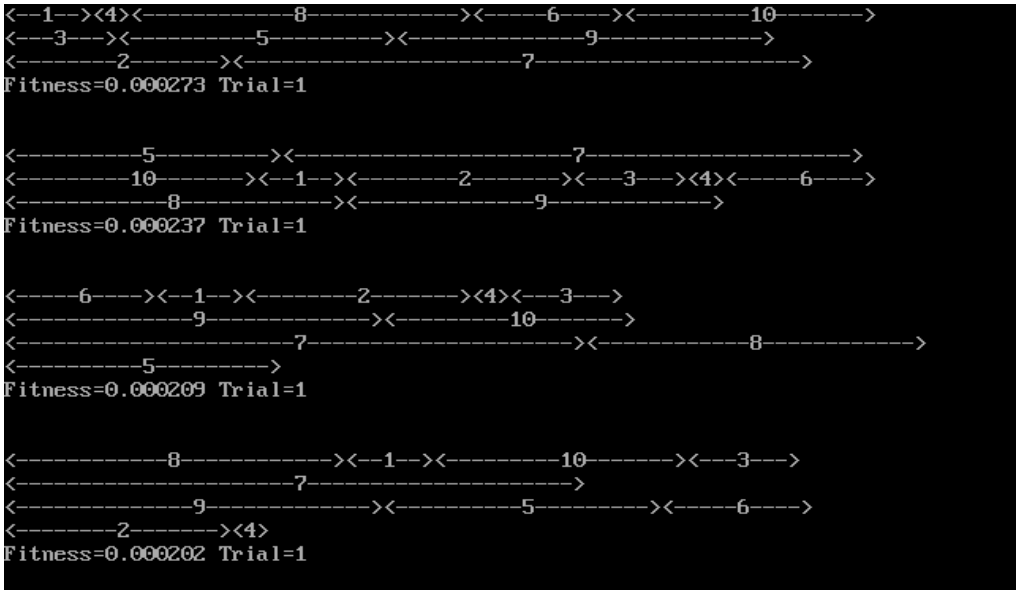
```
<---3---><----------10-------><4>
<----------------------7---------------------->
<--------------9-------------><----------5---------><--1-->
<--------2-------><------------8------------><-----6---->
Fitness=0.000256 Trial=1


<--------2-------><--------------9-------------><---3--->
<------------8------------><----------10-------><--1--><-----6----><4>
<----------5---------><----------------------7---------------------->
Fitness=0.000218 Trial=1


<---3---><----------------------7---------------------><--------2------->
<------------8------------><-----6----><--1--><4><----------5--------->
<----------10-------><--------------9------------->
Fitness=0.000214 Trial=1


<----------------------7---------------------><----------10------->
<---3---><------------8------------><--------2------->
<----------5---------><--------------9-------------><-----6----><4>
<--1-->
Fitness=0.000210 Trial=1
_
```

```
<---------10-------><4><--1--><-----6---->
<----------------------7---------------------><--------2------->
<--------------9------------->
Fitness=0.000209 Trial=1


<-----6----><---3---><--1-->
<----------------------7---------------------><------------8------------>
<--------------9-------------><4><----------10-------><--------2------->
<----------5---------->
Fitness=0.000202 Trial=1


<--1--><----------5---------><---3---><-----6---->
<----------------------7---------------------><------------8------------>
<--------------9-------------><----------10-------><--------2-------><4>
Fitness=0.000192 Trial=1

GENERATION =16

<------------8------------><----------------------7---------------------->
<--1--><----------5---------><----------10-------><4><--------2------->
<--------------9-------------><---3---><-----6---->
Fitness=0.000300 Trial=0
_
```


<----------7----------><---3--->
<----------9----------><---------10-------><-----6----><--1-->
<----------8----------><4><--------2-------><----------5--------->
Fitness=0.000279 Trial=0
<-----6----><----------9----------><4><--------2-------><--1-->
<----------8----------><---------10-------><---3--->
<----------7----------><----------5--------->
Fitness=0.000278 Trial=0
<--1--><4><----------8----------><-----6----><---------10------->
<---3---><----------5----------><----------9---------->
<--------2-------><----------7---------->
Fitness=0.000273 Trial=0
<----------5---------><----------7---------->
<---------10-------><--1--><--------2-------><---3---><4><-----6---->
<----------8----------><----------9---------->
Fitness=0.000237 Trial=0


<----------9----------><---------10------->
<----------7----------><----------8---------->
<----------5---------->
Fitness=0.000209 Trial=0
<----------8----------><--1--><---------10-------><---3--->
<----------7---------->
<----------9----------><----------5---------><-----6---->
<--------2-------><4>
Fitness=0.000202 Trial=0
<--------2-------><-----6----><---3---><---------10------><--1-->
<----------7----------><----------8---------->
<4><----------5---------><----------9---------->
Fitness=0.000198 Trial=0
<4><----------9---------->
<----------7----------><-----6----><--1-->
<---3---><----------5---------><----------8---------->
<--------2-------><---------10------->
Fitness=0.000187 Trial=0


<----------9---------->
<----------7----------><4><---3--->
<----------5---------><----------8----------><--1-->
<---------10-------><--------2-------><-----6---->
Fitness=0.000180 Trial=0
GENERATION =17
<----------8----------><----------7---------->
<--1--><----------5---------><---------10-------><4><--------2------->
<----------9----------><---3---><-----6---->
Fitness=0.000300 Trial=1
<----------7----------><---3--->
<----------9----------><---------10-------><-----6----><--1-->
<----------8----------><4><--------2-------><----------5--------->
Fitness=0.000279 Trial=1
<-----6----><----------9----------><4><--------2-------><--1-->
<----------8----------><---------10-------><---3--->
<----------7----------><----------5--------->
Fitness=0.000278 Trial=1



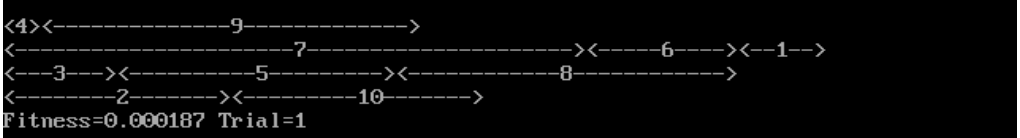
Fitness=0.000273 Trial=1

Fitness=0.000237 Trial=1

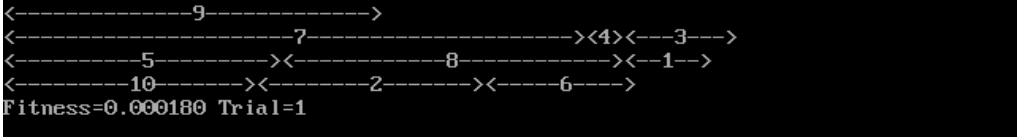
Fitness=0.000209 Trial=1

Fitness=0.000202 Trial=1

Fitness=0.000198 Trial=1

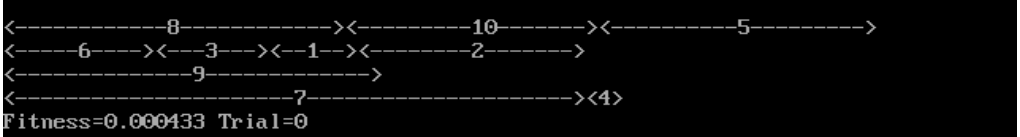


Fitness=0.000187 Trial=1

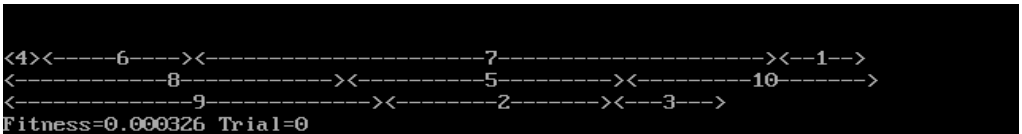


Fitness=0.000180 Trial=1

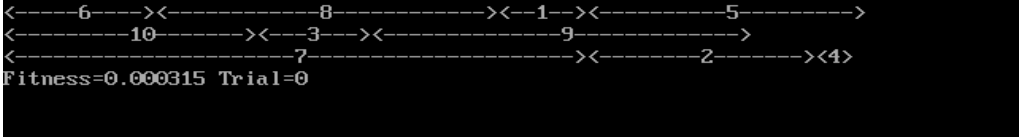
GENERATION =18



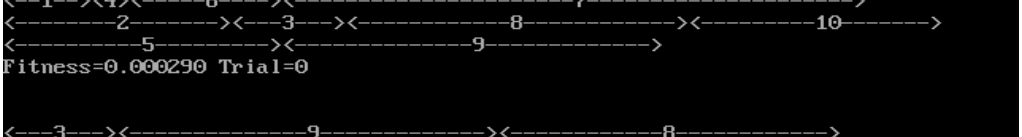
Fitness=0.000433 Trial=0



Fitness=0.000326 Trial=0

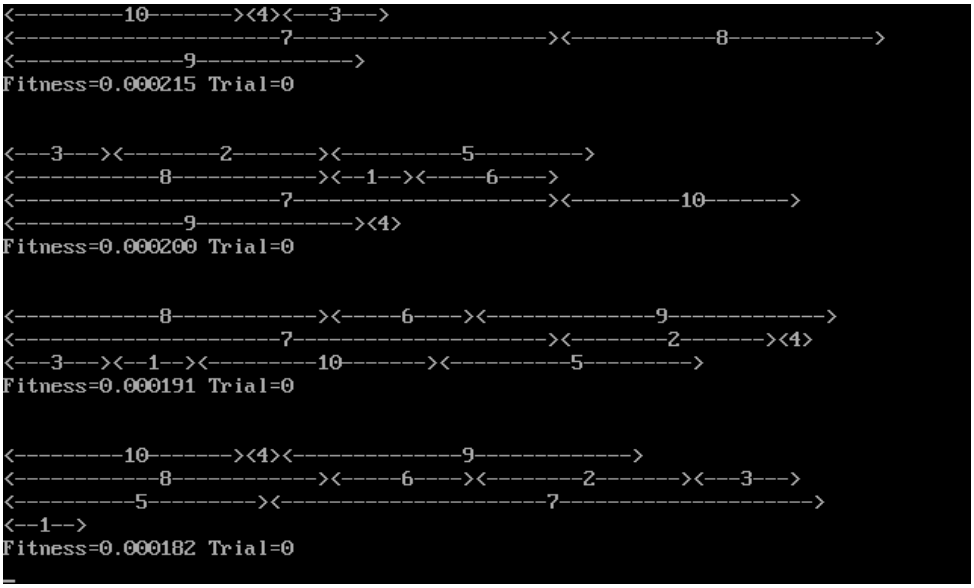


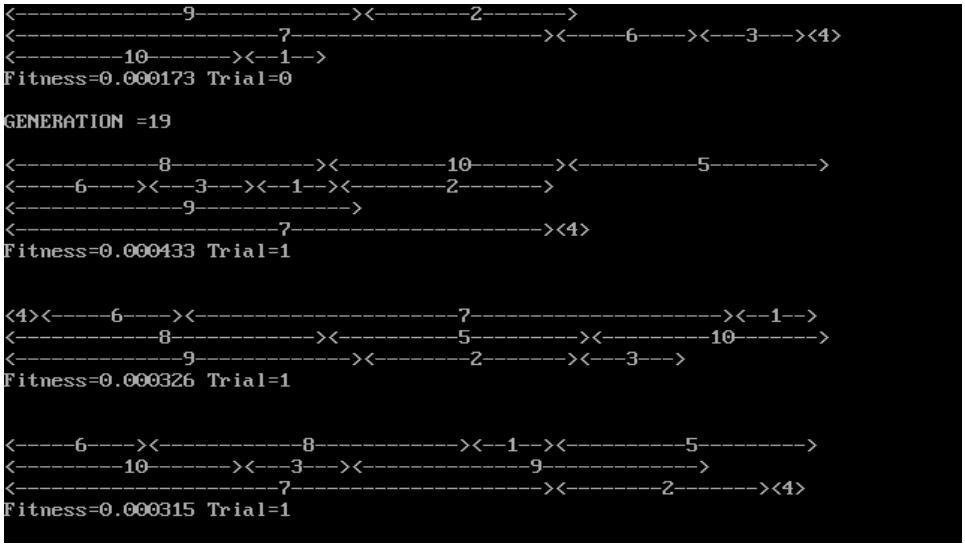
Fitness=0.000315 Trial=0

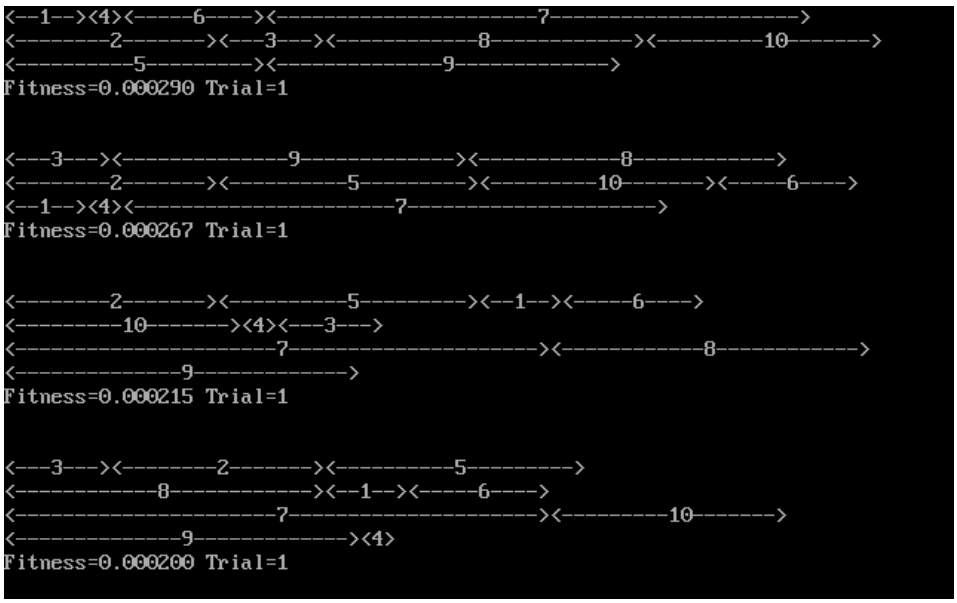


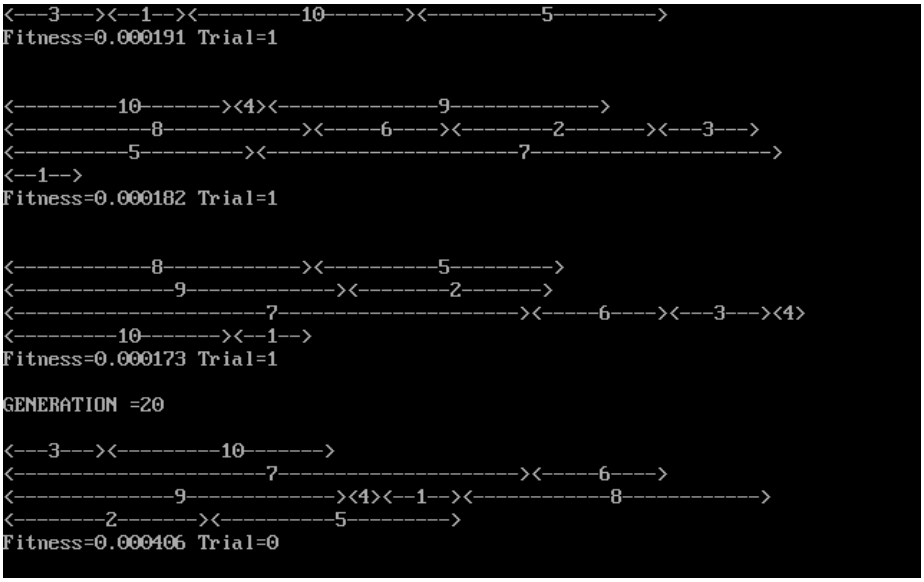
Fitness=0.000290 Trial=0

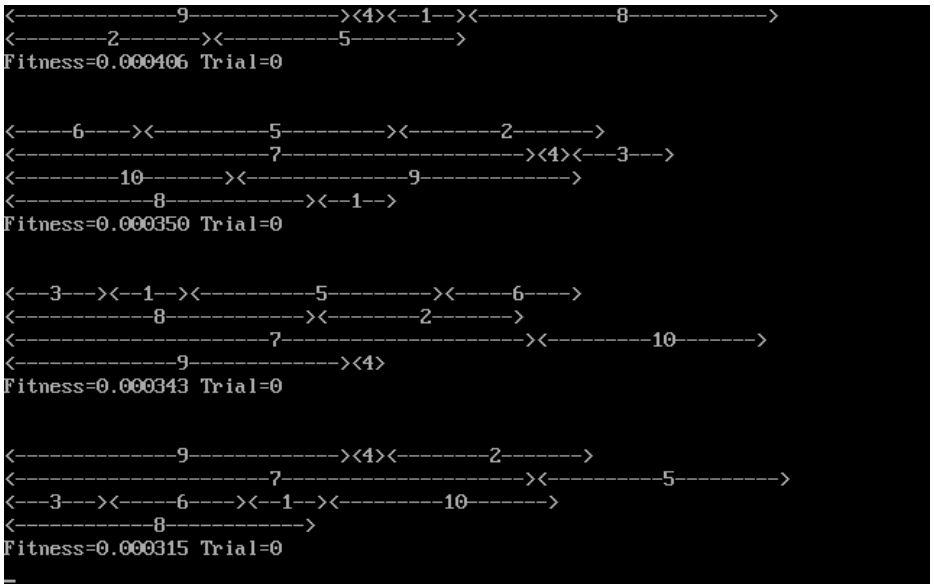
Fitness=0.000267 Trial=0

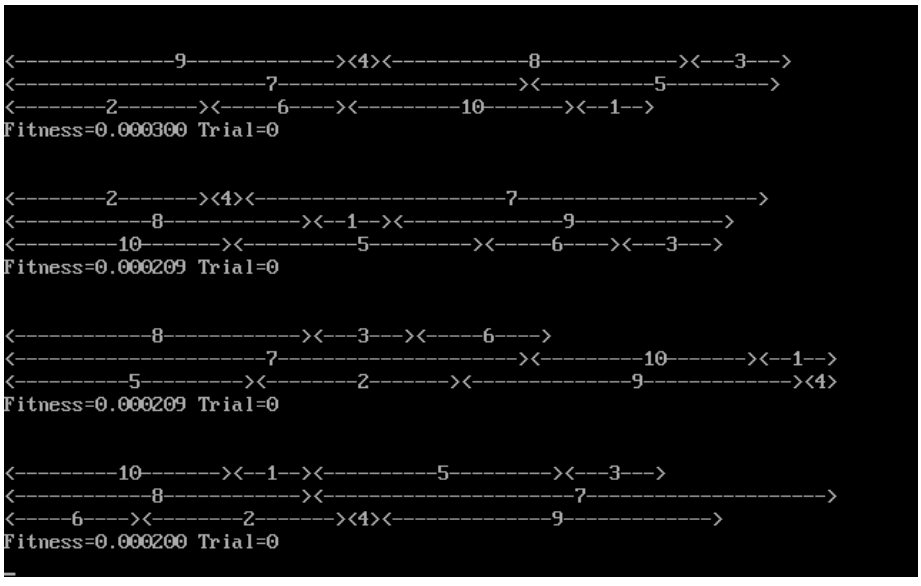

<---------10-------><4><---3--->
<----------------------7---------------------><------------8------------>
<--------------9-------------->
Fitness=0.000215 Trial=0
<---3---><--------2-------><----------5--------->
<------------8------------><--1--><-----6---->
<----------------------7---------------------><---------10------->
<--------------9-------------><4>
Fitness=0.000200 Trial=0
<------------8------------><-----6----><--------------9-------------->
<----------------------7---------------------><--------2-------><4>
<---3---><--1--><---------10------><----------5--------->
Fitness=0.000191 Trial=0
<---------10-------><4><--------------9-------------->
<------------8------------><-----6----><--------2-------><---3--->
<----------5---------><----------------------7---------------------->
<--1-->
Fitness=0.000182 Trial=0

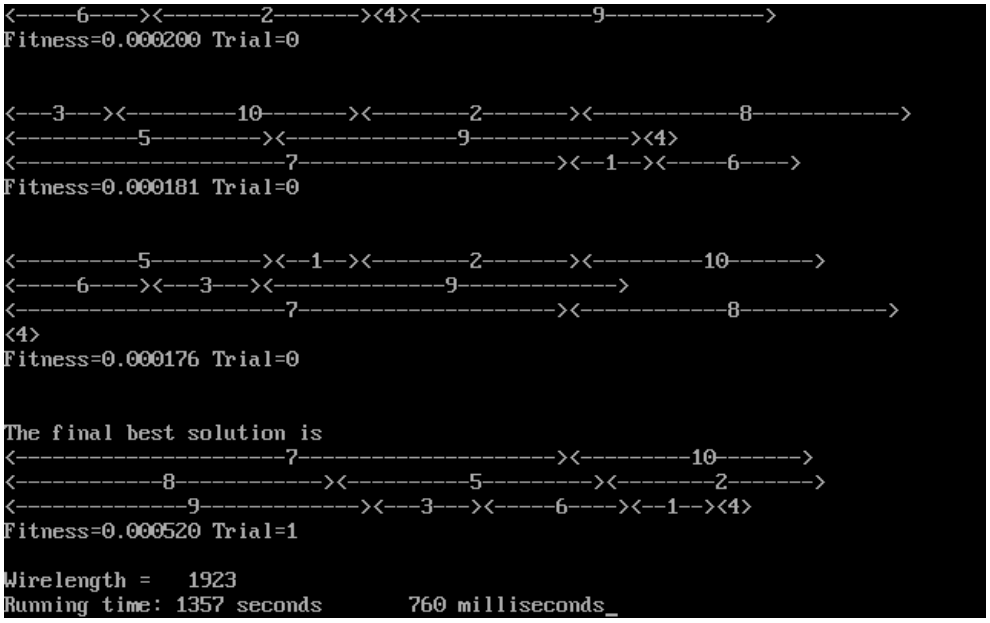

<--------------9-------------><--------2------->
<----------------------7---------------------><-----6----><---3---><4>
<---------10-------><--1-->
Fitness=0.000173 Trial=0
GENERATION =19
<------------8------------><---------10------><----------5--------->
<-----6----><---3---><--1--><--------2------->
<--------------9-------------->
<----------------------7---------------------><4>
Fitness=0.000433 Trial=1
<4><-----6----><----------------------7---------------------><--1-->
<------------8------------><----------5---------><---------10------->
<--------------9-------------><--------2-------><---3--->
Fitness=0.000326 Trial=1
<-----6----><------------8------------><--1--><----------5--------->
<---------10-------><---3---><--------------9-------------->
<----------------------7---------------------><--------2-------><4>
Fitness=0.000315 Trial=1


<--1--><4><-----6----><----------------------7---------------------->
<--------2-------><---3---><------------8------------><---------10------->
<----------5---------><--------------9-------------->
Fitness=0.000290 Trial=1
<---3---><--------------9-------------><------------8------------>
<--------2-------><----------5---------><---------10------><-----6---->
<--1--><4><----------------------7---------------------->
Fitness=0.000267 Trial=1
<--------2-------><----------5---------><--1--><-----6---->
<---------10-------><4><---3--->
<----------------------7---------------------><------------8------------>
<--------------9-------------->
Fitness=0.000215 Trial=1
<---3---><--------2-------><----------5--------->
<------------8------------><--1--><-----6---->
<----------------------7---------------------><---------10------->
<--------------9-------------><4>
Fitness=0.000200 Trial=1


<---3---><--1--><---------10-------><---------5--------->
Fitness=0.000191 Trial=1
<---------10-------><4><--------------9------------->
<------------8------------><-----6----><--------2-------><---3--->
<----------5----------><----------------------7---------------------->
<--1-->
Fitness=0.000182 Trial=1
<------------8------------><----------5--------->
<--------------9------------><--------2------->
<----------------------7---------------------><-----6----><---3---><4>
<---------10-------><--1-->
Fitness=0.000173 Trial=1
GENERATION =20
<---3---><---------10------->
<----------------------7---------------------><-----6---->
<--------------9------------><4><--1--><------------8------------>
<--------2------->
Fitness=0.000406 Trial=0


<----------5---------><---------2------->
Fitness=0.000406 Trial=0
<-----6----><----------5---------><--------2------->
<----------------------7---------------------><4><---3--->
<---------10-------><--------------9------------->
<------------8------------><--1-->
Fitness=0.000350 Trial=0
<---3---><--1--><----------5---------><-----6---->
<------------8------------><--------2------->
<----------------------7---------------------><---------10------->
<--------------9------------><4>
Fitness=0.000343 Trial=0
<--------------9------------><4><--------2------->
<----------------------7---------------------><----------5--------->
<---3---><-----6----><--1--><---------10------->
<------------8------------>
Fitness=0.000315 Trial=0


<--------------9------------><4><------------8------------><---3--->
<----------------------7---------------------><----------5--------->
<--------2-------><-----6----><---------10-------><--1-->
Fitness=0.000300 Trial=0
<--------2-------><4><----------------------7---------------------->
<------------8------------><--1--><--------------9------------->
<---------10-------><----------5---------><-----6----><---3--->
Fitness=0.000209 Trial=0
<------------8------------><---3---><-----6---->
<----------------------7---------------------><---------10-------><--1-->
<----------5---------><--------2-------><--------------9------------><4>
Fitness=0.000209 Trial=0
<---------10-------><--1--><----------5---------><---3--->
<------------8------------><----------------------7---------------------->
<-----6----><--------2-------><4><--------------9------------->
Fitness=0.000200 Trial=0



4.5 Summary

This chapter presented an algorithmic approach for standard cell placement problem combining features of evolutionary approach with the limited trials of the solutions that further breeds the next population based on the trial values, hence avoiding the algorithm getting trapped in local convergence. The experimental setup was discussed giving the details of the benchmark on which algorithm is simulated. The experimental results were discussed and the comparative results were shown in form of graphs. The snapshots of the implemented algorithm in Turbo C were also presented. The paper published for this chapter was:

Jobanpreet Kaur and Maninder Kaur, "Solving Standard Cell Placement Using Evolutionary Approach," *International Journal of Computer Applications and Information technology*, Vol. 2, No. 3, June, 2013.

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

5.1 Conclusion

The exponentially increase in the number of transistors on modern VLSI circuits causes the placement problem to be extremely complex thereby increasing the time to obtain acceptable solutions, especially for large circuits. Moreover, with advanced sub-micron technologies, the interconnect delay affects the circuit delays and becomes an important factor in determining the speed of a chip. These issues emphasize the importance of the placement problem. Therefore, effective and efficient placement techniques are necessary to deal with these new challenges. Evolutionary approaches have been widely used by the researchers for solving SCP problem. However, one disadvantage of these approaches is that they undergo premature convergence. Various hybrid approaches have been developed to counteract premature convergence and improve the running time of algorithm.

This work proposed an evolutionary approach developed for solving SCP problem that included the features of basic genetic algorithm and the limited trials of the solutions that further breeds the next population based on trial values to ensure a reasonable diversity in population. The methodology presented explained the basic working of the proposed algorithm and explained that how the proposed technique could explore the solution space effectively. The proposed algorithm incorporated Order Point crossover of the entire population leading to an entirely new population. The solution can remain in population as long as their trial value is less than equal to 3 i.e. at the maximum three chances would be given to the candidate to produce better offsprings. The values of the basic parameters were fixed and then the algorithm was simulated for 20 generations with population size 10. The experiments were conducted on sample benchmarks to evaluate the extent of improvement of results. The experimental results showed that the EASCP algorithm gave better results than basic GA in terms of wirelength. However, the running time of EASCP was more than that of basic GA but it was comparable for circuits having large number of nodes. The decrease in running time for circuits with larger nodes was due to the reason that the candidates with limited trial values were discarded and were not explored repeatedly.

5.2 Future Scope

The proposed EASCP algorithm can be further enhanced by not allowing the revisits of already generated solutions thereby, improving the running time of the algorithm. Candidate solutions are in general revisited a number of times, thus, lowering diversity and wasting valuable CPU time. Although the proposed algorithm discards the solutions after a number of trials, yet those solutions are sometimes regenerated after crossover, thus revisited and evaluated multiple times. Such reconsiderations of the same candidate solutions may degrade performance substantially by wasting precious computation time. Rejecting candidate solutions that are already generated would improve the situation and increase performance in terms of computation time.

The current approach is implemented in Turbo C. In order to use the same approach for very large circuits whose configurations cannot be supported by data types of C, the algorithm can be implemented using MATLAB software.

5.3 Summary

This chapter concludes the entire thesis work discussing the need of the proposed approach for standard cell placement, brief overview of the proposed algorithm and its advantages and disadvantages. This chapter also proposes the future work that could be done for the enhancement of the algorithm.

References

[1] K. Shahookar and P. Mazumder, "VLSI cell placement techniques," *ACM Computing Surveys (CSUR),* Vol. 23, No. 2, pp. 143-220, 1991.

[2] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, 1st ed, Boston, Addison-Wesley Longman Publishing Co., 1989.

[3] Myung-Chul Kim *et al.*, "SimPL: An effective placement algorithm," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst,* pp. 649-656, Nov., 2010.

[4] Sung Woo Hur and John Lillis, "Mongrel: hybrid techniques for standard cell placement," *Proc. IEEE/ACM Intl. Conf. on Comput.-Aided Des.*, IEEE Press, pp. 165-170, 2000.

[5] Dennis Huang and Andrew B. Kahng, "Partitioning-based standard-cell global placement with an exact objective," *Proc. ACM/IEEE Intl. Symp. on Physical Des.*, ACM, 1997.

[6] Naveed A. Sherwani, *Algorithms for VLSI Physical Design Automation*, 3rd ed., Hillsborough: Kluwer Academic Publishers, 1999.

[7] M.R. Garey and D.S. Johnson, *Computers and Intractability: A guide to the theory of NP-completeness*, W.H Freeman and co., San Francisco, 1979.

[8] Sadiq M. Sait and Habib Youssef, *VLSI physical design automation: theory and practice*, 1st ed., Vol. 6, Singapore: World Scientific Publishing Company, 1999.

[9] Eckart Zitzler, "Evolutionary algorithms for multiobjective optimization: Methods and applications," Ph.D. dissertation, Swiss Federal Inst. of Tech., Zurich, 1999.

[10] Jobanpreet Kaur and Maninder Kaur, "A Survey On Various Genetic Approaches for Standard Cell Placement," *Intl. J. Comp. Technology and Applications,* Vol. 4, No. 3, pp. 533-536, June, 2013.

[11] Zhen Yang, "Area/congestion-driven Placement for VLSI Circuit Layout," M.Sc. Thesis, Dept. The Faculty of Graduate Studies, University of Guelph, Ontario, 2003.

[12] Shawki Areibi and Zhen Yang, "Effective memetic algorithms for VLSI design= genetic algorithms+ local search+ multi-level clustering," *Evolutionary Computation*, Vol. 12, No. 3, pp. 327-353, Sept., 2004.

[13] Christos H. Papadimitriou, "Computational complexity," in *Encyclopedia of Computer Science, 4th* ed., Chichester, UK: John Wiley and Sons Ltd., 2003.

[14] Zhe-Wei Jiang *et al.*, "Challenges and solutions in modern VLSI placement," *In Intl. Symp. on VLSI Design, Automation and Test, 2007, VLSI-DAT 2007,* IEEE, 2007.

[15] Igor L. Markov *et al.*, "Progress and challenges in VLSI placement research," *Proc. IEEE/ACM Intl. Conf. on Comput.-Aided Des.*, IEEE, pp. 275-282, 2012.

[16] Thomas Back, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*, Oxford, UK: Oxford University Press, 1996.

[17] Thomas Weise. (2008, August 29). *Global Optimization Algorithms - Theory and Application*(2nd) [Online]. Available http://www.it-weise.de/.

[18] G. Jones, "Genetic and evolutionary algorithms," in *Encyclopedia of Computational Chemistry,* Chichester, UK: John Wiley and Sons Ltd., 1998.

[19] Thomas Bäck and Hans-Paul Schwefel, "An overview of evolutionary algorithms for parameter optimization," *Evolutionary computation*, Vol. 1, No. 1, pp. 1-23,1993.

[20] D. E. Goldberg, "Genetic and evolutionary algorithms in the real world," Dept. Gen. Engg., University of Illinois, Urbana, Rep. 99013, 1999.

[21] Y. Liu *et al*., "Scaling up fast evolutionary programming with cooperative coevolution," *Proc. 2001 Congress on Evolutionary Computation*, Vol. 2, IEEE, 2001.

[22] Thomas Back *et al*., "Evolutionary computation: Comments on the history and current state," *IEEE Transactions on Evolutionary computation,* Vol. 1, No. 1, pp. 3-17, 1997.

[23] P. Mazumder and E.M. Rudnick, *Genetic Algorithms for VLSI Design, Layout & Test Automation,* Toronto, Canada: Prentice Hall, 1999.

[24] Mahmood R Minhas, "An evolutionary algorithm for low power VLSI cell placement," *2003 IEEE 46th Midwest Symposium on Circuits and Systems*, Vol. 3., IEEE, 2003.

[25] Colin R. Reeves and Jonathan E. Rowe, *Genetic algorithms : principles and perspectives : a guide to GA theory*, Boston, Kluwer Academic Publishers, 2003.

[26] John R. Koza, "Genetic programming," in *Encyclopedia of Computer Science and Technology*, NewYork, NY: Marcel-Dekker, pp. 29-43, 1998.

[27] G. Syswerda, "Uniform Crossover in Genetic Algorithms," In *Proc. Third Intl. Conf. on Genetic Algorithms*, ed. J.D. Schaffer, San Mateo, CA: Morgan Kaufmann, pp. 2-8, 1989.

[28] J. H Holland, "*Adaptation in Natural and Artificial Systems*', MA, MIT Press Cambridge, 1992.

[29] J.H. Holland, "*Adaptation in natural and artificial systems*," Ann Arbor, University of Michigan Press, 1975.

[30] L. Davis (ed.), "*Handbook of Genetic Algorithms*," New York, Van Nostrand Reinhold, 1991.

[31] Melanie Mitchell, "*An introduction to genetic algorithm*", MA, MIT Press Cambridge, 1996.

[32] Guofang Nan *et al.* "An improved genetic algorithm for cell placement," In *Proc. Intl. Conf. on Intelligent Computing*, Springer Berlin Heidelberg, pp. 523-532, 2006.

[33] K. Shahookar and P. Mazumder, "Gasp: a genetic algorithm for standard cell placement," In *Proc. Conf. on European Design Automation*, IEEE Computer Society Press, pp. 660-664, 1990.

[34] Rini Mahajan *et al.*, "A genetic approach to standard cell placement using various genetic operators," *Intl. J. of Comp. App.*, Vol. 1, No.9, 2010.

[35] K. Shahookar and P. Mazumder, "A genetic approach to standard cell placement using metagenetic parameter optimization," *IEEE Trans. On CAD* , Vol. 9, pp. 500-511, May 1990.

[36] Aaquil Bunglowala *et al.*, "Optimization of Hybrid and Local Search Algorithms for Standard Cell Placement in VLSI Design," *Intl. Conf. Advances in Recent Technologies in Communication and Computing, 2009, ARTCom'09,* IEEE, 2009.

[37] Sadiq M. Sait and Mahmood R. Minhas, "GATS: A Novel Hybrid Algorithm for Cell Placement in VLSI Circuit Design," Dept. Comp. Engg., King Fahd University of Petroleum and Minerals, Saudi Arabia, 2002.

[38] V. Schnecke and Oliver Vornberger, "Hybrid genetic algorithms for constrained placement problems," *IEEE Transactions on Evolutionary Computation,* Vol. 1, No.4, pp. 266-277, 1997.

[39] Hartmut Pohlheim, (2005). *GEATbx Introduction evolutionary algorithms: Overview, methods and operators (version 3.8)* [Online]. Available http://www.geatbx.com.

[40] Rubicite Interactive, *Rubicite Interactive Inc.* [Online] 2012, http://www.rubicite.com/Tutorials/GeneticAlgorithms/CrossoverOperators/Order1CrossoverOperator.aspx

[41] S. N. Sivanandam and S.N. Deepa, *"Introduction to Genetic Algorithm,*" New York: Springer Berlin Heidelberg, 2008.

[42] Rubicite Interactive, *Rubicite Interactive Inc.* [Online] 2012, http://www.rubicite.com/Tutorials/GeneticAlgorithms/CrossoverOperators/CycleCrossoverOperator.aspx

[43] Rubicite Interactive, *Rubicite Interactive Inc.* [Online] 2012, http://www.rubicite.com/Tutorials/GeneticAlgorithms/CrossoverOperators/PMXCrossoverOperator.aspx

List of Publications

- Jobanpreet Kaur and Maninder Kaur, "A Survey On Various Genetic Approaches for Standard Cell Placement," *International Journal of Computer Technology and Applications,* Vol. 4, No. 3, pp. 533-536, June, 2013.
- Jobanpreet Kaur and Maninder Kaur, "Solving Standard Cell Placement Using Evolutionary Approach," *International Journal of Computer Applications and Information technology,* Vol. 2, No. 3, June, 2013.