
**ADDIS ABABA UNIVERSITY**

**SCHOOL OF GRADUATE STUDIES**

**ADDIS ABABA INSTITUTE OF TECHNOLOGY**

**DEPARTMENT OF ELECTRICAL and COMPUTER ENGINEERING**

# Design and Performance Evaluation of Hybrid Intelligent System-based Algorithm for Multiple DNA Sequence Alignment

**By**

**Addisu Galassa**

**A thesis submitted to the School of Graduate Studies of Addis Ababa University in partial fulfillment of the requirement for the Degree of Master of Science in Electrical and Computer Engineering (Computer)**

**Advisor**

**Kumudha Raimond (Dr.)**

**October, 2011**

**Addis Ababa, Ethiopia**

**ADDIS ABABA UNIVERSITY**

**SCHOOL OF GRADUATE STUDIES**

**ADDIS ABABA INSTITUTE OF TECHNOLOGY**

# Design and Performance Evaluation of Hybrid Intelligent System-based algorithm for Multiple DNA Sequences Alignment

**By Addisu Galassa Guddissa**

**Approved by Board of Examiners**

| | |
|---|---|
| _____ | _____ |
| **Chairman** | **Signature** |
| **Department of Electrical and Computer Engineering** | |
| _____ | _____ |
| **Advisor** | **Signature** |
| _____ | _____ |
| **External Examiner** | **Signature** |
| _____ | _____ |
| **Internal Examiner** | **Signature** |

# Acknowledgment

This research project would not have been possible without the support of many people. First of all I would like to express my gratitude to Doctor Kumudha Raimond for her invaluable contributions. She has been guiding and inspiring me throughout to carry on this project in spite of difficulties I encountered. I also want to thank her for her comprehensive lectures on Machine Learning which helped me solve the problem of the thesis. In short, the accomplishment of this work would have been difficult had it not been for her precious advice and comments.

My gratitude also goes to my class mate Ato Taye Tolu and my colleague Ato Wehib Abubeker for encouraging me to take this title and for helping me by providing materials related to it. Special thanks also to Miss Gelila Negash who has been there for me ideally and materially.

Lastly, I offer my regards and blessings to all of those who supported me in any respect toward the completion of the work.

# Table of Contents

# List of Tables

# List of Figures

# List of Acronyms

ASIC……..…….....…….Application Specific Integrated Circuit

BP……..…….……Base pair

BSM…….……..……..……Block of Sequence Move

C…….....…….……Cytosine

DNA……...……..……Deoxyribonucleic acid

EBI……..…….....……European Bioinformatics Institute

FPGA……..……..…..……Field Programmable Gate Arrays

G……..……..……Guanine

GA…….……..…….....……Genetic Algorithm

HDL……..…..…….……..……..Hardware Description Language

MAFFT……..…..…Multiple sequence Alignment using Fast Fourier Transform

MSA…….……...…..…..……Multiple sequence Alignment by Dynamic Programming

MultAl……....……...……Multiple Alignment with hierarchical clustering

OMA……..…….....……Optimal Multiple Sequence Alignment

PHGA……..……...……...…….parallel  hybrid Genetic Algorithm

PRNG…….……..……..……...…….Pseudo Random Number Generator

RNA……..……..……Ribonucleic acid

RTL……...……..…..…….Register Transfer Level

SA……..……..……....……Simulated Annealing

SAGA……...……..……..……..……Sequence Alignment by Genetic Algorithm

SSM…….……..……...……..……Single Sequence Move

T…….……..……Thymine

T-COFFEE……..……Tree-based Consistency Objective Function For alignment Evaluation

TS…….……..……Tabu Search

VHDL……...…….Very high speed integrated circuits Hardware Description Language

# Abstract

In this thesis work, a method to align multiple DNA sequences is designed. The proposed design is an intelligent system based hybrid algorithm of two optimization algorithms: Genetic Algorithm (GA) and Tabu Search (TS). GA phase is used to find new region of solution while TS explores regions of solution not explored by GA. The designed hybrid system is implemented using MATLAB. The TS part of the system is adapted so as to be processed by AccelDSP Synthesis tool and implemented in VHDL (Very high speed integrated circuits Hardware Description Language). The designed system is evaluated using benchmark methods CLUSTALW and MAFFT (Multiple sequences Alignment using Fast Fourier Transform). The system performs less than both the benchmarks. It performs less with percentage of matches differing at most by 8.6 from CLUSTALW for 8 sequences. It also performs less with percentage of matches differing at most by 4.25 from MAFFT for 16 sequences.



**Key Words:** Multiple DNA sequence alignment, Hybrid system, Genetic Algorithm, Tabu Search and FPGA based TS.

# Chapter 1

# Introduction

## 1.1 Background and Motivation

All living organisms, from bacteria to human beings, contain hereditary material called deoxyribonucleic acid (DNA), in each of their cells. DNA in each cell of every organism contains the entire genetic information of that organism. The information in DNA is stored as a code made up of four chemical bases: Adenine, Guanine, Cytosine and Thymine abbreviated as A, G, C and T respectively. The sequence or the order of these bases determines the information available for building and maintaining an organism [1].

Each base of DNA is attached to a sugar molecule and a phosphate molecule and forms nucleotides. The bases of nucleotides are also pair up with each other, A with T and C with G, to form units called base pairs. Nucleotides are arranged in two long strands that form a spiral called a double helix. The structure of the double helix is somewhat like a ladder, with the base pairs forming the ladder's rungs and the sugar and phosphate molecules forming the vertical sidepieces of the ladder. Figure 1.1 shows double helix DNA structure formed by base pairs attached to a sugar-phosphate backbone [1].

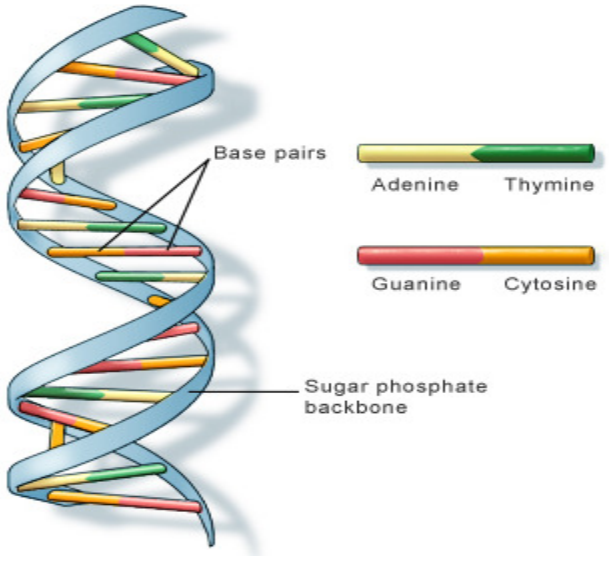



Figure 1.1 Double helix DNA structure [1]

The double stranded DNA, however, needs to be separated into single strands to result in a sequence of DNA. The sequences of DNAs can then be aligned with each other.

Biological sequence alignment is simply arranging two or more biological sequences (nucleotides or amino acids) so as to maximize (highlight) the similarities between them [2, 3]. Fig 1.2 shows a fragment of a multiple alignment of five hypothetical DNA sequences.

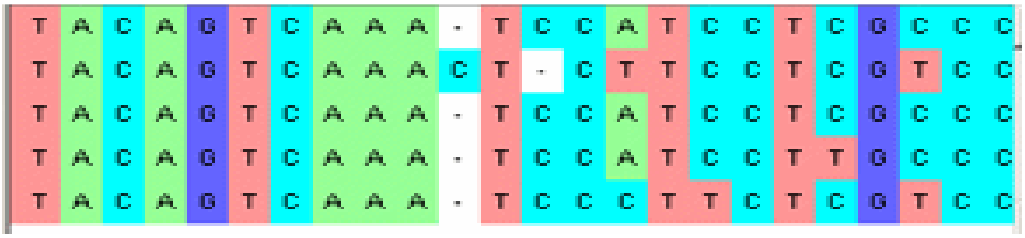



Figure 1.2 A fragment of an alignment of five DNA sequences

In the alignment of figure 1.2, the gap characters '-' are inserted into sequences to find similar regions of sequences (marked in same color).

Sequence alignment is classified into two based on how many sequences are aligned. When only two sequences are aligned, it is called pair wise sequence alignment. Otherwise, it is called multiple sequence alignment [2]. Sequence alignment can also be grouped in two types, global and local alignment. In global alignment, attempts are made to detect the best alignment of the entire sequences. In local alignment, the best alignment is constructed for segments of sequences with the highest density of matches, while the rest of the sequences are ignored [4].

Alignment of the biological sequences (DNA, RNA and Protein) plays an important role in Bioinformatics and Molecular Biology [5]. It is used to study molecular evolution [2]. Aligning DNA sequences enable the construction of evolutionary tree. Moreover, sequence alignment is also used to predict the function/structure of unknown sequence by aligning with other sequence whose function/structure is already known [3]. In this research global alignment of multiple DNA sequence is studied. Thus, from this section onwards, the word alignment will refer to global alignment.

## 1.2 Problem Statement

A number of surveys on algorithms for the multiple sequence alignment have been carried out. Consequently multiple sequence alignment has been shown to be an optimization problem which exhibits a great time and space complexity [30]. In general, optimization algorithms for multiple sequence alignment is classified into three categories [4]; progressive, exact (dynamic) and iterative.

The majority of multiple sequence alignment heuristics is now carried out using the progressive approach [2]. In this approach, a set of extremely similar sequences are first aligned, then subsequent sequences, which are not as similar, are progressively added to the original query set. The well known multiple sequence alignment named CLUSTALW, MultAl (Multiple Alignment with hierarchical clustering) and T-COFFEE (Tree-based Consistency Objective Function For alignment Evaluation) are among the widely used methods of this approach. This approach has the advantages of speed and simplicity. However, the greedy nature of the approach causes local minimum problem [2].

Another approach is to prune the search space of the Dynamic Programming algorithm for simultaneously aligning multiple sequences. Dynamic programming approach involves the alignment of any two sequences first and then any gaps in between becomes later filled in by assessing possible matches with other sequences. MSA (Multiple Sequences Alignment by dynamic programming) and OMA (Optimal Multiple Sequence Alignment) are examples of methods that use Dynamic Programming approach to align multiple sequences. Algorithms of this approach often find better quality solutions than those of the progressive approach [2, 23]. However, aligning sequences using these methods faces drawbacks of complexity, running time and memory requirement, so they can only be applied to problems with a limited number of sequences (about 10) [2].

The iteration-based approach is also applied to the multiple sequence alignment. Iterative methods align sequences based on a selected alignment scoring method and then realign sequence subsets. The realigned subsets are then themselves aligned to produce the next iteration's multiple sequence alignment. This approach includes iterative refinement algorithms (e.g., simulated annealing [5], Genetic Algorithms (GAs) etc. GAs differs from the others in that

they search for the solution from a population of potential solutions [2]. When used alone, however, GA has the drawbacks of relatively poor quality and slow speed [2, 23]. Furthermore, if the initial population of the GA is created completely random, the search for the solution can be trapped in local minima [3]. Nevertheless, hybridizing GA with other methods can evade being trapped in local minima [3] and produce impressing result [12].

Algorithms falling in one of these approaches are used independently to align multiple sequences. However, algorithms categorized in same or different approaches are also used. For example, a hybrid system of GA and SA (Simulated Annealing) both falling in iterative approach is used for alignment of protein sequences [4]. The GA phase used to find new region of solution while SA was used as an alignment improver for any near optimal solution produced by GA. In addition, GA was also hybridized with four heuristics such as local search, elitist strategy, modified Hopfield neural network and neighbor improvement for example for classical travelling salesman problem [8]. All the possible GA hybrids were investigated including hybrid systems of all the methods. Experimental results of the work showed that the results obtained from the hybrid GA with Neighbor Improvement outperformed those results obtained from other hybrid systems in terms of quality of the results obtained [8].

In this work, an intelligent system based technique which falls in class of iterative approach discussed formerly in this section is selected for DNA sequence alignment. The method is then hybridized with another optimization technique for further refinement of the alignment.

Iterative algorithms have been implemented in different high level languages. Some works, however, has been implemented in hardware. In this regard, reprogrammable logic devices, FPGAs, have been widely used for implementations of computational intensive bioinformatics algorithms and resulted in a significant improvement in speed [7]. Besides, increasing the computational speed of the algorithm, reconfigurability of FPGA makes it suitable for design verifications before final implementation in Application Specific Integrated Circuits, ASICs.

## 1.3 Objective

### 1.3.1 General Objective

- The main objective of this thesis is to design and evaluate the performance of a hybrid intelligent system for global multiple DNA sequence alignment.

### 1.3.2 Specific Objectives

- To study the existing DNA sequence alignment, select intelligent techniques to design an hybrid intelligent system.
- To implement the designed system in MATLAB and VHDL (using FPGA simulator software for the latter) and compare its performance with the benchmarks with regard to alignment quality and number of sequences to be aligned.
- To compare the software (MATLAB implementation) and the FPGA implementation via simulation.

## 1.4 Methodology

To attain the goal of this research, the work proceeded in three major phases: investigation of existing techniques; system design; and implementation and performance evaluation.

### 1.4.1 Investigation of existing techniques

The research work started with going through books, journals, previous research works on the areas of biological sequence alignment to have a clear understanding of the area. While making thorough review of the literatures, intelligent system based techniques and any other techniques used for sequence alignment have been studied. The detailed discussion of the literatures reviewed is given in chapter two.

### 1.4.2 System design

During this phase an intelligent system has been selected based on the statement of the problem and investigations of the sequence alignment methods. In addition, another optimization

algorithm has also been selected for hybridization to enhance the performance of the designed system. Hence, a hybrid system has been designed for global multiple DNA sequence alignment. The model of the system is discussed in detail in chapter four.

### 1.4.3 Implementation and performance evaluation

The designed algorithm has been coded in MATLAB and verified. The MATLAB code is then modified so as to be synthesized by AccelDSP synthesis tool. That is, the MATLAB floating point design has been transformed to a model in VHDL that can be implemented on Xilinx FPGA. Then the VHDL code has been simulated using FPGA simulator (Xilinx's ISE design Suite). Finally, the performance of the algorithm has been compared with benchmark technique from European Bioinformatics Institute (EBI).

## 1.5 Scope of the study

The design system is composed of GA and TS both of which fall in class of iterative approach. DNA sequences to test the designed system are taken from the database of EBI. In this work, alignment of multiple DNA sequences over the entire length of the sequence is taken care of. No attempt is made to work on local alignment of DNA sequences. The hybrid system is implemented using MATLAB. However, an attempt to implement both the GA and TS phase in FPGA is not achieved. Synthesizing the MATLAB design and transforming it to hardware module by synthesis tool called AccelDSP is not successful for the GA phase. Consequently, only the MATLAB design for TS phase is adapted to the requirements of the synthesis tool and transformed to hardware module that can be implemented in Xilinx FPGA.

## 1.6 Thesis outline

In the subsequent chapter, literatures related to the work of this thesis are discussed. The theoretical background of optimization algorithms used in this research is then described in chapter 3. In fourth chapter of the report, the model of the designed system and its implementation detail are presented. The results of the work along with discussions are given in chapter five. Finally, conclusions and recommendations are given.

# Chapter 2

# Literature Review

## 2.1 Introduction

Nowadays, numbers of researches are being carried out in areas of bioinformatics. Most of the researches being conducted on sequence alignment of biological sequences, namely, Protein, RNA and DNA. Some of the studies use GAs or its hybrid form for alignment of biological sequences. Studies based on other techniques such as Dynamic programming, progressive approaches are also remarkable. Besides the software implementation, FPGAs were being used to implement and significantly accelerate algorithms.

## 2.2 Literatures on Software methods

Cédric Notredame, et al. [3] on their paper described an approach to align multiple protein sequences using GAs and an associated software package called SAGA (Sequence Alignment by GA). The method involves evolving a population of alignments in a quasi evolutionary manner and gradually improving the fitness of the population. The performance of the method was compared using test cases, 12 of them from Pascarella Structural alignment database and one from Chymotrypsin Sequences. Nine small alignments (4-8 sequences and 60-280 residues long) among the test cases were treated as the first group of test case whereas the remaining four test cases of larger alignments (9, 12, 15 and 32 sequences) are classified as the second group. The second group of test case cannot be handled by MSA unlike the first group. The result of SAGA was analyzed by comparing its score with that of MSA and CLUSTALW. SAGA and MSA were compared with regard to its ability to optimize an objective function that MSA attempts to optimize for the first group of test cases. For all the test cases, SAGA was able to produce a score at least as good as that produced by MSA. The ability of SAGA to perform multiple sequence alignment on sequences that couldn't be aligned by MSA was also analyzed using a second group of four test cases by comparing its results with those given by CLUSTALW. For these test cases SAGA performed better than CLUSTALW.

A research group of four researchers Hung Dinh NGUYEN, et al. [2] presented a parallel hybrid Genetic Algorithm (PHGA) based method for multiple protein sequence alignment. The method is based on GA where only one offspring is created by either crossover or mutation. It also involves local search heuristics and parallelism to exploit the benefit of multiprocessor system. That is, a number of individuals (sub-populations) are acted up on by parallel processors and after a predefined number of generations called migration interval, best individuals are exchanged between sub-populations. The sum-of-pairs-score objective function is used and optimized in this method as well. To validate the method, a total of 85 alignment cases were taken from BALiBASE library. Sequences from reference 1 of the BALiBASE library (82 sequences) are used to compare with OMA and MSA methods as these sequences are small enough for both methods to handle. It has been shown that PHGA wins 29, losses 3 and draws 50 cases as compared to the OMA method. And as compared to MSA methods, the number of wins, losses and equals of the PHGA are 30, 10 and 42 respectively]. The remaining 3 larger sequences from reference 2 were used to test how aptly the algorithm scales with the size of the sequences It has been verified that the method can scale quite well with the problem size

Four researchers Mohd. Faizal Omar, et al. [5] also solved problems of multiple sequence alignment using hybrid system of GA and Simulated Annealing. The GA phase will find new region of solution while Simulated Annealing can be considered as an alignment improver for any near optimal solution produced by GAs. Simulated Annealing also helps to prevent local minima problem compared to the Dynamic programming. Even if the result produced from the pre-alignment and GA phase was significant, the effect of simulated annealing was very less.

Warattapop Chainate, et al. [8] conducted a research on improving the performance of GA through classical travelling salesman problem. Attempting to use the best configuration of the algorithm itself and adding in other heuristics as sub process of the algorithm are proposed ways of improving the performance. Specifically, the paper aimed at hybridizing GA with four heuristics (Local Search, Elitist Strategy, Modified Hopfield Neural Network and Neighbor improvement). Experimental result showed that hybridizing the GA with any of the four heuristics outperformed the non-hybrid GA in terms of quality of the result.

Inaddition, TS has also been used for combinatorial optimization problems. In research conducted by riq Riaz, et al. [9] TS has been used for multiple sequence alignment. The adaptive memory features of tabu search, which is simply concept tracking forbidden and candidate list, was used to align multiple sequences. The adaptive memory helps the search process to explore solution space economically and effectively and hence, to avoid local optima. The algorithm was implemented in SUN Java language. Datasets from BAliBASE benchmarking database was used to test the algorithm on a 1.4GHz Pentium III computer. It was shown that for datasets comprising orphan sequences, divergent families and long internal insertions, tabu search generates better alignment as compared to other methods: SAGA, PRRP, CLUSTALW and ML_PIMA.

Apart from independent implementation of TS, it was also used with GA forming combinatorial heuristic for solving optimization problem though not particularly for multiple sequence alignment. R.Thamilselvan, et al. [10] used TS with GA to minimize travelling time and cost for Travelling Salesman Problem. Both algorithms are tested independently and also in hybrid form for meeting the objective of the TSP. It was concluded that the hybrid system of TS and GA minimizes the travelling time and cost of the TSP than both of the algorithms.

## 2.3 Literatures on Hardware Implementation

With regard to implementation, FPGAs were being used to implement and significantly accelerate algorithms. Xunying Zhang, et al. [11] designed hardware based architecture to perform GA though it is not specifically for Multiple Sequence Alignment. The GA Kernel designed is divided into eight main functional components and memory components. The eight functional components in the kernel are Centric controller, Random Number Generator, Initialization, Parent selector, Genetic Operation, Individual selector, Bus controller and Population replacement. In the research, the GA kernel using chromosome length of 16 bits, fitness length of 16 bits and a population size of 128 is synthesized using Verilog HDL (Hardware Description Language) on Xilinx Spartan 2E XC2S300EFG456. It was noted on the paper that analyzing the system carefully, the architecture can be modified using pipeline in order to improve the performance of the system.

Tu Lei, et al. [12] also presented FPGA based architecture for GA. The GA model in this paper is implemented in VHDL using a Xilinx XC2S100 FPGA and the hardware is divided in to six logic modules (control state machine, storage modules, selection module, crossover module, mutation module and random data generation module). The hardware model of the GA was compared with the software based model on personal computer or work station. It was shown that it took 0.15seconds to finish the computing operation of 1000 generations on the FPGA based hardware with a clock frequency of 20MHz, which is nearly 1000 times faster than the implementation on WS. Moreover, it was stated on the paper that the GA hardware model can be applied to the studies and implementation of evolvable hardware based on dynamic reconfiguration technology.

A general purpose VHDL based intended for hardware implementation was presented by Scott, et al. [13] It was stated on the paper that it is possible to implement the GA model on reprogrammable FPGAs, exploiting the speed up of the hardware while retaining the flexibility of a software implementation. The GA engine designed was useful in many applications where software based GA implementations are too slow.

Stefan Dydel, et al. [7] also presented hardware implementation specifically for bioinformatic algorithm. In this paper, Smith-Waterman algorithm which performs a pairwise local alignment of protein sequences using FPGAs was implemented. It was shown that the FPGA based implementation of Smith-Waterman algorithm can accelerate sequence alignment on a Pentium desktop computer by two orders of magnitude.

The research work of Tim Oliver, et al. [26] is again FPGA implementation of multiple sequence alignment. In the paper, a method based on progressive approach was designed and implemented on reconfigurable hardware platforms, FPGAs. It was indicated in the report that speed up of 50 is gained by the FPGA implementation.

## 2.4 Summary

As illustrated in statement of the problem section of chapter one, the greedy nature of the progressive approach brings about local minima problem and the exact (dynamic programming) methods are limited to aligning about 10 sequences. With regard to the iterative approach, though it can be trapped in local minima, the problem can be triumphed over by hybridizing the algorithm with other iterative algorithm. Therefore, algorithms from iterative approaches are given preference. In particular, GA is selected for the fact that it has got the ability to find new regions of solutions. Moreover, it can produce an impressing result when combined with other technique as pointed out. It was also mentioned in this chapter that for hybrid system of GA and SA used for multiple sequence alignment, the GA phase showed significant performance while the effect of SA was very less. However, for hybrid system of GA and TS used for travelling salesman problem, not only the GA showed significant performance but also the TS did [10]. Hence, TS is chosen for further refinement of the optimal alignment produced by GA. The nature of the two algorithms favors the order where GA is followed by TS. GA begins with the creation of a number of initial alignments and ends with a number of alignments among which the best can be taken as optimal solution. TS, on the other hand, begins the optimization problem with single alignment and also produces only one alignment. Thus, a hybrid system in which GA is followed by TS is proposed.

Molecular biologists frequently compute task of multiple sequences alignment for roles of task presented in chapter one. Though aligning few sequences requires minutes, need to align plenty (hundreds, thousands, millions or more) of sequences will arise. Due to rapid growth of biological sequence databases, biologists have to compute multiple sequence alignment in far shorter time [26]. Therefore, an effort has been made in this work to come up with the FPGA implementation of the designed system to enhance the speed of multiple sequence alignment.

# Chapter 3

# Optimization Algorithms

## 3.1 Introduction

The search for an optimal state is one of the most fundamental principles of this world. Optimization is going through a period of growth, driven largely by new applications in many areas using different algorithms. These algorithms, also called optimization algorithms, are those methods which are used to find optimal solutions for given problems [14, 22]. The goal of global optimization is to find the best possible elements *x'* from a set *X* according to a set of criteria F = $\{f_1, f_2 \ldots f_n\}$.These criteria are expressed as mathematical functions, the so-called objective functions [14, 22]. Some of the most popular algorithms in this respect include: GAs, Genetic Programming, Learning Classifier Systems, Evolution Strategy, Differential Evolution, Particle Swarm Optimization, Ant Colony Optimization, Simulated Annealing, Extremal Optimization, TS, and Random Optimization. Attributed to the nature and objective of this work, the theoretical basics of GA and TS are presented in this section.

## 3.2 Genetic Algorithm

GA is adaptive heuristic search algorithm inspired from natural evolution. The basic concept of GA is designed to simulate processes in natural system necessary for evolution, specifically those that follow the principles first laid down by Charles Darwin of survival of the fittest. As such they represent an intelligent exploitation of a random search within a defined search space to solve a problem [14, 15].

GA is optimization technique based on selection and recombination of promising solutions. The collection of candidate solutions is called populations of GA whereas candidate solutions are sometimes named as individuals, chromosomes, etc. Each individual is an encoded representation of variables of the problems at hand. Each component (variable) in an individual is termed as gene. Sometimes the components (genes) are independent of one another and sometimes they are correlated.

As they are a particular class of evolutionary algorithms, the operations common to all GAs are inheritance, mutation, selection, and crossover (also called recombination).

### 3.2.1 Crossover

It is exchange of genetic material (substrings) denoting rules, structural components, and features of a machine learning, search, or optimization problem. In short, crossover selects genes from parent chromosomes and creates new offsprings.

### 3.2.2 Selection

Selection refers to the application of the fitness criterion to choose which individuals from a population will go on to reproduce.

### 3.2.3 Replication

Replication is the propagation of individuals from one generation to the next. It is simply the formation of next generation.

### 3.2.4 Mutation

The genetic operation called mutation is modification of chromosomes for single individuals. Hence, mutation changes randomly the new offspring.

Given a clearly defined problem to be solved and a binary string representation for candidate solutions, a basic GA can be represented as in 3.1.

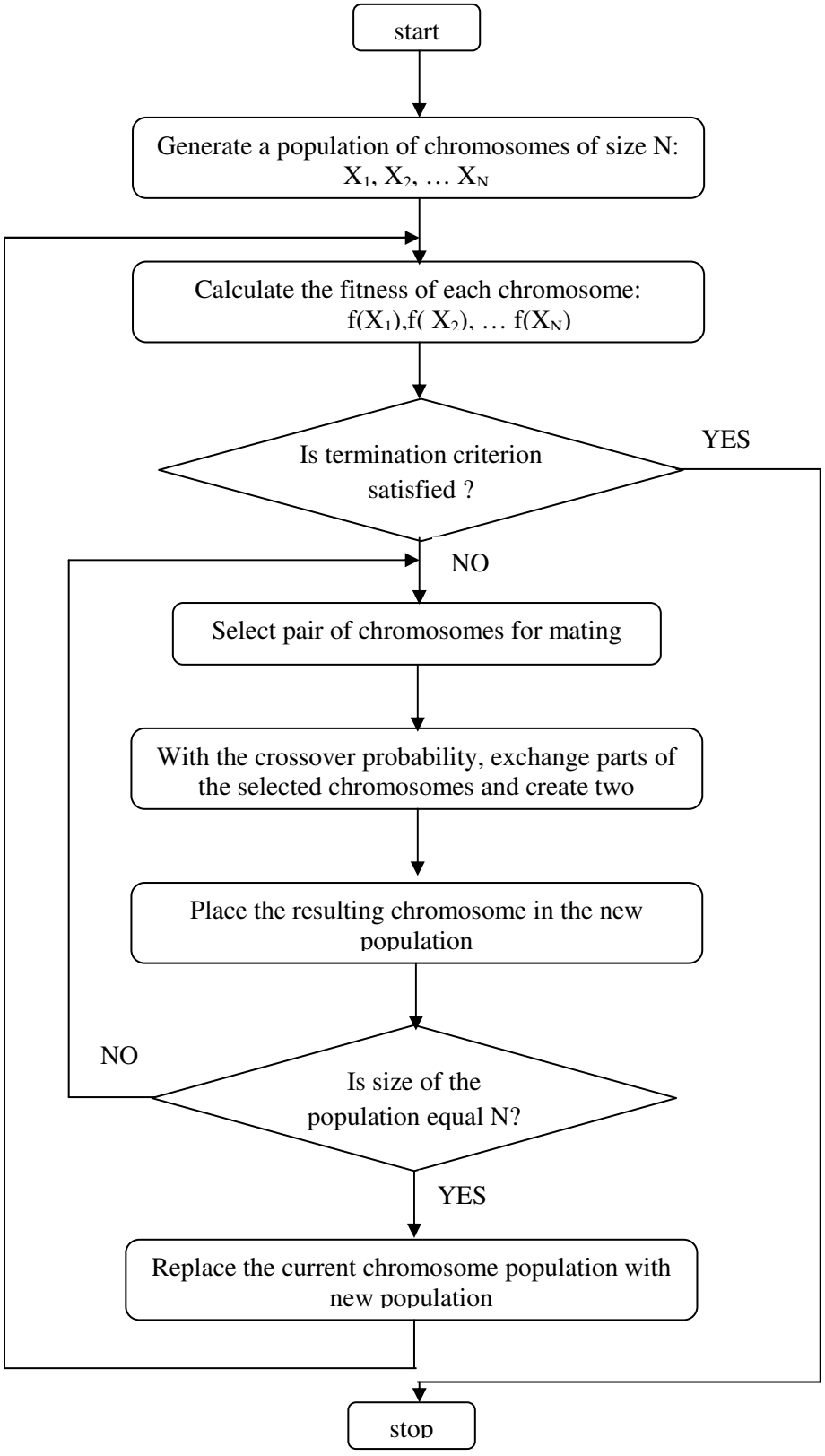



Figure 3.1 A basic genetic algorithm [16]

The more elaborate fundamental steps incorporated in GAs include the following [15, 16]:

**Step 1**: Represent the problem variable domain as a chromosome of a fixed length; choose the size of a chromosome population N, the crossover probability pc and the mutation probability pm.

**Step 2**: Define a fitness function to measure the performance, or fitness, of an individual chromosome in the problem domain. The fitness function establishes the basis for selecting chromosomes that will be mated during reproduction.

**Step 3**: Randomly generate an initial population of chromosomes of size *N:* $x_1, x_2, \ldots, x_N$

**Step 4**: Calculate the fitness of each individual chromosome: $f(x_1), f(x_2), \ldots, f(x_N)$

**Step 5**: Select a pair of chromosomes for mating from the current population. Parent chromosomes are selected with a probability related to their fitness. Highly fit chromosomes have a higher probability of being selected for mating than less fit chromosomes.

**Step 6**: Create a pair of offspring chromosomes by applying the genetic operators – crossover and mutation.

**Step 7**: Place the created offspring chromosomes in the new population.

**Step 8:** Repeat Step 5 until the size of the new chromosome population becomes equal to the size of the initial population, N.

**Step 9**: Replace the initial (parent) chromosome population with the new (offspring) population.

**Step 10**: Go to Step 4, and repeat the process until the termination criterion is satisfied.

As can be observed from the flow chart of Figure 3.1, after the initial population is created the GAs repeats the following steps until the solution is found:

- Evaluate current population
- Select candidates
- Create a new population

Generally, GA is best used when the objective function is discontinuous, highly nonlinear, and stochastic and has unreliable or undefined derivatives.

## 3.3 Tabu Search

TS, first introduced by Glover, is a heuristic procedure to find good solutions to combinatorial optimization problems [17, 18]. The meta-heuristic approach called TS is dramatically changing our ability to solve problems of practical significance. It has also been used to create hybrid procedures with other heuristic and algorithmic methods, to provide improved solutions to problems [17]. Current applications of TS span the realms of resource planning, telecommunications, VLSI design, financial analysis, scheduling, space planning, energy distribution, molecular engineering, logistics, pattern classification, flexible manufacturing, waste management, mineral exploration, biomedical analysis, environmental conservation and scores of others [17].

## 3.4 Basics of Tabu Search

TS begins in the same way as ordinary local or neighborhood search, proceeding iteratively from one solution x to another solution x' in the neighborhood of x. The search progresses iteratively until a chosen termination criterion is satisfied [17]. Each solution x $\epsilon$ X has an associated neighborhood $N(x)$ C $X$ , and each solution $x' \in N(x)$ is reached from x by an operation called a move [17].

TS permits moves that enhance the current objective function value [17]. Besides the enhancement or deterioration of the objective function, the history of the states encountered during the search is used to select the moves. During the search process, TS uses short term memory structures that describe the visited solutions. Once a potential solution has been determined, it is marked so that the alignment process does not visit that possibility repeatedly. Thus, making use of the history of the states, the search process effectively finds a better solution. The short term memory structure that records list of already visited moves and, hence, marked tabu is called Tabu List. Sometimes, a move in the Tabu list can be applied if it passes a set criterion called aspiration criteria. That is if the move has got the greatest objective function

value of all the values of previous moves. It is believed that overriding tabu criteria in this way might lead to a new path toward better optimal solution [9].

TS uses attributive memory for guiding purposes. This type of memory records information about solution attributes that change in moving from one solution to another. The most common attributive memory approach is called recency-based memory structure [17]. Recency-based memory structure keeps track of solutions properties that have changed during the recent past. To utilize this memory, solutions already visited have attributes called Tabu Tenture, which is the number of iterations for which the corresponding move remains in the Tabu List. Another memory structure called frequency-based memory is also used in TS implementations. This memory structure uses ratios about the number of iterations a solution property has changed.

The model of standard TS is given in figure 3.2.

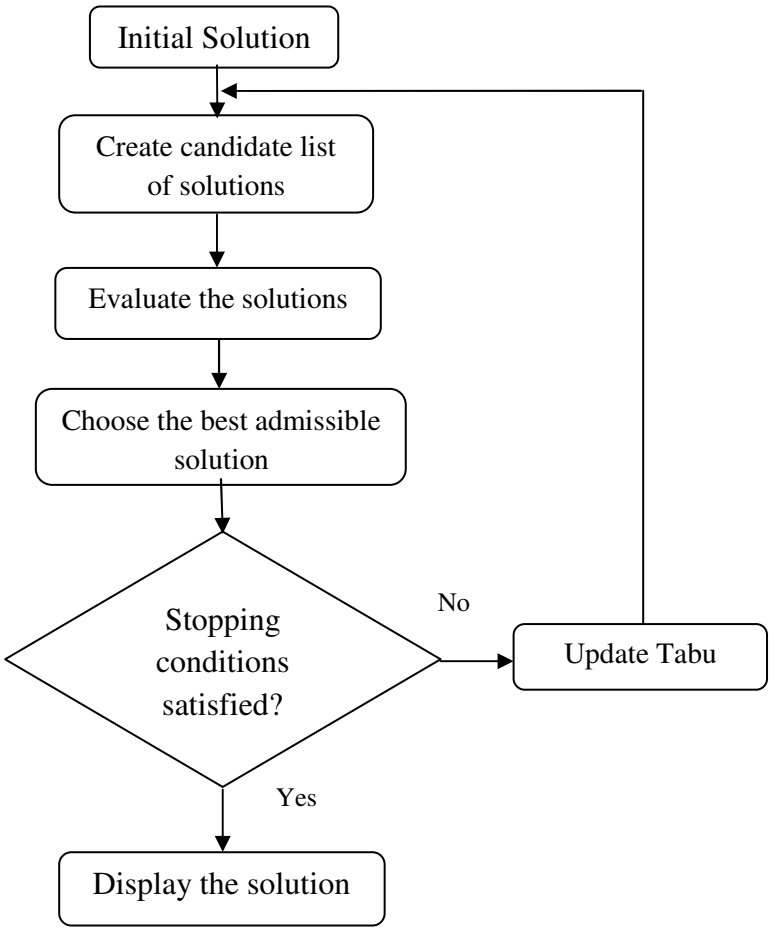



Figure 3.2 Standard Tabu Search model [19]

As shown in figure 3.2, the search process passes through a series of seven steps to bring about optimal solution. These steps are discussed in the following section.

**Step 1:** Initialize the TS parameters; set Tabu Tenture which is the number of iterations after which a solution marked tabu can be reapplied. Define aspiration Criteria, a criteria which can override tabu criteria. Set the initial solution

**Step 2:** Explores the neighborhood of the solution. That is, different solutions are generated forming a set of candidate solutions.

**Step 3:** Evaluate the fitness of the solutions in the candidate list.

**Step 4:** Choose the best solutions among the candidate list and make the solution the current solution if it is not in the Tabu List. If it is in the Tabu List, the solution is discarded and another solution not available in the Tabu List and having better fitness than the remaining is chosen.

**Step 5:** Check stopping condition. Usually unimproved solution for specified number of iterations is used as stopping condition. If the criterion is not met, go to step 6. Otherwise end the algorithm.

**Step 6:** Update the Tabu List and Tabu Tenture. If a move is applied in step 4, the applied move is added to the Tabu List and the same move is not allowed for specified number of iterations called Tabu Tenture [9]. The Tabu Tenture is decremented every iteration and when it reaches zero, the move is released from the forbidden list. Upon completion of this step, the algorithm returns back to step 2.

## 3.5 Summary

Optimization Algorithms in general maximizes or minimizes the objective function moving from one variable set to another based on some determinant sequence of steps. The hybrid systems of two optimization algorithms GA and TS, then optimizes the objective function more effectively by virtue of combining the features of the algorithms. The possibility of exploring a region of a solution not explored by the GA phase obviously exists. In such cases TS is believed to enhance the performance of genetic algorithm and other search methods.

# Chapter 4

# Design and Implementation

## 4.1 Introduction

This chapter presents the proposed system and its implementation details. The proposed system is a hybrid intelligent system of two optimization algorithms, GA and TS which are selected based on the literature survey. GA acts on the input sequences and provides an optimal alignment. However, it is not guaranteed that the optimal alignment produced by GA is the best solution. Sometimes it might be trapped in local minima [3] and may result in an alignment with poor quality. Under such circumstances, the TS part of the algorithm refines the alignment. The designed system is implemented in MATLAB. Moreover, the TS part is implemented in VHDL as well for simulation for FPGA implementation. The VHDL implementation is aimed to show the speed up gained by hardware implementation over the software implementation.

## 4.2 Proposed system

The alignment process begins by reading input sequences from a file. As the string data types are not supported for hardware implementation, the sequences are represented as an array of integers. Moreover, the integer array representation of the sequences takes into account minimizing the number of input output pins required for the FPGA implementation. As stated in section 1.1, DNA sequences are made up of four bases Adenine, Cytosine, Guanine and Thymine. The integer representation of these bases and the gap character is given in Table 4.1.

Table 4.1 Integer representation of bases (nucleotides)

| **Character** | **Symbol** | **Integer representing the bases** |
|---|---|---|
| Adenine | A | 1 |
| Cytosine | C | 2 |
| Thymine | T | 4 |
| Guanine | G | 3 |
| Gap | - | 5 |

Consider the strings of A, C, T and G in table 4.2 as samples of 4 DNA sequences which are to be aligned. The sequences $S_1$, $S_2$, $S_3$ and $S_4$ are, hence, represented as row vectors of integers. Sample fragments and the corresponding integer representation are shown in Table 4.2 and 4.3 respectively.

Table 4.2 Hypothetical fragment of DNA sequences

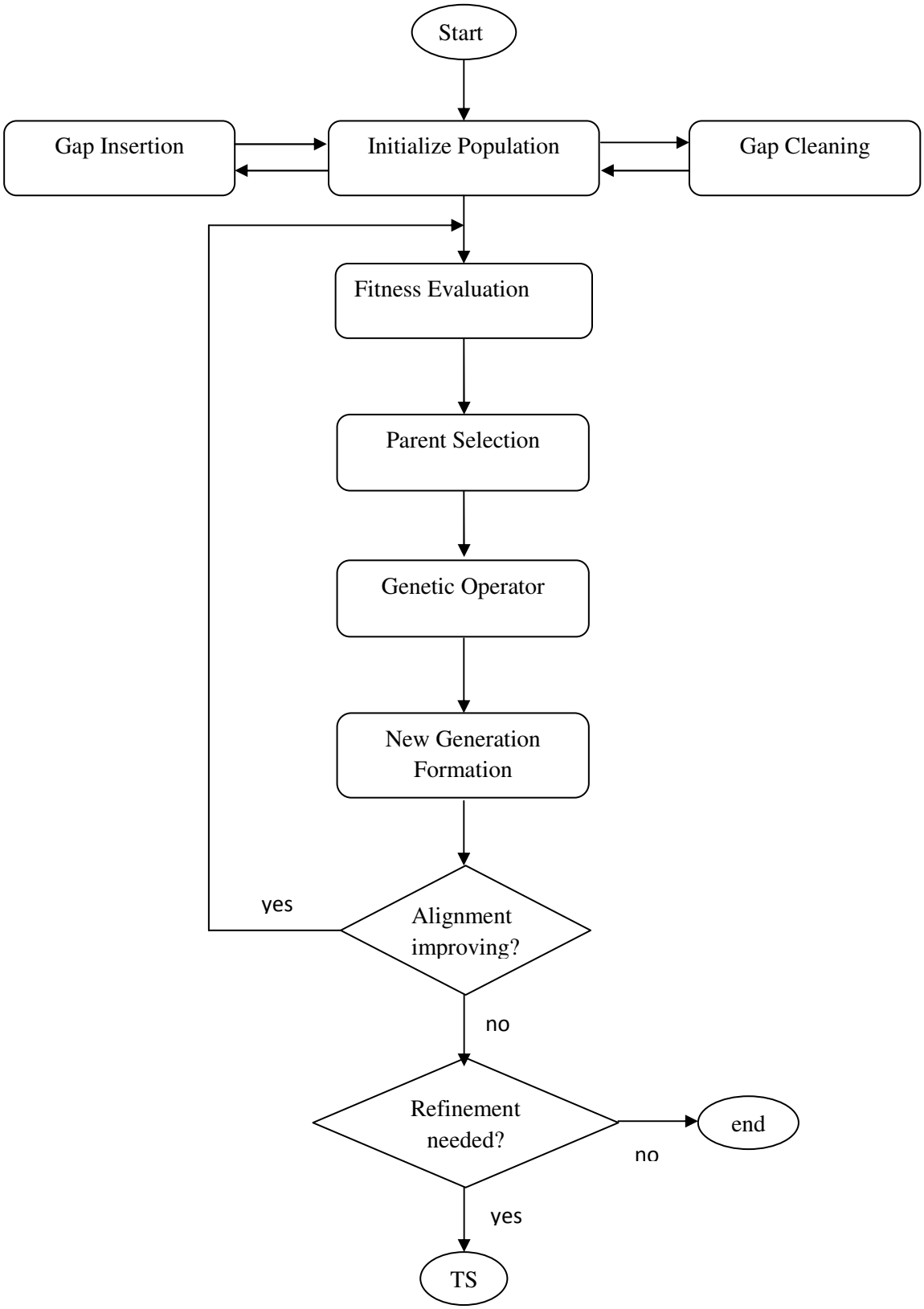
| **Sequence label** | **Integer representing the bases** |
|---|---|
| **$S_1$** | AACTGGGCAACG |
| **$S_2$** | ACTGGCAAACC |
| **$S_3$** | AATGGCAAG |
| **$S_3$** | ACTGCAACG |

Table 4.3 Integer representation of sequences shown in table 4.2

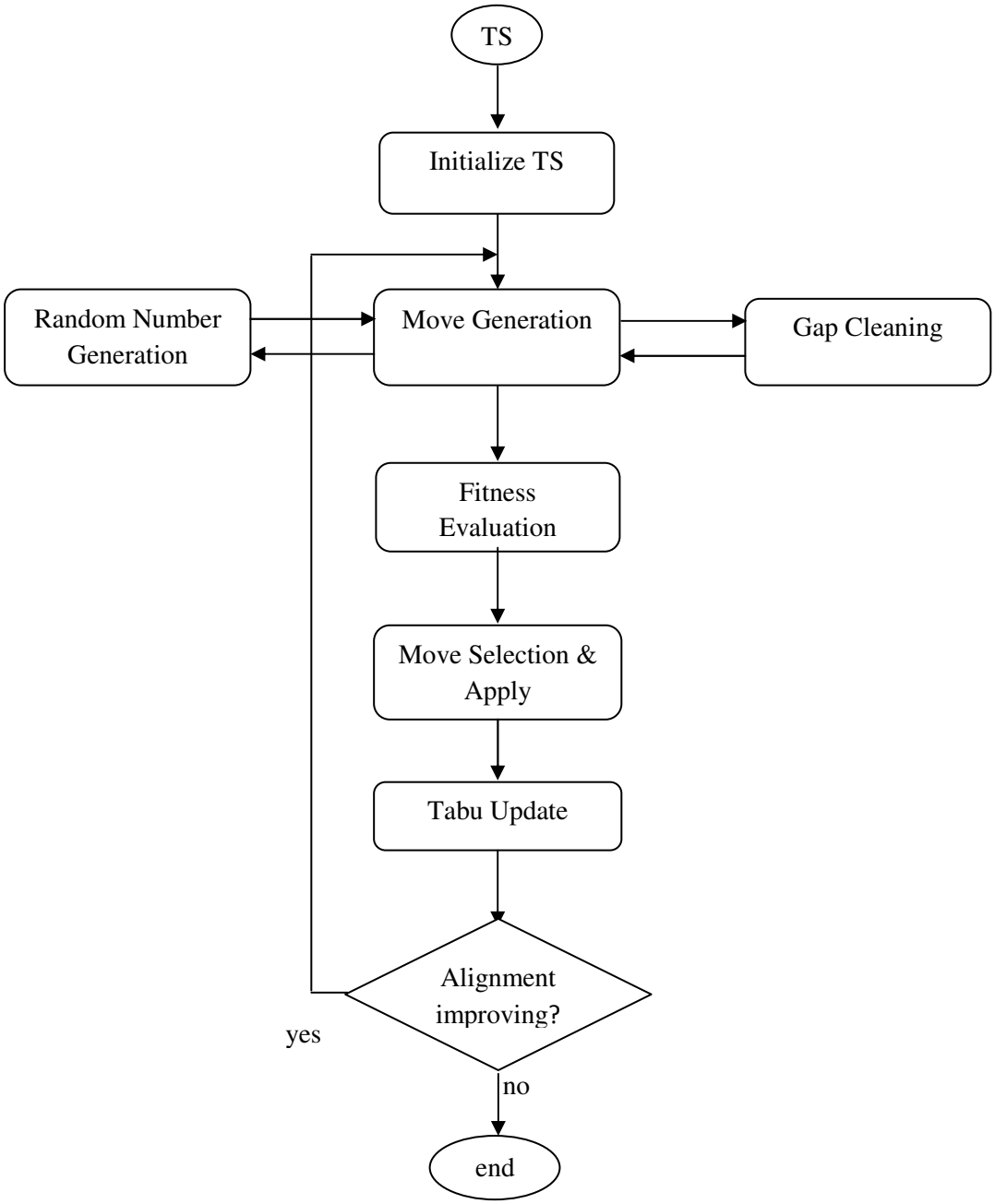
| **Sequence label** | **Integer representing the bases** |
|---|---|
| **$S_1$** | 1 1 2 4 3 3 3 2 1 1 2 3 |
| **$S_2$** | 1 2 4 3 3 2 1 1 1 2 2 |
| **$S_3$** | 1 1 4 3 3 2 1 1 3 |
| **$S_3$** | 1 2 4 3 2 1 1 2 3 |

For the hardware implementation, the sequences are read by the FPGA as binary values via its input output pins. According to this representation, each nucleotide requires 3 pins of the hardware as 3 bits are enough to represent 4 nucleotides and a gap character.

The proposed system manipulates multiple DNA sequences based on the described representation and produces an optimal alignment. Figure 4.1 shows the model of the proposed system.




(a) Genetic Algorithm for multiple DNA sequence alignment




(b) Tabu Search for multiple DNA sequence alignment

Fig 4.1 Hybrid system of GA and TS for multiple DNA sequence alignment

The details of GA and TS algorithms are discussed in sections 4.2.1 and 4.2.2 respectively.

### 4.2.1 Genetic Algorithm phase

The proposed system begins with setting parameters of the GA phase. The size of the population is set to 200. Therefore, the number of alignments in each generation is 200. Crossover and mutation operators have been assigned with probabilities of 0.5 and 0.01 respectively. Among 200 alignments of a given generation, half of the alignments are, hence, selected for mating. As two alignments (children) are produced from two parents, 100 new alignments are formed. According to the mutation probability, therefore, only one of these new alignments is selected for mutation. Once the parameters are set, the sequences to be aligned are read and represented as arrays of integers. Then GA phase performs a series of steps to be discussed in the following sections to produce an optimal alignment.

#### 4.2.1.1 Initial Population Initialization

After the parameters are set and the original sequences are represented as arrays of integers, the system proceeds with the creation of initial alignments. The set of initial alignments is called initial population $G_0$. Given the original sequences $S_1$, $S_2$, $S_3$ …$S_k$ as arrays of integers (row vectors), an alignment is formed as two dimensional matrix of integers M. Each sequence with some gap characters constitute row of the matrix M. That is to say, by inserting a specified number of gap characters at randomly selected positions in each of the sequences [2, 3, 4, 5, 9], the corresponding row of the matrix M is obtained. The number of gap characters to be inserted in each sequence is limited so as to make the length of any row equal. Hence, the size of the matrix M becomes K x N, where K is the number of sequences to be aligned and N is the number of columns of the matrix. It is obvious that N has to be greater than or equal to the length of the longest initial sequence. It is usually the length of the longest sequence up-scaled as per equation (4.1).

$$N = l_{max} * sf \tag{4.1}$$

Where $l_{max}$ is the length of the longest sequence and *sf* is scaling factor. Adequate scaling factor is between 1.4 and 1.6 [20].

Hence, Gap Insertion part of the algorithm performs the process of inserting gaps at random positions in each sequence and forming alignments of K rows and N columns called individuals or chromosomes. The gap insertion is in such a way that when gap characters are removed from any row, the corresponding original sequence (array of integers) results. However, as gap characters are inserted at random positions, there is a possibility that columns containing only gap characters will appear. Under such circumstance, Gap cleaning part of the algorithm cleans such gaps as column(s) of all gap character is/are undesirable.

To illustrate the formation of initial population, consider four sequences $S_1$, $S_2$, $S_3$ and $S_4$ given in table 4.2 and the corresponding integer representation of the sequences given in table 4.3. Since the longest sequence $S_1$ has a length of 12, then $l_{max}$ = 12. Choosing scaling factor of 1.5, N = 18. Therefore, an alignment of 4 x 18 matrix is formed by inserting gaps. One possible alignment can be the one shown in Table 4.4.

Table 4.4 Sample alignment of the sequences of Table 4.2

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 4 | 5 | 3 | 3 | 3 | 5 | 2 | 5 | 5 | 1 | 5 | 1 | 2 | 3 | 5 |
| 1 | 2 | 5 | 4 | 5 | 3 | 5 | 5 | 3 | 2 | 1 | 1 | 5 | 5 | 1 | 2 | 2 | 5 |
| 1 | 5 | 1 | 4 | 3 | 3 | 2 | 5 | 5 | 5 | 5 | 5 | 1 | 5 | 1 | 3 | 5 | 5 |
| 5 | 1 | 2 | 5 | 5 | 4 | 5 | 3 | 5 | 2 | 5 | 1 | 1 | 5 | 5 | 5 | 2 | 3 |

Other alignments are also formed in similar fashion. Therefore, initial population initialization creates 200 (population size used in this work) random alignments.

#### 4.2.1.2 Fitness Evaluation

This module evaluates each individual (alignment) in a population and assigns fitness value to each individual. To be evaluated, each individual is subjected to a scoring function called SOP, Sum-Of-Pairs scoring function. The SOP score of an alignment is obtained by calculating the pairwise scores of all possible combinations of the sequences and then summing up all pairwise scores of the alignment.

Mathematically, the score of aligning any two rows $R_i$ and $R_j$ of an alignment A is given as:

$$score(R_i, R_j) = \sum_{p=1}^{N} score(R_{ip}, R_{jp}) \quad (4.2)$$

where N is the number of columns in the alignment. Then overall score of the alignment A is given by the sum of all pairwise scores in the alignment

$$\text{score (A)} = \sum_{i=2}^{K} \sum_{j=1}^{i-1} score(R_i, R_j) \quad (4.3)$$

While evaluating the score of aligning any two rows of an alignment, a reasonable value is associated with aligning any two characters (A, C, T, G, '-'). In practical terms, the associated value indicates substitution value, which gives the value for match or mismatch, and gap penalty, which is a numerical quantification of insertion or deletion. For DNA sequence, a positive substitution value is used for match and a negative substitution value is used for mismatch [20]. This clearly implies that adding a set score for a match between two sequences and subtracting a penalty for a difference between the two sequences is basis of the scoring. In this research, the scoring value used for match and mismatch given in the Table 4.5 is used.

Table 4.5 Score for matches and mismatches

| | **A** | **C** | **G** | **T** |
|---|---|---|---|---|
| **A** | 4 | -2 | -2 | -2 |
| **C** | -2 | 4 | -2 | -2 |
| **G** | -2 | -2 | 4 | -2 |
| **T** | -2 | -2 | -2 | 4 |

The gap penalty also contributes to the overall score of the alignment and, hence, to the alignment quality. For biological sequence alignment either Constant Gap Penalty or Affine Gap Penalty is used. In case of Constant Gap Penalty scheme, a negative value is used for all the gaps. That is, any gap introduces the same penalty. However, in case of Affine Gap Penalty scheme, initial penalty is used to start a gap and a smaller linear penalty is used for extending the gap an extra position (equation 4.4) [20].

$$d = g + rx \quad (4.4)$$

where g is the gap opening penalty; r is the cost of extending the gap; x is the size of gaps and d is the affine gap penalty. In this work, Constant Gap Penalty scheme is employed. Numerically, penalties of -4 and -3 are used in this work for a gap character aligned with a gap and a gap character aligned with nucleotide respectively.

The Fitness Evaluation Module is, therefore, based on the substitution values (used for match or mismatch) and the gap penalty to map an alignment to a numeric value called fitness value of the alignment.

#### 4.2.1.3 Parent Selection

The fitness values associated with each alignment by the Fitness Evaluator module serves as a basis for selection of parents. Since the SOP scoring used is a measure of similarity, the higher the fitness value, the better the alignment quality is. In this work, half of the parents to be selected for mating are simply the alignments in the population having higher fitness values and the remaining are alignments having higher matching characters. However, duplicates are not allowed when selecting parents. Therefore, the better alignments in the generation are selected and the rest half are simply discarded.

#### 4.2.1.4 Genetic Operation

The formation of new alignments and the modification of the newly formed alignment are the two tasks carried out after parent selection. These tasks are operations of GA namely, Crossover and Mutation.

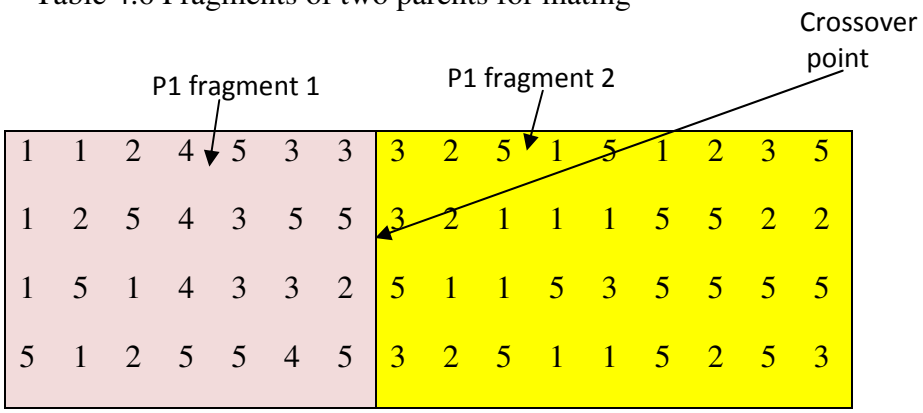
***Crossover***

As Crossover is the process of creating new individual by combining the genetic information from two parents, the Genetic Operator Module can combine two alignments through single exchange (One-point crossover) or two point exchange (two point-crossover). However, one-point crossover is simpler and is used in this work.

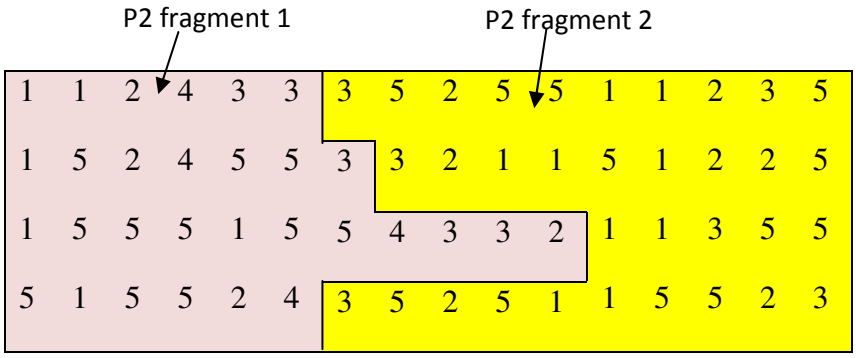
One of the alignments is cut straight at some random position. As a result two fragments of the alignment are obtained as shown in Table 4.6 a. The second alignment is also fragmented in to

two parts as given in Table 4.6 b. The fragmentation of the alignment is, however, not done straight at random position. Rather it tailored so that the number of nucleotides of the resulting fragments conforms to the number of nucleotides in the fragments of the first alignment. Simply, the number and order nucleotides in fragment No. 1 of alignment 1 should conform to the number and order of nucleotides in fragment No. 1 of alignment 2. The same criterion needs to be met for second fragments of the two alignments.

Table 4.6 Fragments of two parents for mating




a. Fragments of parent (alignment)1




b. Fragments of parent (alignment) 2

In doing so, a fragment may not have sequences of equal lengths. If such condition happens, any void spaces that appear at the junction point are filled with gaps. Now, two fragments one from each alignment are combined to produce two new alignments as shown in Table 4.7.

Table 4.7 new alignments after crossover

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 4 | 5 | 3 | 3 | 3 | 5 | 2 | 5 | 5 | 1 | 1 | 2 | 3 | 5 |
| 1 | 2 | 5 | 4 | 3 | 5 | 5 | 5 | 3 | 2 | 1 | 1 | 5 | 1 | 2 | 2 | 5 |
| 1 | 5 | 1 | 4 | 3 | 3 | 2 | 5 | 5 | 5 | 5 | 5 | 1 | 1 | 3 | 5 | 5 |
| 5 | 1 | 2 | 5 | 5 | 4 | 5 | 3 | 5 | 2 | 5 | 1 | 1 | 5 | 5 | 2 | 3 |

a. Alignment from p1 fragment1 and p2 fragment 2

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 4 | 3 | 3 | 5 | 5 | 5 | 5 | 5 | 3 | 2 | 5 | 1 | 5 | 1 | 2 | 3 | 5 |
| 1 | 5 | 2 | 4 | 5 | 5 | 3 | 5 | 5 | 5 | 5 | 3 | 2 | 1 | 1 | 1 | 5 | 5 | 2 | 2 |
| 1 | 5 | 5 | 5 | 1 | 5 | 5 | 4 | 3 | 3 | 2 | 5 | 1 | 1 | 5 | 3 | 5 | 5 | 5 | 5 |
| 5 | 1 | 5 | 5 | 2 | 4 | 5 | 5 | 5 | 5 | 5 | 3 | 2 | 5 | 1 | 1 | 5 | 2 | 5 | 3 |

b. Alignment from p2 fragment 1 and p1 fragment 2

***Mutation***

Mutation operator slightly alters the alignment to introduce a modification of the newly formed alignment. For a newly formed alignment, the mutation site is randomly selected. Then if the random character is gap character, it is switched with the neighboring non-gap character as shown in table 4.8.

Table 4.8 Mutation of an alignment

Mutation site

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 4 | 5 | 3 | 3 | 3 | 5 | 2 | 5 | 5 | 1 | 1 | 2 | 3 | 5 |
| 1 | 2 | 5 | 4 | 3 | 5 | 5 | 5 | 3 | 2 | 1 | 1 | 5 | 1 | 2 | 2 | 5 |
| 1 | 5 | 1 | 4 | 3 | 3 | 2 | 5 | 5 | 5 | 5 | 5 | 1 | 1 | 3 | 5 | 5 |
| 5 | 1 | 2 | 5 | 5 | 4 | 5 | 3 | 5 | 2 | 5 | 1 | 1 | 5 | 5 | 2 | 3 |

a. Alignment before mutation

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 4 | 5 | 3 | 3 | 3 | 5 | 2 | 5 | 5 | 1 | 1 | 2 | 3 | 5 |
| 1 | 2 | 5 | 4 | 5 | 3 | 5 | 5 | 3 | 2 | 1 | 1 | 5 | 1 | 2 | 2 | 5 |
| 1 | 5 | 1 | 4 | 3 | 3 | 2 | 5 | 5 | 5 | 5 | 5 | 1 | 1 | 3 | 5 | 5 |
| 5 | 1 | 2 | 5 | 5 | 4 | 5 | 3 | 5 | 2 | 5 | 1 | 1 | 5 | 5 | 2 | 3 |

b. Alignment after mutation

Therefore, the Genetic Operator Module is responsible for the continuation of the generation with modification thereby enabling the appearance of the better alignments from one generation to the next.

#### 4.2.1.5 Formation of new generation

Once new alignments (half as much as the number of population) are formed using genetic operators, the alignments formed directly become members of the next generation. This implies that half of the alignments of the new generation are formed from the newly created alignments. The rest half are fitter alignments which are selected for mating in the previous stage called parent selection.

#### 4.2.1.6 GA convergence check

At last, the decision as to whether to iterate back in the GA phase or to end the algorithm or to further refine the alignment if the need arises is made. The fitness value of the optimal alignment and the number of matching pairs of nucleotides in the optimal alignment in each generation serves as criteria for convergence check. Furthermore, the iteration count and a reference fitness value are also used to check convergence of the search process. A fitness value used as a reference in this work is the fitness value of the alignment aligned using most commonly used multiple sequence alignment method called Clustalw.

If the fitness value of the optimal alignment and/or the matching pairs of nucleotides in that alignment keeps changing over number of successive iterations, the GA iterates back to fitness

evaluation step. If the alignment doesn't show improvement in terms of both fitness value and matching pairs of characters over successive iterations, then it is an indication that either the GA phase has found out best alignment by itself or the current alignment needs further refinement.

The reference fitness value now is made use of as to whether further process to find optimal solution is required or not. If the fitness value of the current solution is less than the Clustalw's fitness value (i. e., threshold), then it is regarded as optimal solution has not been found yet. Therefore, the TS phase proceeds with the search process. On the other hand if the fitness value of the current solution exceeds the threshold value, the search process quits. In this case, the fittest of the alignments of the current generation is the one taken as optimal solution.

## 4.2.2 Tabu Search phase

TS phase of the proposed system takes the fittest of all alignments of the last generation. It sets this alignment as its current solution. The search process then proceeds iteratively from the current solution to another solution in a way that strives to improve the alignment. In fact, the parameters of the TS need to be defined and initialized before beginning the search process.

### 4.2.2.1 Tabu Search Initialization

As mentioned in the preceding section, the optimal alignment obtained from the GA phase is set as the current solution of the TS. A list of forbidden alignments called Tabu list is set to zero. Besides, a list of candidate alignments called Candidate Move list is also set to zero. It is usual that a move in a Tabu List can be applied after a number of iterations called Tabu Tenture. In this work Tabu Tenture is set to ten. Hence, an alignment in Tabu list can be taken as the solution after ten iterations. Moreover, a criterion that can override the tabu criterion is defined. This criterion is called aspiration criterion. In this work, the number of matching pairs of nucleotides being the maximum value ever attained during the search process is used as aspiration criterion. This enables the TS to take an alignment in Tabu list as a solution after iterations less than Tabu Tenture provided the aspiration criterion is met.

### 4.2.2.2 Move Generation and Evaluation

TS has a capability of exploring new solutions by means of generating different alignments from a given alignment. This is called move generation. It is a movement of gaps in sequences of an alignment in such a way that the original order of the nucleotides in the sequences doesn't change.

There are two types of sequence moves, single sequence move and block sequence move. Single sequence move is a move where gap(s) is/are made to move within a sequence for each of the sequence. Block sequence move is a move where block of gaps are made to move within the alignment. This module performs single sequence move for each sequence and block sequence move for every possible block of gaps during each iteration of the TS part. The chart given in Table 4.9 gives an example of block sequence move.

Table 4.9 Example of block move

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 4 | 5 | 3 | 3 | 3 | 5 | 2 | 5 | 5 | 1 | 1 | 2 | 3 | 5 |
| 1 | 2 | 5 | 4 | 5 | 3 | 5 | 5 | 3 | 2 | 1 | 1 | 5 | 1 | 2 | 2 | 5 |
| 1 | 5 | 1 | 4 | 3 | 3 | 2 | 5 | 3 | 5 | 5 | 5 | 1 | 1 | 3 | 5 | 5 |
| 5 | 1 | 2 | 5 | 5 | 4 | 5 | 3 | 5 | 2 | 5 | 1 | 1 | 5 | 5 | 2 | 3 |

Block of gaps selected at random

a. Alignment before block move

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 4 | 5 | 3 | 3 | 3 | 5 | 2 | 5 | 5 | 1 | 1 | 2 | 3 | 5 |
| 1 | 2 | 5 | 4 | 5 | 3 | 5 | 3 | 5 | 2 | 1 | 1 | 5 | 1 | 2 | 2 | 5 |
| 1 | 5 | 1 | 4 | 3 | 3 | 2 | 3 | 5 | 5 | 5 | 5 | 1 | 1 | 3 | 5 | 5 |
| 5 | 1 | 2 | 5 | 5 | 4 | 5 | 3 | 5 | 2 | 5 | 1 | 1 | 5 | 5 | 2 | 3 |

b. Alignment after block move

Sometimes, the moves generated might have column(s) of all gap characters. When such case arises, this module makes use of Gap Cleaning sub-module to clean the Gaps in such column(s) filled entirely with gap characters. For instance, in the alignment of Table 4.9 b, the ninth column is filled up wholly with gaps. This column is cleaned and a fitter alignment is produced. The module also evaluates the moves and assigns a fitness value to each of the moves using the same Fitness Evaluation scheme used for the GA phase. Then the moves having higher fitness value than the current optimal alignment are added to candidate Move List.

#### 4.2.2.3 Move Selecting and Applying

In this module, best move is selected from the candidate Move List (ML) based on its absence in the Tabu List. First alignment having highest fitness value, which is the best of all alignments in the candidate move list, is selected. The alignment is then checked for its presence in Tabu List. If the move (alignment) exists in Tabu List, it is discarded and the next best move is tried in order of decreasing fitness value until all the moves are checked or until a move not available in TL is found. Once a move which is not available in TL is found, it is applied. A move can be applied though it is in tabu list provided that the aspiration criterion is met The applied move, then, becomes the current solution. Nevertheless, if all the moves in the candidate ML are available in TL, the current solution remains being the former solution.

#### 4.2.2.4 Tabu Search Updating

Having applied a move or rejecting all the moves, the algorithm iterates back to Moves Generation and Evaluation until the current solution stabilizes. Before going back to Moves Generation and Evaluation step, the TS parameters need to be updated. Tabu List is updated provided a valid move had existed so that the move currently applied is added to Tabu List. Also, Tabu Tenture of each alignment in the Tabu List is decremented by one. A move in the Tabu List whose Tabu Tenture reaches zeros is freed from the forbidden list.

#### 4.2.2.5 Fitness Evaluation

Fitness Evaluation follows exactly same scheme as that for the GA phase of the system.

#### 4.2.2.6 TS convergence check

Lastly, TS iterates back to the move generation and evaluation step or terminates the algorithm providing the current solution as optimal alignments of the sequences. The quality of the current solution (the fitness value and the number of matching pairs of nucleotides) over number of successive iterations serves as a basis of convergence check. If the fitness value and the number of matching pairs of nucleotides of current solution over successive iterations don't changes, the search for optimal alignment terminates. If both or one of the two keeps on changing nevertheless, the TS continues the optimization process.

## 4.3 Implementation of the proposed system

The proposed system is implemented in MATLAB and the refining part, the TS, is implemented in VHDL as well. The VHDL implementation of the TS has been carried out to come up with hardware module that can be ported to FPGA. In the upcoming sections, the hardware block of TS and its synthesis are discussed.

### 4.3.1 The hardware block

As mentioned in the preceding section, the TS part of the system is implemented in VHDL. The MATLAB design serves as the basis of this implementation. The MATLAB design is tailored in a format the AccelDSP allows for synthesis. Then the MATLAB floating point design is converted to desired hardware module that can be implemented on Xilinx FPGA. Figure 4.2 shows the block of the TS hardware.

The data ports of the hardware derive their names from the MATLAB variables used to move data in and out of the design function. Therefore, the ports csIN and csOUT shown in Figure 4.2 which are used to move the sequences in and out of the hardware block is simply the name given to MATLAB variables for current solution input and current solution output respectively. Similarly, Fitness and seed are MATLAB variable names and hence, they are used to designate the ports which move seed value to the block and fitness value from the block.

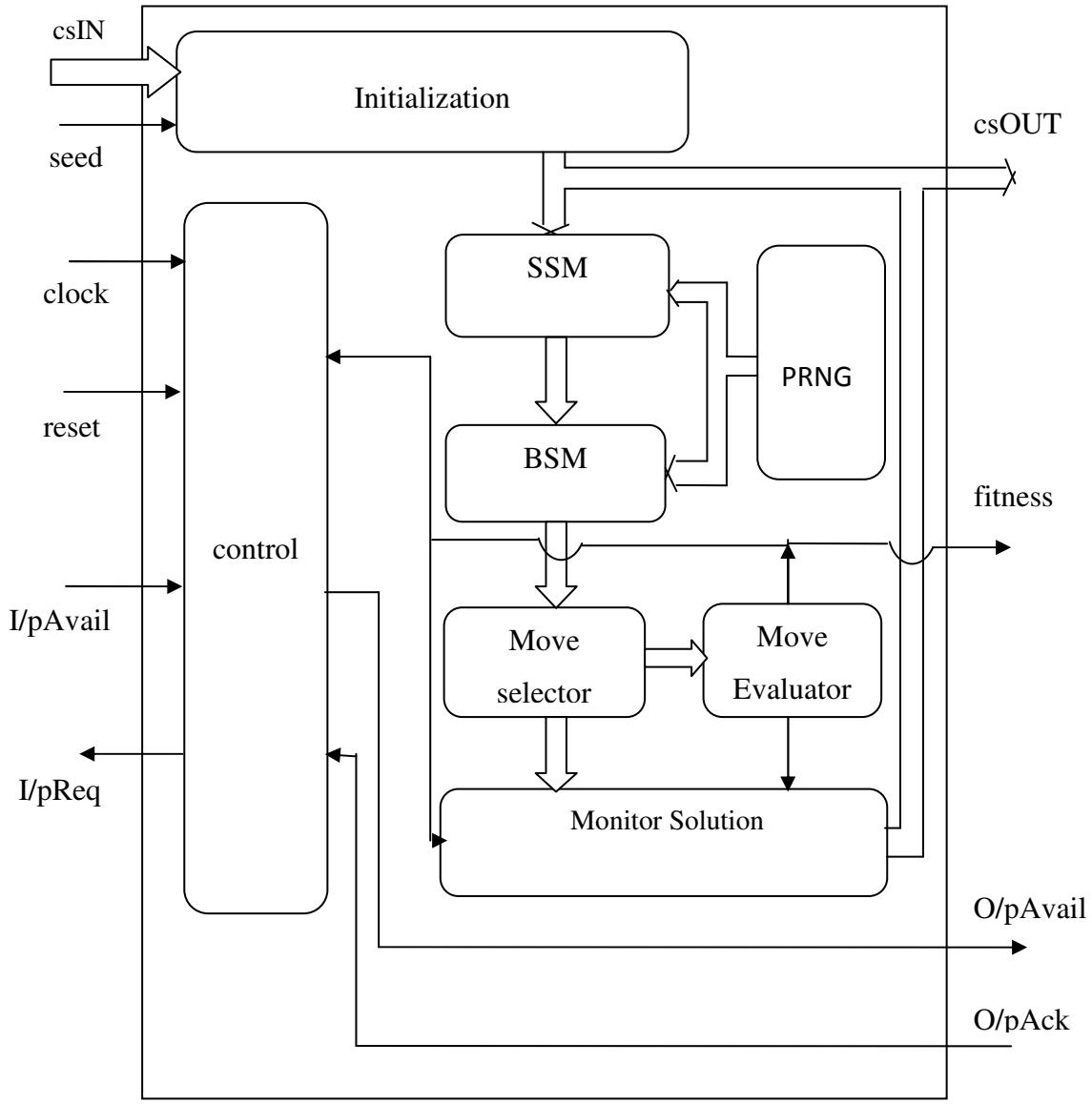



Figure 4.2 FPGA based Hardware block of TS

## 4.3.2 Modules in the block

### 4.3.2.1 Initialization module

- This module carries out TS initialization process explained in section 4.2.2.1. In addition to initializing the parameters of the TS, this module also handles updating the parameters during the search process (section 4.2.2.4).

#### 4.3.2.2 SSM and BSM modules

- The SSM (Single Sequence Moves) and BSM (Block Sequence Moves) modules are responsible for generation of moves as discussed in section 4.2.2.2.

#### 4.3.2.3 PRNG module

- Random numbers required for generating moves in SSM and BSM modules are generated in module called PRNG (Pseudo Random Number Generator) module.

#### 4.3.2.3 Move Evaluator module

- This module evaluates the fitness of each move (alignment) as discussed in section 4.2.1.2.

#### 4.3.2.4 Move Selector module

- This module selects which move has to be applied as current based on TS parameters and fitness values of the moves. How it performs is already discussed in section 4.2.2.3

#### 4.3.2.5 Solution monitoring module

- Solution monitoring module applies the selected move and holds the current solution. This module also performs gap cleaning operation. In addition, it keeps record of the fitness values of all the moves that are applied during the search process.

#### 4.3.2.6 Control module

- This module handles the TS convergence check discussed in section 4.2.2.6. Hence, it is responsible for the search process either to quit or proceed.

### 4.3.3 Interface Configuration

A MATLAB design that is synthesized by AccelDSP has either Push mode Interface protocol or Full handshake Interface protocol. If the number of clock cycles per design function call is constant, then by default Push mode Interface protocol is used. Otherwise, Full handshake Interface protocol is employed. However, it is possible that the later is used while the design has

constant latency per design function call also. In this design, Full handshake protocol is used. Consequently, the flow of data into and out of the hardware ports is controlled by a handshake interface protocol as shown in Figure 4.3.

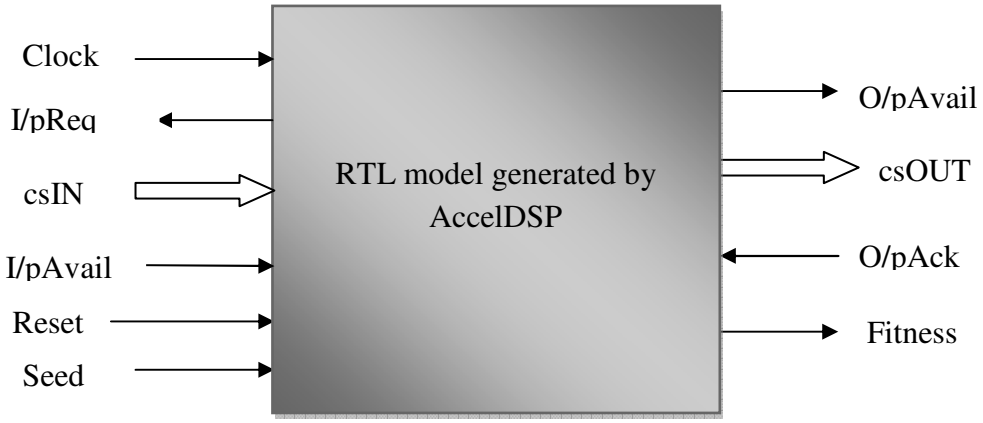



Figure 4.3 Full Handshake Interface of the hardware module

In this scenario, the resulting hardware module is, therefore, granted to be a part of the larger design on FPGA.

#### 4.3.3.1 Global Signals

The hardware module has two global signals Clock and Reset. Data transfers on each data port are synchronized to the Clock. The global Reset helps the registers within the block to be in a known state. Initially it must be held active high for at least one clock cycle and returns all registers to a known state.

#### 4.3.3.2 Input Synchronizing Signals

**I/pAvail**: This signal is controlled by the external design. When this signal is set high, it indicates that data on the input ports is valid. This causes the receiving device (the hardware module) to capture the data on the rising edge of the next clock cycle.

**I/pReq**: This signal is controlled by the hardware module. This signal when set high, on the other hand, indicates that the module is ready to capture new data from the input ports on the

rising edge of the next clock cycle. However, when the module sets this signal low, the external design should immediately stop sending new data as the module is not ready to capture new data.

#### 4.3.3.3 Output Synchronizing Signals

**O/pAvail**: This signal is controlled by the hardware module. When set high, it indicates that data on the output ports is valid. This initiates the receiving device to capture the data on the output ports. Once this signal is set high, it will remain high until the receiving device acknowledges the data capture by setting ac_OutputAck high.

**O/pAck**: This signal is controlled by the external design. High state of the signal is acknowledgment for the hardware module that the data on the output ports has been captured by the external design.

### 4.3.4 Hardware Synthesis

The hardware block given in figure 4.2 is synthesized to a module that can be implemented in FPGA. To do so the MATLAB design has to follow some guidelines so as to be synthesized by the synthesis tool. Having MATLAB code in proper coding style, the design passes through synthesis stages. The next section describes the MATLAB coding style for AccelDSP synthesis and the synthesis stages.

#### 4.4.4.1 Coding Style for hardware Synthesis

The synthesis tool, AccelDSP, requires the floating point design conform to guidelines for coding style. The floating point model is represented by script M-File named *script.m* and Design function M-file named *tabuSearch.m*. The structure of the code style accepted by the synthesis tool is given in Figure 4.4.

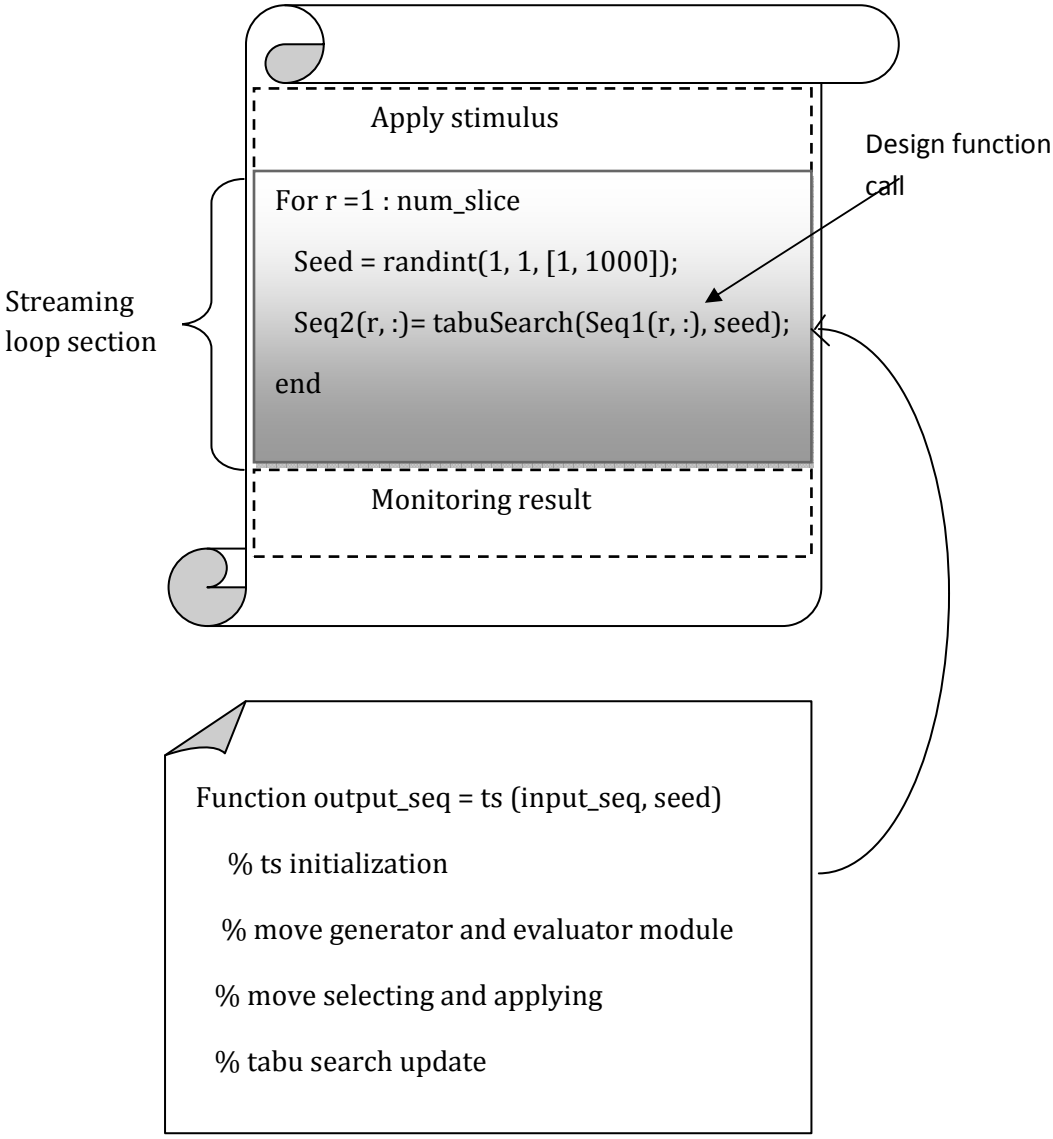



Figure 4.4 Code styles of script M-File and Design function M-File

***Script M-File***

The script M-File called *script.m* has three sections. The top section provides input stimulus to the streaming loop where as the bottom section monitors the result of the streaming loop. The middle section, the streaming loop section, contains the design function to be synthesized. The design function to be synthesized is named *tabuSearch.m.* For hardware synthesis, the infinite streams of data entering and leaving the design must be partitioned into manageable slices in

order to process the data [21]. As the number of pins of the target device is limited to 502, an alignment is sliced into a number of slices with each slice having a maximum of 64 nucleotides so that the number of input output lines required is less than 502. Given the alignment of Table 4.7 b, a slice having 64 nucleotides (see Table 4.10) is formed by taking the first 16 columns of the alignment.

Table 4.10 Example of slice of alignment to be passed to the design function

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 4 | 3 | 3 | 5 | 5 | 5 | 5 | 5 | 3 | 2 | 5 | 1 | 5 |
| 1 | 5 | 2 | 4 | 5 | 5 | 3 | 5 | 5 | 5 | 5 | 3 | 2 | 1 | 1 | 1 |
| 1 | 5 | 5 | 5 | 1 | 5 | 5 | 4 | 3 | 3 | 2 | 5 | 1 | 1 | 5 | 3 |
| 5 | 1 | 5 | 5 | 2 | 4 | 5 | 5 | 5 | 5 | 5 | 3 | 2 | 5 | 1 | 1 |

The streaming loop, which is for loop construct, makes a call to top-level design function and passes the slice of an alignment and seed value for pseudo random number generator module.

***Design function M-File***

The body of the design function M-File namely *tabuSearch.m* models the main processing algorithm. This code is synthesized into hardware block [31]. Other external functions are called from this top-level design function. Such external functions are SSM, BSM, Fitness Evaluator, Move Selector, Tabu Updater and PRNG modules shown in figure 4.2. These external functions (modules) are synthesized into sub-blocks of the hardware.

The MATLAB floating point design having been ready according to the coding styles for the AccelDSP synthesis, the design passes through synthesis stages discussed hereafter.

#### 4.4.4.2 Synthesis stages

***Floating Point Design Verification***

MATLAB floating point design is run and the verification constructs (results) at the end of the script file are obtained. This result will be used as a basis for comparison in future simulation runs (for comparison with the result of fixed point model).

***Floating Point Analysis***

The MATLAB floating point model is analyzed. That is the streaming loop and the top level design functions are identified. The shapes of the variables are also analyzed at this stage. As a result of floating point analysis, an equivalent in-memory data model of the design is created. Any changes that will be made to the design from this point are made to the in-memory design, not to the MATLAB source.

***Fixed Point Generation***

At this stage, an equivalent fixed point model is generated in either C++ or MATLAB. Important information about the design like loop unroll status of loops, quantization of variables are also generated as generate fixed point report. This report enables us to modify the design.

***Fixed Point Verification***

A MATLAB simulation is run on generated fixed point model and the result is compared to the simulation result of Floating Point verification. The design needs to be modified until the deviation is acceptable.

***RTL Generation***

At this stage, the Register Transfer Level (RTL) of the in-memory design is generated in VHDL or Verilog format. AccelDSP also generates a Testbench which applies input stimulus to the design, then monitors and compares the output results. The Test bench is generated in the same language as the RTL.

***RTL Verification***

During RTL verification step, the Testbench generated during RTL generation step applies stimulus to the RTL model (commonly called design under test, DUT). The stimulus applied to the DUT is from the input file that was applied during MATLAB Fixed point simulation (Fixed point verification). At this stage, the result obtained is compared with the simulation result of Fixed Point verification stage. If the result matches, the simulation passes and generates a timing report.

## 4.4 Summary

The hybrid system of GA and TS is used in this work to find the optimal alignment of multiple DNA sequences. The whole system is implemented in MATLAB. Moreover, the system is tailored so that TS phase is implemented on Xilinx FPGA xc3s1400an device of Spartan 3A family. The hardware implementation of the TS phase makes use Xilinx AccelDSP synthesis tool 10.1 and Xilinx ISE Design suite.

# Chapter 5

# Results and Discussion

## 5.1 Introduction

In this work, the alignment of multiple DNA sequences by the proposed hybrid system of GA and TS has been achieved. As described in section 4.3 of the previous chapter, the system is wholly implemented using MATLAB and the TS part is implemented in VHDL too. The implementation has revealed the effect of hybridization. The performance of the system has also been compared to selected methods. The benchmark methods selected are two known software packages being used by EBI. These software packages are called CLUSTALW and MAFFT. CLUSTALW falls in class of progressive approach whereas MAFFT combines the features of both progressive and iterative approaches. In the upcoming sections, test case sequences, the effect of hybridization, performance evaluation and speed up of the hardware implementation are discussed.

## 5.2 Test case sequences

For testing the proposed system, DNA sequences are taken from database of EBI [25]. The sequences are organized as four test cases based on the number of sequences in the test case. For example, the first test case is composed of 2 sequences which are taken from class of 'mammal'. For all the test cases, the number of sequences constituting the test cases with the maximum and minimum length of the sequences comprising the test cases is given in table 5.1

Table 5.1 sequences used as test cases

| Test case | Number of sequences | Minimum Length | Maximum Length |
|---|---|---|---|
| 1 | 2 | 682 | 903 |
| 2 | 4 | 444 | 724 |
| 3 | 8 | 470 | 810 |
| 4 | 16 | 203 | 879 |

Table 5.2 shows a portion of the first sequence in test case one as a sample. As can be shown in the table, sequence begins with '>' character which indicates that the nucleotide bases begin from the next line. The first line, which is the line having '>' at the beginning, is description of the sequence. For instance, the sequence shown in table 5.2 has an Identification number, ID HQ395007. With this ID, it can be identified from all the sequences in the EBI database. It is also shown in abbreviations that this sequence is taken from mammal (MAM indicates that). The number of base pairs, BP 682, is also given along with the number of nucleotide bases in the sequence. The number of base pairs is simply the length of the sequence.

Table 5.2 Sample sequence (sequence 1 of test case 1)

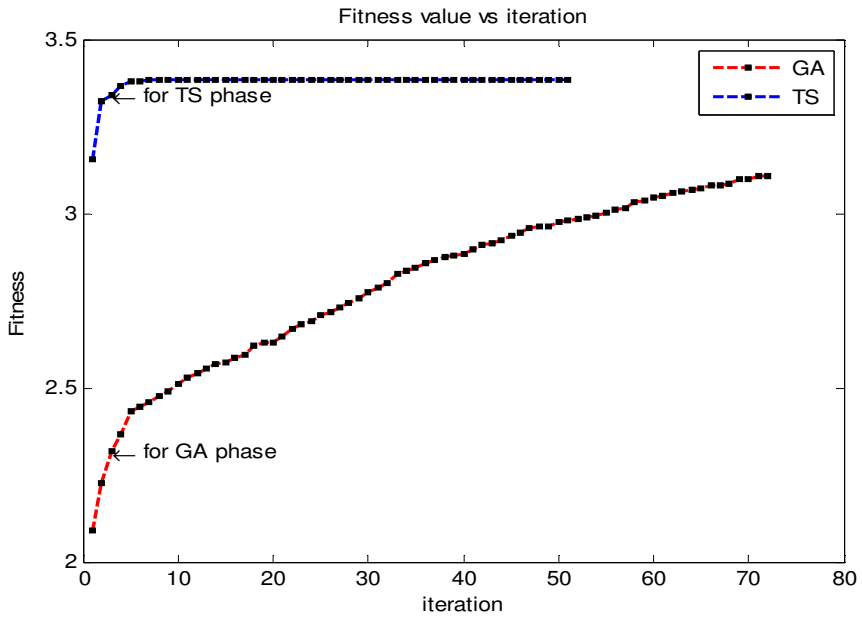
| |
|---|
| **>SQ1: ID HQ395007; MAM 682 BP; 145 A; 197 C; 200 G; 140 T; 0 OTHER;** |
| GGCCGAGGGACAGCTGACACTGCAGCAGTTTGCGCAGTCCACGGAGATGC |
| TGAAGCGCGTGGTGCAGGAGCACCTACCGCTGATGAGCGAAGCGGGCGCC |
| GGCCTGCCCGACATGGAGGCTGTGGCGGGTGCCGAAG…….. |

The entire sequences forming test case one is given in Appendix A.

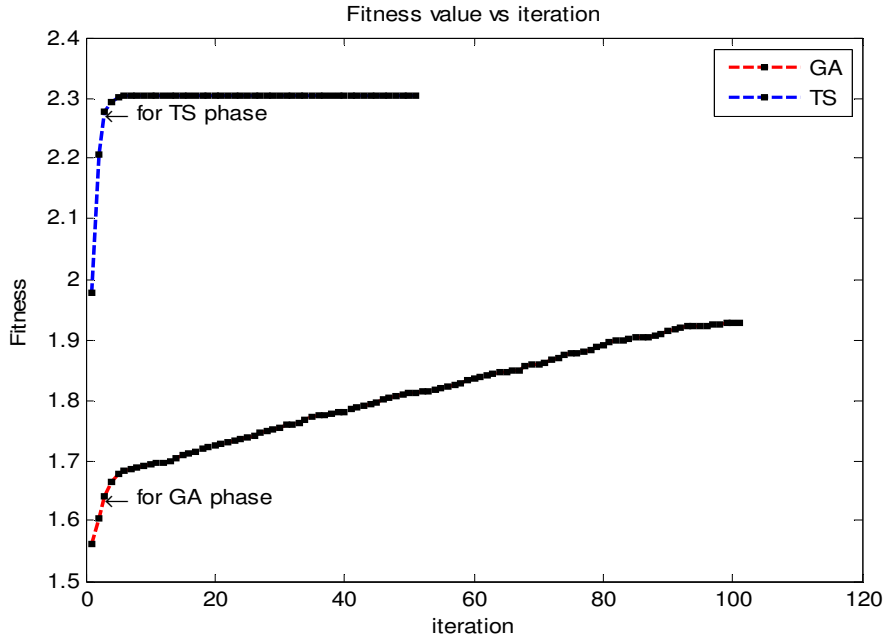
## 5.3 Effect of hybridization

It is customary that combination of two or more different methods is used to solve real world problems. In this work, TS has been incorporated into GA believing that it improves the alignment quality. To show this effect, the sequences of each test cases mentioned in the preceding section have been used as input to the proposed system.

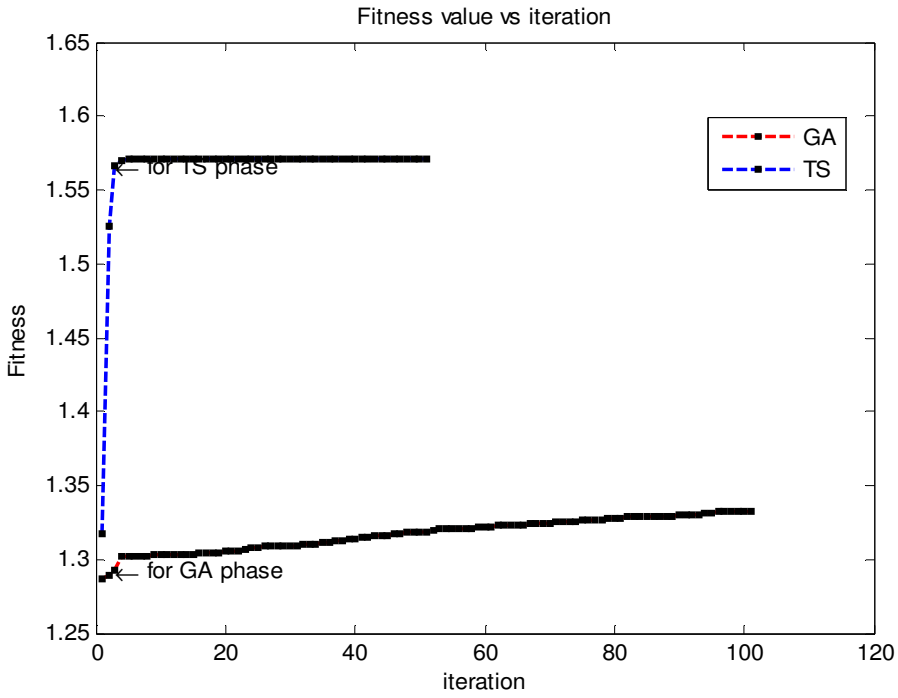
As the alignment process proceeds, the trend of the alignment quality called fitness value of an alignment is traced in steps of some number of iterations (10 iterations used in this work). The fitness value versus iteration count in steps of ten is then graphed. Four plots of Figure 5.1 show the trend of alignment quality by the proposed hybrid system for the four test cases. The trend of alignment quality by GA as standalone system can be observed from figure 5.1 whereas that of standalone TS is given in figure 5.2.



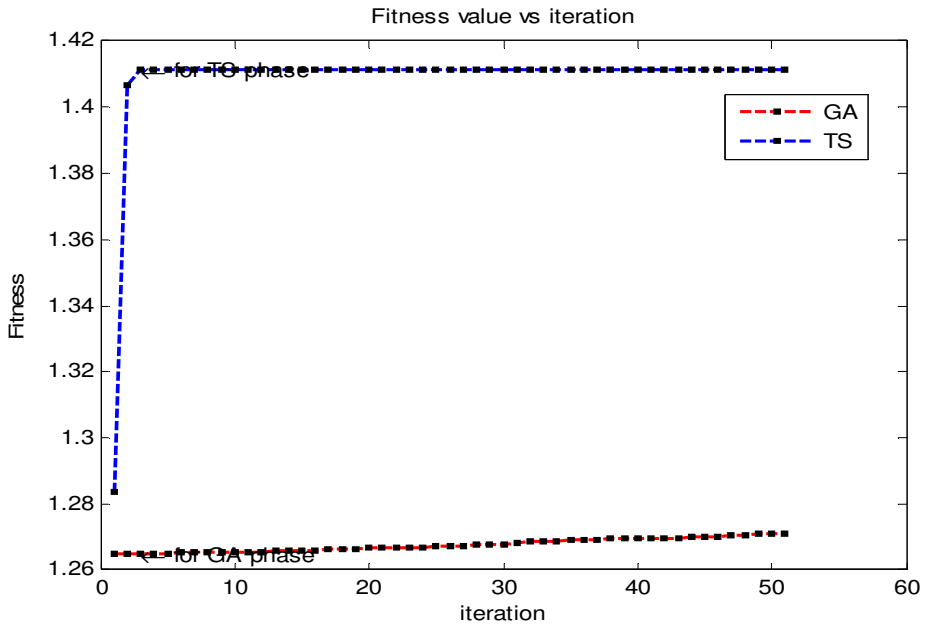

a. Fitness versus iteration count for test case 1




b. Fitness versus iteration count for test case 2

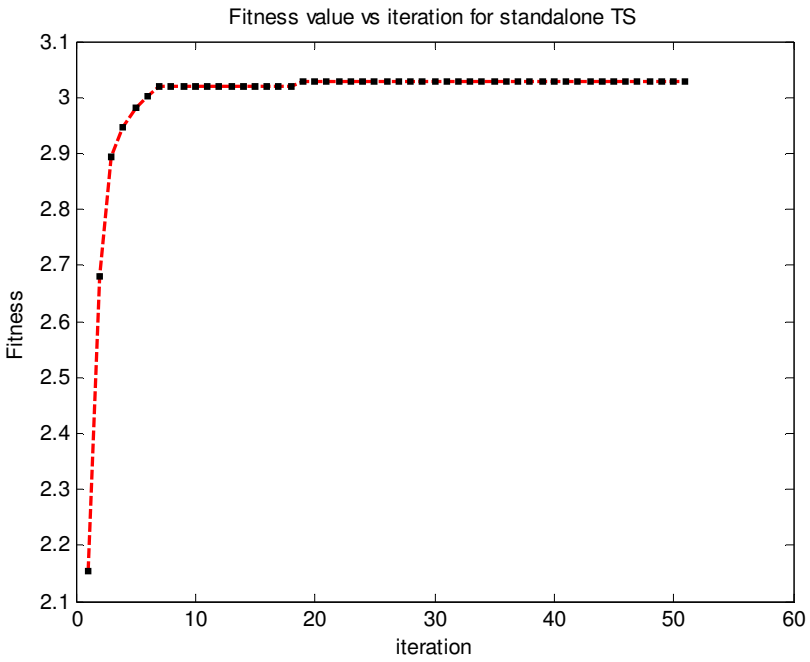



c. Fitness versus iteration count for test case 3

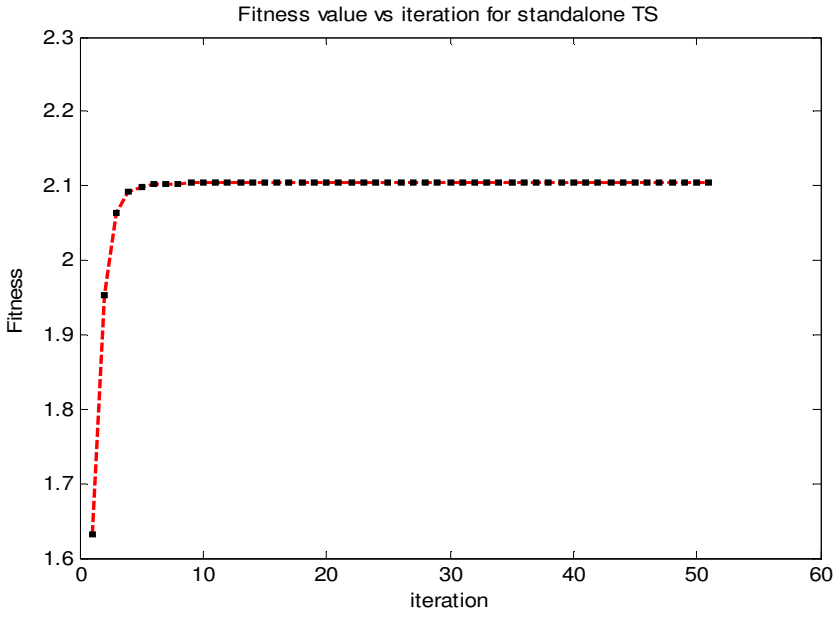



d. Fitness versus iteration count for test case 4

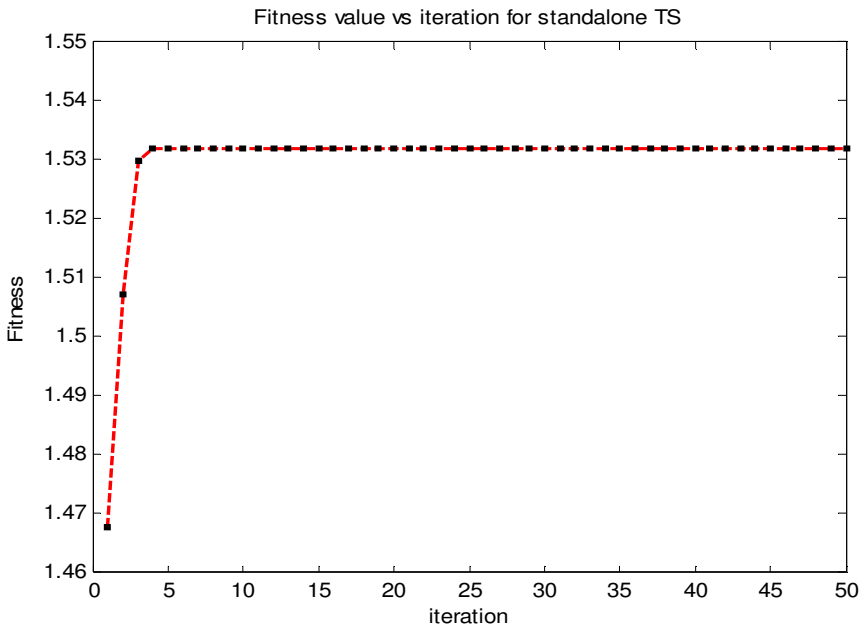
Figure 5.1 Plot of fitness value of alignment versus iteration by the hybrid system of GA and TS



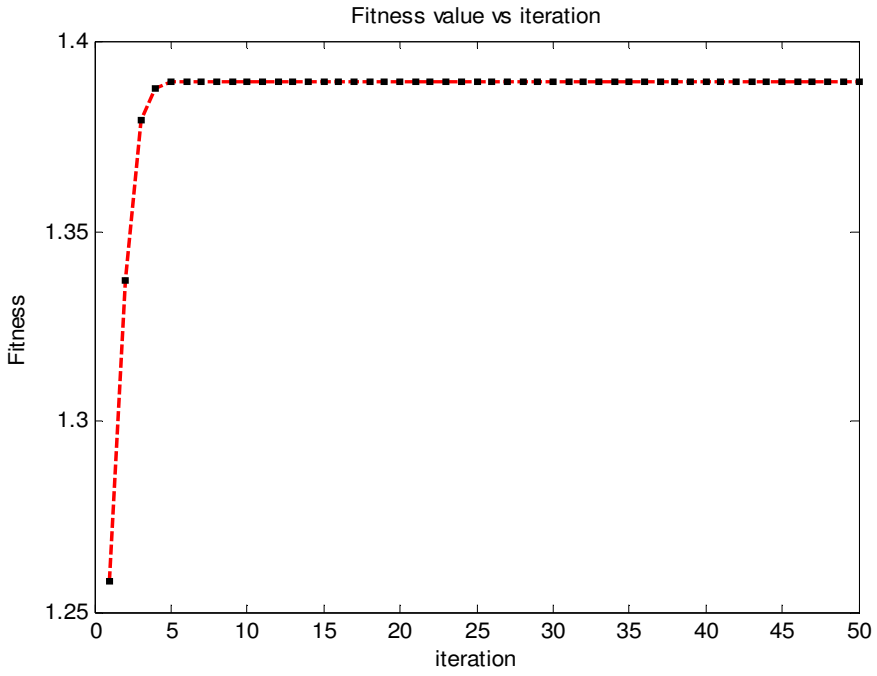

a. Fitness versus iteration count for test case 1




b. Fitness versus iteration count for test case 2




c. Fitness versus iteration count for test case 3




d. Fitness versus iteration count for test case 4

Figure 5.2 Plot of fitness value of alignment versus iteration count by standalone TS

*Note that in figure 5.1 the fitness values are sampled at intervals 10 iteration count. Hence, the last fitness value shown in figure during GA phase and the first fitness value shown during TS phase might be different. This is the case when the number of iteration at which the GA stops is not multiple of 10.*

From figure 5.1 and 5.2, the performance of the standalone GA (as seen from Figure 5.1) and standalone TS, is less as compared to the performance of the hybrid system of GA and TS as expected. In the following paragraphs, the effect of hybridizing GA with TS is discussed.

From the graph of figure 5.1, it can be seen that GA has the effect of enhancing the fitness value as the algorithm proceeds from iteration to the next. The fitness value has shown no sign of decrement for all the test cases. This is the reflection of the fact that the selection of parents for both crossover operation and formation of new generation is based on elitism as explained in chapter four. That is to say, alignments having higher quality, fitness values and higher number of matching nucleotides, are selected during parent selection stage of the GA and so are during formation of new generation. This enforces the continuity of higher quality alignments for generations as long as the newly created alignments. Hence, fitness value of an optimal solution at the end of a given iteration is at least as high as the fitness value of the optimal alignment of the former iteration.

It can also be inferred from the graphs that the TS phase has shown same effect as that of GA for all the test cases. Initially, the fitness value has kept on increasing for some number of iterations. This is the consequence of the strategies followed during moves generation and evaluation phase of TS. It is explained in chapter four that when alignments are generated from a current alignment, only those alignments having better quality than the parent alignment are added to candidate move list. Hence, one or none among the candidates is taken as the current alignment based on content of tabu list and aspiration criterion set. This condition restricts the current alignment to be always at least as good as the former one. However, as can be seen from the graphs, the fitness value remains constant after some five values shown in the figure (which is about fifty iterations in average as the values on the figure are sampled at intervals of ten iterations). This implies that TS has produced an alignment which couldn't be improved further

in about fifty iterations. Therefore, as the search process proceeds from one solution to the other, it is guaranteed that the new solution cannot be of lower quality.

The graphs of Figure 5.1 also roughly show how much the TS improved the alignment quality from GA phase. It can be seen for all test cases that hybridization of the GA with TS has a positive effect. This effect is numerically quantified as given in table 5.3 based on the fitness values of the alignment attained by the end of GA phase and those values attained by the end of the hybrid system (i.e., at the end of the GA plus TS phase).

Table 5.3 Numerical quantification of effect of hybridization based on fitness value

| Test case | Fitness attained during GA phase | Final Fitness attained by the proposed system | Percentage of fitness Improvement due to hybridization |
|---|---|---|---|
| 1 | 2.2251 | 3.3943 | 52.55% |
| 2 | 1.7154 | 2.3549 | 37.28% |
| 3 | 1.2633 | 1.6659 | 31.87% |
| 4 | 1.2510 | 1.4550 | 16.31% |

In table 5.3, the percentage of fitness improvement is calculated by taking the difference between the final fitness attained and fitness attained by GA phase. Then the difference is expressed as percentage of the fitness attained by the GA phase. As can be seen from table 5.3, hybridizing GA with TS results in a significant improvement on the quality of the alignment. However, the percentage of improvement decreases with increase in number of sequences to be aligned. TS, therefore, enhances the performance of the system with a property that its achievement decreases with increasing number of sequences to be aligned.

In section 5.4.1 the possible causes for the observed decrement of fitness improvement as the number of sequences increase is discussed.

## 5.4 Performance Evaluation

As mentioned in the introduction section of this chapter, the performance of the hybrid system proposed is shown by means of comparing its result with the result of standard methods. For this

purpose, the test cases have been submitted electronically to EBI server for alignment using CLUSTALW and MAFFT. The sequences are then acted upon by the software packages and the optimal alignments obtained have been made available. Form the alignments by the benchmarks, the number of matching nucleotide bases, the number of columns of the alignment, the number of pairs of comparisons and the fitness values of the alignment have been computed and tabulated as shown in Table 5.4. The same parameters have been computed and tabulated for alignment by the proposed method in same table. The parameters used in the table such as matches comparisons and columns represent the number of matching pairs of nucleotide bases, the number of pairs of comparisons and the column length of the alignment respectively.

Table 5.4 Performance the proposed system as compared to the benchmarks

| TEST CASE | | ONE | TWO | THREE | FOUR |
|---|---|---|---|---|---|
| CLUSTALW | Matches | 353 | 1393 | 5979 | 23540 |
| | comparisons | 912 | 5100 | 24864 | 136320 |
| | Columns | 912 | 850 | 888 | 1136 |
| | Percentage of matches | 38.71 | 27.31 | 24.05 | 17.27 |
| MAFFT | Matches | 312 | 1226 | 4686 | 19829 |
| | comparisons | 917 | 4698 | 24136 | 116760 |
| | Columns | 917 | 783 | 862 | 973 |
| | Percentage of matches | 34.02 | 26.10 | 19.41 | 16.98 |
| PROPOSED SYSTEM | Matches | 314 | 1182 | 4533 | 16605 |
| | comparisons | 1022 | 6078 | 37408 | 176880 |
| | Columns | 1022 | 1013 | 1336 | 1474 |
| | Percentage of matches | 30.72 | 19.45 | 12.12 | 9.39 |

The number of pairs of comparisons (named comparisons in the table) is calculated from the number of sequences K and the number of columns N in the alignment according to equation 5.1.

$$Comparisions = N * \sum_{s=1}^{K-1} \mathrm{s} \quad (5.1)$$

The percentage of matches which reflects how much percent of the pairs of comparisons are matching pairs of nucleotide bases is, then, calculated using equation 5.2.

$$Percentage\ of\ matches = \frac{\text{number of matches}}{\text{Comparisions}} * 100 \quad (5.2)$$

The percentages matches obtained are used of evaluating the proposed system and are graphed and shown in Figure 5.3.

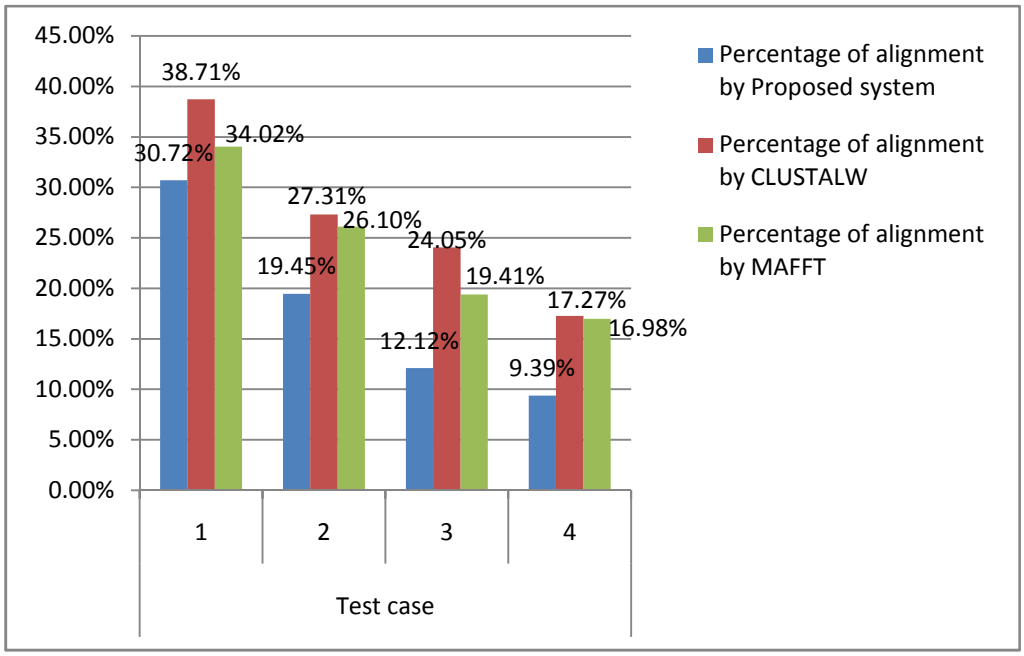



Figure 5.3 Proposed system vs benchmarks in percentage of matches

According to figure 5.3, 30.72% of the pairs are made matching pairs of nucleotides bases by the proposed system while this figure is 38.71% for CLUSTALW and 34.02% for MAFFT for test case 2. From the figure, it can be seen that the proposed system shows performs near to MAFFT for test case 1 where the percentage differs by 3.3. However, it performs worse for test case 3 where the percentage differs by 11.93 from CLUSTALW's. In general, the proposed system performs less than the two benchmarks for all the test cases. The possible causes due to which the proposed system performs less is discusses in next section.

### 5.4.1 Possible causes of less performance

In the first place, the number of columns in the final alignment plays a significant role in determining the percentage of matches. The number of possible comparisons made within the alignment is the function of the number of columns and number sequences as can be inferred from equation 5.1. Referring to table 5.4, it can be seen that the number of columns in the alignment by the proposed system is greater than the corresponding values for CLUSTALW and MAFFT for all the test cases. This results in much greater values of the number of comparisons. Therefore, even if the number of matches is comparable, the resulting percentage of matches differs greatly. For test case 1, for example, percentage of matches for the MAFFT (34.02%) is greater than the proposed system (30.72%) despite the fact that the number of matches of MAFFT (312) is less than that of the proposed system (314). Therefore, the higher number of columns in the final alignment by the proposed system is one of the causes for the system to perform less than the benchmarks.

Another cause is related to the requirement of the implementation scheme followed in this work. From table 5.3, it is observed that the effect of TS to enhance the performance of the proposed system decreases with increasing number of sequences to be aligned. It can be said that this is the effect of slicing the input sequences for the purpose of hardware synthesis. It has been declared in the previous chapter that the FPGA implementation needs the MATLAB code to conform to coding style acceptable by the synthesis tool, AccelDSP. According to the acceptable coding style, data entering and leaving the design must be partitioned into manageable slices [21]. In this work, the sequences to be aligned are partitioned in such a way that 64 nucleotides can be passed to the design function. Therefore only small fragment of the alignment having 64 nucleotides is processed per design function call.

As a result of slicing the input sequences same nucleotide bases that could be arranged in same columns might be separated to two different segments. This leads to condition which results in poor alignments at the boundary of the slices. Therefore, slicing the input sequences for hardware synthesis is another cause that made the system to perform less than CLUSTALW and MAFFT. Moreover, as the number of sequences increase, the characters around the boundary which might be poorly aligned increases. Hence, it can be expected that the quality of the alignment reduces with growing number of sequences as Table 5.3 witnesses. To quantify this

effect numerically, the system performance without slicing the input sequences is investigated and discussed next.

## 5.4.2 System performance without slicing

In this section, the result of the proposed system implemented by avoiding the concept of slicing is discussed. The proposed system is implemented in MATLAB as if there is no restriction on the amount of the sequence to be passed to the design function during the hardware implementation. Afterward the alignment obtained using this scenario is compared with the alignment obtained using the proposed system with slicing. Furthermore, it is compared with benchmarks.

### 5.4.2.1 Proposed system with and without slicing

For purpose of understanding the effect of slicing, the fitness values of the alignment obtained by without slicing the input sequences is tabulated along with the fitness value given in Table 5.3 (fitness by GA plus TS with slicing).

Table 5.5 Improvement in fitness value gained from non-slicing

| Test case | Fitness attained by proposed system without slicing | Fitness attained by proposed system with slicing | Improvement gained from non-slicing |
|---|---|---|---|
| 1 | 3.4945 | 3.3943 | 3.0% |
| 2 | 2.5513 | 2.3549 | 8.3% |
| 3 | 1.9400 | 1.6659 | 16.5% |
| 4 | 1.6900 | 1.4550 | 16.1% |

According to Table 5.5, it is can be shown that the percentage of improvement gained increases with increasing number of sequences aligned. That is to say, the alignment quality degrades with increasing number sequences. This conforms to the fact that the nucleotides that might be poorly aligned at the boundary increases with increase in number of sequences. Therefore, slicing the input sequences impose negative effect on the alignment quality which gets worse as the number of sequences increases.

### 5.4.2.2 Proposed system without slicing versus benchmarks

Using same approach as in section 5.4, the parameters used in Table 5.4 (matches, columns and comparisons) are determined for alignment by the proposed method, of course without slicing the input sequences in this case. The values along with those for the benchmarks (Table 5.4) are given in Table 5.6.

Table 5.6 Comparison between the benchmarks and the proposed system without slicing

| TEST CASE | | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| CLUSTALW | Matches | 353.00 | 1393.00 | 5979.00 | 23540.00 |
| | comparisons | 912.00 | 5100.00 | 24864.00 | 136320.00 |
| | Columns | 912.00 | 850.00 | 888.00 | 1136.00 |
| | percentage | 38.71 | 27.31 | 24.05 | 17.27 |
| MAFFT | Matches | 312.00 | 1226.00 | 4686.00 | 19829.00 |
| | comparisons | 917.00 | 4698.00 | 24136.00 | 116760.00 |
| | Columns | 917.00 | 783.00 | 862.00 | 973.00 |
| | percentage | 34.02 | 26.10 | 19.41 | 16.98 |
| PROPOSED SYSTEM | Matches | 321.00 | 1267.00 | 5357.00 | 21716.00 |
| | comparisons | 1005.00 | 5784.00 | 34664.00 | 170640.00 |
| | Columns | 1005.00 | 964.00 | 1238.00 | 1422.00 |
| | percentage | 31.94 | 21.91 | 15.45 | 12.73 |

In the table the number of comparisons and the percentage of matches are calculated using equation 5.1 and 5.2 respectively. Based on the percentage of matches, the system is evaluated as shown in Figure 5.4.

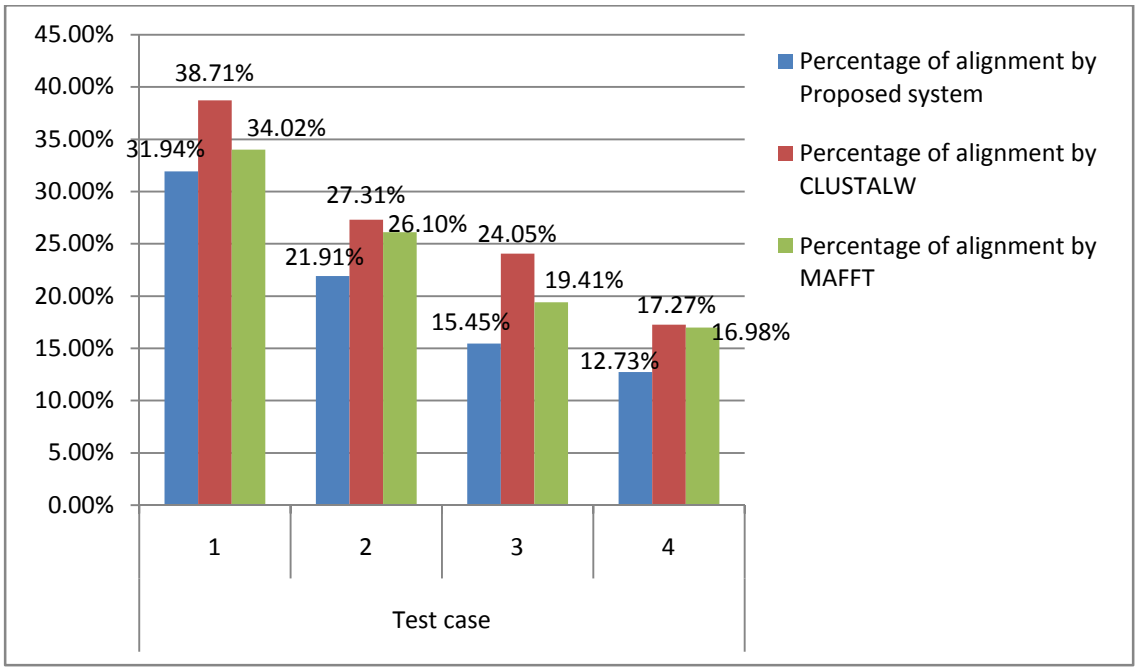



Figure 5.4 Proposed system (no slicing) vs benchmarks in percentage of matches

As can be seen from the column graphs of Figure 5.4 and 5.3, the percentage of matches attained with no slicing has shown an increment of 1.22, 2.46, 3.33 and 3.34 for test case 1, 2, 3 and 4 respectively. This again shows the negative effect of slicing which is worse for more number of sequences to be aligned.

In regard to comparison with the benchmarks, the percentage of matches for the proposed system is still less than those of the benchmarks. But maximum difference of percentage from CLUSTALW is now reduced from 11.9 to 8.6. The minimum difference of percentage between MAFFT and the proposed system which is 3.3 is now reduced to 2.1. Hence, the proposed system without slicing improves the percentage of matches within the alignment.

# Chapter 6

# Conclusions and Recommendations

## 6.1 Introduction

In this work, hybrid system to align multiple DNA sequences is designed. The designed system is implemented using MATLAB and VHDL (for the TS part). The system is finally compared with benchmark methods CLUSTALW and MAFFT. Furthermore, the MATLAB implementation is compared with the FPGA based implementation. Based on the results, the following conclusions and recommendations are made.

## 6.2 Conclusions

The quality of the optimal alignment during any iteration is always greater than the quality of optimal alignment of the previous iteration. Both GA and TS always improve the alignment quality as the optimization process proceeds. Moreover, TS phase brings about significant result in enhancing the performance of the proposed system. However, the alignment quality for design tailored for hardware implementation decreases with increasing the number of sequences to be aligned. The system performs less than both the benchmarks. It performs less with percentage of matches differing at most by 8.6 from CLUSTALW for 8 sequences. It also performs less with percentage of matches differing at most by 4.25 from MAFFT for 16 sequences.

FPGA based implementation of multiple sequence alignment results in significant improvement in speed over the software implementation. In this work, speed up of about 64 is gained by the hardware implementation. As long as the original MATLAB design is not changed, both implementations result in alignment of same quality.

## 6.3 Recommendations

As can be inferred from the effect of slicing imposes on the alignment quality, as the number of sequences to be aligned increases, the alignment quality decreases. Having a target FPGA with lower number of input output pins demand higher number of slices of the input sequences. Therefore, the target device for implementation should be selected so as to have number of pins as high as possible.

In addition, the number of gaps inserted to the sequences at the beginning of the GA phase needs to be examined carefully as it determines the number of columns in the final alignment. The number of columns in the final alignment, in turn, determines the number of possible pairs of comparisons needed to find out the percentage of matches and the fitness value.

Needles to mention, the performance of the hybrid system is the result of the chosen parameters and objective function set. In particular, mutation probability, scoring schemes for matches and mismatches, the gap penalty and tabu tenture might influence the performance of the system. Hence, proper selection of these parameters should be done.

# References

[1] http://ghr.nlm.nih.gov/SiteMap, *"Your Guide to understanding Genetic Conditions"*, Hand book of U.S. National Library of Medicine, USA

[2] Hung Dinh Nguyen, Ikuo Yoshihara, Kunihito Yamamori, Moritoshi Yasunaga, *"Aligning Multiple Protein Sequences by Parallel Hybrid Genetic Algorithm"*, M.Sc thesis, Graduate School of Engineering, Faculty of Engineering, Miyazaki University, Japan. 2002 IEEE

[3] Cédric Notredame and Desmond G. Higgins, *"SAGA sequence alignment by genetic algorithm"*, EMBL outstation, The European Bioinformatics Institute, Hinxton Hall, Hinxton, Cambridge CB10 1RQ, UK, 1996

[4] M. F. Omar, R. A. Salam, R. Abdullah, N. A. Rashid, *"Multiple Sequence Alignment Using Optimization Algorithms"*, International Journal of Information and Mathematical Sciences 1:2 2005

[5] Mohd. Faizal Omar, Rosalina Abdul Salam, Nuraini Abdul Rashid, Rosni Abdullah, "*Multiple Sequence Alignment Using Genetic Algorithm and Simulated Annealing*", School of Computer Science, Universiti Sains Malaysia, I1900 Penang Malaysia, 2004

[6] Layeb Abdesslem, Meshoul Soham, Batouche Mohamed, *"Multiple Sequence Alignment by Quantum Genetic Algorithm"*, University of Mentouri, Constantine, Algeria, 2006 IEEE

[7] Stefan Dydel and Piotr Bala, *"Large Scale Protein Sequence Alignment Using FPGA Reprogrammable Logic Devices"*, Faculty of Mathematics and Computer Science, N. Copernicus University, Chopina 12/8, 87-100 Torun, Poland, 2004.

[8] Warattapop Chainate, Peeraya Thapatsuwan, and Pupong Pongcharoen, *"A New Heuristic for Improving the Performance of Genetic Algorithm"*, Naresuan University, Phitsanulok, Thailand, 2007

[9] Tariq Riaz, Yi Wang, Kuo-Bin Li, *"Multiple Sequence Alignment Using Tabu Search"*, M.Sc thesis, Bioinformatics Institute, 30 Biopolis Street, Singapore, 2004

[10] R.Thamilselvan and Dr.P.Balasubramanie, *"A Genetic Algorithm with a Tabu Search for Traveling Salesman Problem",* Kongu Engineering College/Computer Science and Engineering, Erode, India, 2009

[11] Xunying Zhang, Chen Shi, and Fei Hui, *"FPGA-Based Genetic Algorithm Kernel Design"*, Xi'an Institute of Microelectronics Technology, 710054, Xi'an, Shaanxi, China, 2007.

[12] Tu Lei, Zhu Ming-cheng and Wang Jing-xia, *"The Hardware Implementation of a Genetic Algorithm Model with FPGA"*, M.Sc thesis, Shenzhen University and Shenzhen Polytechnic, Shenzhen 518060, P.R.C, 2002 IEEE

[13] Stephen D. Scott, Sharad Seth, and Ashok Samal, "A Synthesizable VHDL Coding of a Genetic Algorithm", 1999 by CRC Press LLC, Canada, 1999.

[14] Thomas Weise, "Global Optimization Algorithms – Theory and Application", Version: 2009-06-26 ebook, 2009.

[15] Nardos Asnake, *"Incrementally Autonomous Light Weight Agent Architectures for Optimization Task"*, Addis Ababa University School of Graduate Studies, Department of Electrical and Computer Engineering March, 2005

[16] Michael Negnevitsky, *"Artificial Intelligence -A Guide to Intelligent Systems",* Second Edition, Pearson Education Limited, England London, 2005

[17] Fred Glover, Manuel Laguna, *"Principles of Tabu Search",* Leeds School of Business, University of Colorado, USA, 2007.

[18] Panos M. Pardalos, L.Pitsoulis1, T. Mavridou, and Mauricio G.C. Resende*, "Parallel Search for Combinatorial Optimization: Genetic Algorithms, Simulated Annealing, Tabu Search and GRASP"*, Center for Applied Optimization and Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL 32611-6595 USA, 1995

[19] Pham, D.T. and Karaboga D., *"Intelligent Optimization Techniques – Genetic Algorithms, Tabu Search, Simulated Annealing and Neural Networks",* Springer-Verlag, London, 2000

[20] C. Gondro and B.P. Kinghorn, "*A simple genetic algorithm for multiple sequence alignment*", The Institute for Genetics and Bioinformatics (TIGB), University of New England, Armidale, Australia, Oct 05, 2007

[21] http://www.xilinx.com, "*AccelDSP Synthesis Tool - user guide"*, Release 10.1 March, 2008

[22] Randy L. Haupt, Sue Ellen Haupt, *"Practical Genetic Algorithms"*, Second Edition, A john Wiley & sons Inc., publication, USA, 1998

[23] L. A. Anbarasu, P. Narayanasamy and V. Sundararajan, *"Multiple Sequence Alignment Using Parallel Genetic Algorithm"*, Anna University, Chennai 600 025, Center for Development of Advanced Computing, Pune 411 007, 1999

[24] Kosmas Karadimitriou, Donald H. Kraft, *"Genetic Algorithms and Multiple Sequence Alignment Problem in Biology"*, M.Sc thesis, Department of Computer Science, Louisiana State University, USA, 1996

[25] http://www.ebi.ac.uk, "European Bioinformatics Institute", United Kingdom, 2011

[26] Tim Oliver, Bertil Schmidt, Darran Nathan, Ralf Clemens and Douglas Maskell, "*Multiple Sequence Alignment on an FPGA*", School of Computer Engineering, Nanyang Technological University, Singapore, 2005

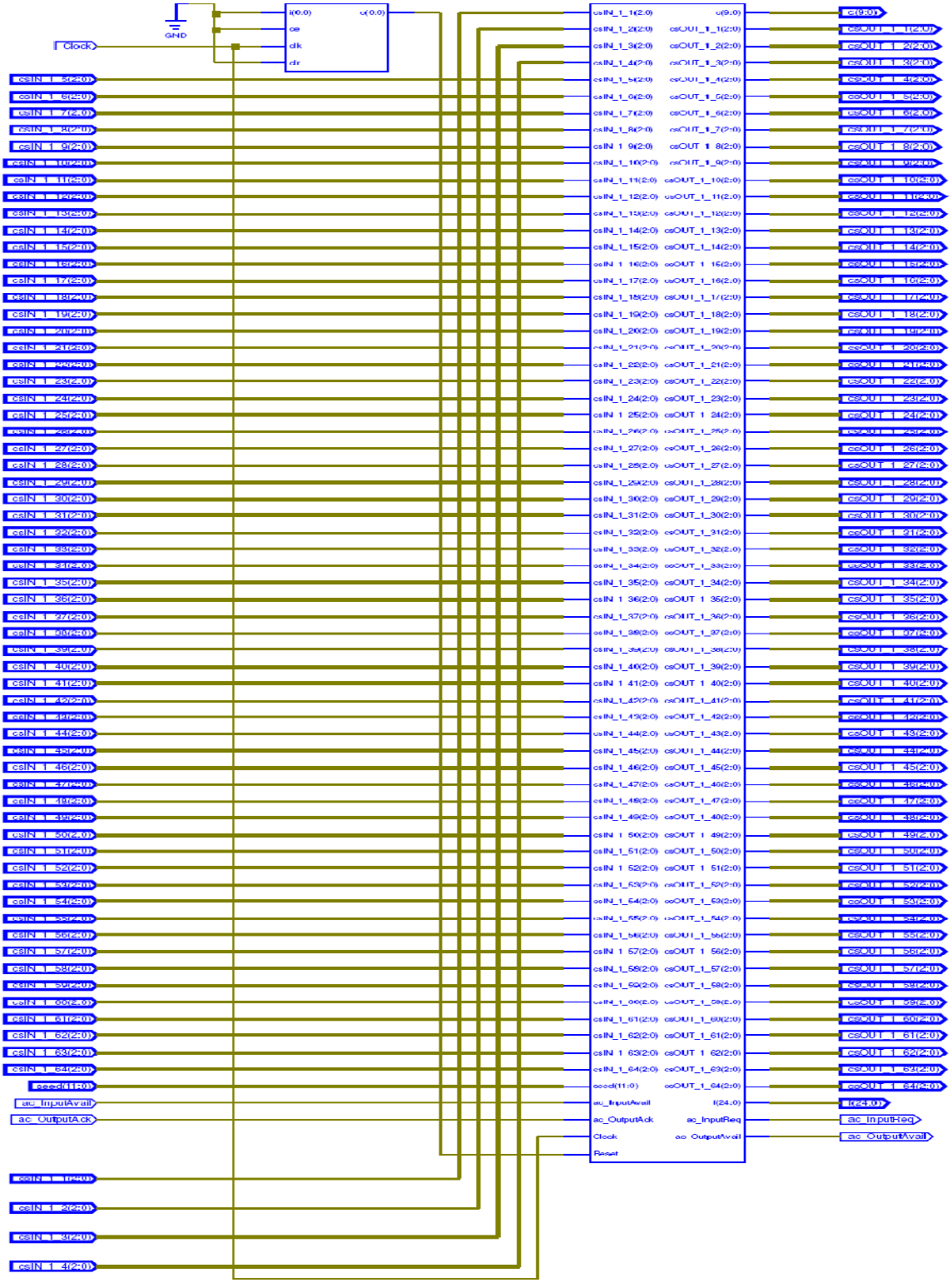
# Appendix A: Test case 1 Sequences

```
>SQ1: ID  HQ395007; MAM 682 BP; 145 A; 197 C; 200 G; 140 T; 0 OTHER;
GGCCGAGGGACAGCTGACACTGCAGCAGTTTGCGCAGTCCACGGAGATGCTGAAGCGCGT
GGTGCAGGAGCACCTACCGCTGATGAGCGAAGCGGGCGCCGGCCTGCCCGACATGGAGGC
TGTGGCGGGTGCCGAAGCACTCAATGGCCAGTCCGACTTCCCCTACCTGGGCGCCTTTCC
CATCAACCCAGGCCCCTTCATCATGACCCCCGCGGGCGTGTTCCTGGCTGAGAGCGCGCT
GCACATGGCCGGCCTGGCTGAATACCCCATGCGGGGAGAGCTGGCCTCCGCCATCAGCTC
CGGCAAGAAGAAGCGGAAACGCTGCGGCATGTGTGCGCCCTGCCGGCGGCGCATCAACTG
CGAGCAGTGCAGCAGTTGTAGGAACCGAAAGACTGGCCATCAGATTTGCAAATTCAGAAA
ATGTGAGGAACTCAAAAAGAAGCCTTCCGCTGCTCTGGAGAAGGTGATGCTTCCGACGGG
AGCCGCCTTCCGGTGGTTTCAGTGACGGCGGCGGGAACCCAAAGCTGCCCTCTCTGTGCA
ATGTCACTGCTCGTGTGGTCTCCGGCAAGGGATTCGGGCGAAGACAAACGGATGCACCCG
TCTTTAGAACCAAAAATATTCTCTCACAGATTTCATTCCTGTTTTTATATATATATTTTT
TGTTGTCGTTTTAACATCTCCA
>SQ2: ID  HQ395009; MAM 903 BP; 155 A; 280 C; 260 G; 208 T; 0 OTHER;
CTGGAATTCTACCGCTATTACCGGCAGCTCAAGCTCGAAGGGCCCGGGGAGCAGGAGACC
AAGCTGGACCTCATTAGTAAAGGAGAGGAAGCAAAACAGGCCAAGAGGAGACCCGATTCT
CGGCCCCCAGCTCTCAGCCCACCAAGGAAAGCCACTCTGTCCGCACTATCCTCAAAAGTA
TCTTGGTCCCGGCTTTCTCCGTCTGCTTCGTCTTCACCATCACCATTGGGATATTTCCTG
CCGTGACAGCTGAGGTCGAGTCCAGCATTGCGGGCACCAGCGCCTGGAAGGCCTACTTCA
TTCCTGTGTCCTGTTTCTTGACTTTCAATGTCTTTGACTGGCTGGGCCGGAGCCTCACAG
CCATCACCATGTGGCCTGGGAAGGACAGCTACTGGCTCCCGAGCCTGGTGCTGGCCCGGC
TGGCCTTCGTGCCCCTGCTGCTGCTGTGTAACGTCCAGCCCCGCCGCAACCTGCCTGTGA
TCTTTGAGCACGATGCCTGGTTCATCATCTTCATGGCTGCCTTCGCCTTCTCCAACGGCT
ACCTTGCCAGTCTCTGCATGTGCTTCGGGCCCAAGAAAGTGAAGCCGGCTGAGGCGGAGA
CAGCTGGAACCATCATGGCCTTCTTTCTGTCTCTGGGCCTGGCCCTGGGGGCCGTCTTCT
CCTTCCTGTTCCGGGCAATCGTGTGACCGCGGATGGACAGCAGGATGGCACCAGCTGCCT
GCTCCTGCCTTCCCCTTTGGGGGTGGGCAGGGGTCCTCTGGAGGTTTTCCTGTCTAGCCA
TCTTCTGCTTGCTCATGGCCTGTGCTGGGCCCGGCATCCAGGCCCCGAGGAGAGAGCTCT
GTGGATGAACATGGGGACACTGTGGGCTGCAGGGTCAGAGTCAAGGGAGGGGACACAGCC
TCT
```

# Appendix B: Aligned sequences of Test case 1

```
//sequence1 : ID: HQ395007
G-G-CCGA--GGGA-C-AGC-TGA-CA-CTGC-AGC--AGTTTG---C-GCAGTCCAC-GGAGA-T--G
CTGAAGCGCGTGGT-G--CAGG--AG---C--ACCTAC--CG-CTGATGAG-CGAAGCGGG-C--G-CC
--G-G----C--C-TGCC--CGACATG---GAGGCTGTGG---C-GG-GT---G-CC-GAAGCACTC-A
A----T---GGCC-AGTCCG---ACTTCCCC-T-A-C-CT-GGGCGCC-T-TT-CC----CA-TC-AA-
CCCA-GGCCC--CTTCAT-CATG-ACC--------------CCCGCGG--------GCGTGTTCCTGGC
-TGAGAGC--G-C-GCT-GCA---CATGG-CC-GGCCT--GGCT-GAA-T-ACCCC-A--TGC--GGGG
AGAG-CT-GG-----C--CT-CC--GC---C-A-TC-AGCTCCGGCAAGAAGAAGCGG---AAAC-G-C
-T-GC-GGCATGTG-TGC-GCC-CTG-C-CG--GCGGCGC-ATC--AACT-GCGAGC-AGTGCAG-CAG
TTGT-A-G-G-AACCGA-AAGACTGGC---CAT-C--A-GA-TTTG-CAAAT-T-C-AG-AA--AATGT
GAGGA---A----C-TC--AAAAA---G--A-A---GCCTTCCGCT--GCTC-TG-G-AGAAGGTG---
-A----------------T-----------GCTT--CCGACGGGA-GCCG-CCTT-CCGGT--GGTTTC
-AGT--GAC----GGCGGCGGGA-ACCCAA--A-GCTGCCCTCTCTG-TGC-AAT-G-TC-ACTG-CTC
GTGTG--G-T---C-----TCC-G--GCAA-GGG---AT-TCGGG---CG--------A-AGA-CA-AA
CGG--ATGCACCCGTCTTT-AGA----ACCAAAAAT-ATTCTC--T-CAC--AGATTTCATT-C-C--T
--G-T--TTTTATATATATAT-TTTT----TGTTGTCGTTTT-AACATCTCC---A
//sequence2: ID: HQ395009
CTGGAA-TTCTACCGCTATTACCGGCAGCT-CAAGCTCGA-AGGGCCCGGG-GA--GCAGGAGACCAAG
CTGGACCTC--ATTAGTAAAGGAGAGGAAGCAAAACAGGCC-AA-GAGGAGACCC-G-A-TTCTCGGCC
CCCAGCTCTCAGCCCACCAAGGAAA-GCCAC-T-CTGTCCGCACTATCCTCAAAAGTATCTTG-GTCCC
GGCTTTCTCCGTCTGCTTCGTCT--TCAC-CATCACCATTGGGAT-ATTTCCTGCCGTGACAG-CTGAG
GTCGAG-TCCAGC--ATTGCGGGCACCAGCGCCTGGAAGGCCT--A-CTTCATTCCT-GTGT-CCT-GT
TT-----CTTGACTT-TCA-ATGTCTTTGACTGG-C-TGGG-CCGGAGCCT-CACAGCCAT-CAC-CAT
GTGGCCTGGGAAGGACAGCTA-CTGGCTCCCGAGCCT-G---------G--TG-CTGGCCCG-GCTGGC
CTTCGTGCC-CCTGCTGCTG-CTGTGTAACGTCCAGCCCCG-CCGCAACCTGCCTGTGA-TCT-TTGAG
-CACGATGCCTGGTTCATCATCTTCA-TGGC-TGCCTTCGCC-TTCTCCAA-CGGCTACCTTGCCA-GT
--CTCTGCATGTGCTTCGG--GCCCAAGAAAGTGAAG-------C-CGGCTGAGGCGGAG-AC-AGCTG
GAACCATCATGGCCTTCTTTCTGTCTCTGGGCCTGG-C-CCTGGGGGCCGTCTTCTCCT-TCCTGTT-C
CGGGCAATCGTGTGACCGCGG-ATGGAC-AGCAGGATG-GC-ACCAGCTGCCTGCTCCT-GCCT-TCCC
CTTTGGGGGTGGGCAGGGGTCCTCTGGAGGTTTTCCTGTCT-AGCCATCTTCTGCTTGCTC-ATG-GC-
CTGTGCTGGGCCCGGCATCCAGGCCCCGA--GGAGAGAG-CTCTGTGGATGAACATG-GG-GACACTGT
GGGCTGCAG-G----GT--C-AGAGTCAAGGG-AG--GG-GACA-CA--G-CCTCT
```

# Appendix C: Synthesized RTL Schematic Block of TS



# Appendix D: MATLAB Source Code

```
%Designed system for Multiple DNA sequence alignment
%script_msa
t1_msa = rem(now, 1);
popSize = 200;
maxIteration = 2000;
maxF = 0;
maxM = 0;
averageFitness = 0;
iteration = 0;
mutationProb = 0.01;
scaling_factor = 1.6;
stable = 0;
gaTrace = 0;
refinment_needed = false;
sequences = textread('initialSeq8.txt', '%s');   %cell array
numOfSeq = numel(sequences);
initialSeed = randint(1,1, [1, 100]);
seed = initialSeed;
cl_matches = 5979;
%integer representation of the sequences
for i = 1 : numOfSeq
    seqq{i} = double(sequences{i});
    seq{i} = intRepresentation(seqq{i});
    length(i) = size(sequences{i}, 2);
end
maxLength = max(length);
unscaledCol = maxLength * scaling_factor;
col = round(unscaledCol);
%formation of initial population
for n = 1 : popSize
   parent{n} = popInitialization(seq, length, col, numOfSeq);
   parent{n} = clean(parent{n});
end
f_steps = 0;
while(iteration <= maxIteration && stable == 0)
    % Evaluating Fitness of the individuals
     for n = 1 : popSize
        numOfRow = size(parent{n}, 1);
        numOfCol = size(parent{n}, 2);
        [fitness(n), matches(n)] = fitnessCalculation(parent{n}, numOfRow,
numOfCol);
     end
     prevF = max(fitness);
     prevM = max(matches);
     [sortedFitness, findex] = sort(fitness, 'descend');
     [sortedmatches, mindex] = sort(matches, 'descend');
    % Selection of parents for mating
     for n = 1 : popSize/4
         selectedParent{n} = parent{findex(n)};
         usedIndices(n) = findex(n);
     end
     n = 1;
     indx = 0;
```

```
      offset = popSize/4;
      while(indx < popSize/4)
          if(IndexUsed(mindex(n), usedIndices) == false)
              indx = indx +1;
              selectedParent{indx + offset} = parent{mindex(n)};
              additionalIndices(indx) = mindex(n);
              n = n+1;
          else
              n = n+1;
          end
      end
      for n = 1 : popSize/2
          selected(n) = false;
      end
      n = 1;
      ptr = 1;
      while(ptr <= popSize/2)
          num = randint(1,1,[1, popSize/2]);
          if(selected(num) ==0)
              selIndex(ptr) = num;
              selected(num) = true;
              ptr = ptr +1;
          end
      end
     % cross over
     for n = 1 : 2: popSize/2
         [child1, child2] = crossOver(selectedParent{selIndex(n)},
selectedParent{selIndex(n+1)});
          child{n} = clean(child1);
          child{n+1} = clean(child2);
     end
     numOfMutant = round(popSize * mutationProb);
      if(numOfMutant >= 1)
         for m = 1 : numOfMutant
             indexOfMutant = randint(1,1, [1, popSize/2]);
             numOfCol = size(child{indexOfMutant},2);
             randRow = randint(1,1, [1, numOfSeq]);
             mutant = child{indexOfMutant};
             child{indexOfMutant} = mutation(mutant, randRow);
         end
      end
      % Forming new generation
      offset = popSize/2;
      for n = 1 : popSize/2
          parent{n} = selectedParent{n};
          parent{n + offset} = child{n};
      end
      clear fitness;
      clear matches;
      for n = 1 : popSize
          numOfRow = size(parent{n}, 1);
          numOfCol = size(parent{n}, 2);
          [fitness(n), matches(n)] = fitnessCalculation(parent{n}, numOfRow,
numOfCol);
      end
      iteration = iteration +1;
      maxF = max(fitness);
```

```
maxM = max(matches);
currF = maxF;
currM = maxM;
diffOfF = currF - prevF;
diffOfM = currM - prevM;
[sortedFitness, findex] = sort(fitness, 'descend');
[sortedMatches, mindex] = sort(matches, 'descend');
iterationCount = iteration;
if(iteration ==1 || rem(iteration, 10) == 0)
    f_steps = f_steps +1;
    f1 = sortedFitness(1)
    f_ga(f_steps) = f1;
end
if(diffOfF < 0.0001 && diffOfM ==0)
    gaTrace = gaTrace +1;
else
    gaTrace = 0;
end
if(gaTrace < 20 && iteration < maxIteration)
    stable = 0;
    for n = 1 : popSize
        parent{n} = clean(parent{n});
    end
else
    if(currM > cl_matches)
        stable = 1;
        disp('Alignment quality attained with GA only');
        gaSolution = parent{mindex(1)};
        sq_GA = gaSolution;
        refinment_needed = false;
    else
        disp('proceeding to TS');
        stable = 1;
        refinment_needed = true;
        initialAlignment = parent{mindex(1)};
        saveSequencesFromGA(initialAlignment);


        for a = 1 : 8
        ss = initialAlignment;
        colCurrent = size(ss, 2);
        rowCurrent = size(ss, 1);
        total_elements = colCurrent * rowCurrent;
        elements_per_slice = 64;
        num_slice = floor(total_elements/elements_per_slice);
        cps = elements_per_slice/rowCurrent;   %column per slice = 16
        rowVector = zeros(1, 64);
       for r = 1 : num_slice
          indx = r*cps -cps;
          for i = 1 : numOfSeq
          offset = (i-1) * cps;
              for j = 1 : cps
                   rowVector(offset +j) = ss(i, indx+j);
              end
          end
          input_Array(r, 1:64) = rowVector;
       end
```

```
            if(rem(total_elements, elements_per_slice) >= 1)
               extra_slice = 1;
               extra =zeros(1, 64);
               r = num_slice +1;
               num_slice = num_slice +1;
               indx1 = r*cps - cps;
               indx2 = size(ss, 2);
               for i = 1 : numOfSeq
                   offset = (i-1) * cps;
                   for j = 1 : (indx2 - indx1)
                       extra(offset +j) = ss(i, indx1+j);
                   end
               end
               input_Array(r, 1:64) = extra;
            else
               extra_slice = 0;
            end
            %streaming loop
            seed_TS(a) = randint(1,1,[1, 100]);
            inputSeq = zeros(1, 64);
            outputSeq = zeros(1, 64);
            t1_ts = rem(now, 1);
            for r = 1 : num_slice
                seed = seed_TS(a);
                inputSeq = input_Array(r, :);
                [outputSeq, fitness, comp] = tabuSearch(inputSeq, seed,
numOfSeq);
                output_Array(r, :) =  outputSeq;
                fitness_ts(r, :) = fitness;
                comparision(r, :) = comp;
            end
            t2_ts = rem(now, 1);
            for k = 1 : size(fitness_ts(1, :), 2);
                  non_normalized_f(k) = 0;
                  c_sum(k) = 0;
                  for r = 1 : num_slice
                     non_normalized_f(k) = non_normalized_f(k) +
fitness_ts(r, k) * comparision(r, k);
                     c_sum(k) = c_sum(k) + comparision(r, k);
                  end
                  normalized_TS_fitness(k) = non_normalized_f(k)/c_sum(k);
            end
            fit_ts = normalized_TS_fitness;
            if(a==1)
                f_ts = fit_ts;
            end
            coll = cps * num_slice;
            sqq = zeros(rowCurrent, coll);
            for r = 1 : num_slice
                indx = r*cps -cps;
                for i = 1 : numOfSeq
                    offset = (i -1)*cps;
                    for j = 1 : cps
                        sqq(i, indx+j) = output_Array(r, offset+j);
                    end
                end
            end
```

```
            numColumns = size(sqq, 2);
            counter = 0;
            for k = 1 : numColumns
                if(sqq(:, k) ~= 0)
                    counter = counter + 1;
                    sq_TS(:, counter) = sqq(:, k);
                end
            end
            numColumns = size(sq_TS, 2);
            numRows = size(sq_TS, 1);
            for j = 1 : cps
                if(j <= cps/2)
                    initialAlignment(:, j) = sq_TS(:, j);
                else
                    initialAlignment(:, j) = 5;
                end
            end
            offset = cps/2;
            for j = cps/2 +1 : numColumns
                initialAlignment(:, j + offset) = sq_TS(:, j);
            end
            end %end of repeatition for loop


        end
    end
end %% end of outer while loop
t2_msa = rem(now, 1);
disp('Best Alignment: ');
if(refinment_needed == true)
        for i = 1 : numOfSeq
            for j = 1 : numColumns
                if (sq_TS(i,j) == 1)
                    seq_msa(i,j) = double('A');
                elseif(sq_TS(i,j) == 2)
                    seq_msa(i,j) = double('C');
                elseif(sq_TS(i,j) == 3)
                    seq_msa(i,j) = double('G');
                elseif(sq_TS(i,j) == 4)
                    seq_msa(i,j) = double('T');
                elseif(sq_TS(i,j) == 5)
                    seq_msa(i,j) = double('-');
                else
                    seq_msa(i,j) = double(' ');
                end
            end
        end
        final = char(seq_msa);
else
    numColumns = size(sq_GA, 2);
    numRows = size(sq_GA, 1);
    for i = 1 : numOfSeq
       for j = 1 : numColumns
            if (sq_GA(i,j) == 1)
                seq_msa(i,j) = double('A');
            elseif(sq_GA(i,j) == 2)
                seq_msa(i,j) = double('C');
```

```
            elseif(sq_GA(i,j) == 3)
                seq_msa(i,j) = double('G');
            elseif(sq_GA(i,j) == 4)
                seq_msa(i,j) = double('T');
            elseif(sq_GA(i,j) == 5)
                seq_msa(i,j) = double('-');
            else
                seq_msa(i,j) = double(' ');
            end
        end
     end
     final = char(seq_msa);
  end

  %Reporting the result
    [matches, mismatches, basegaps, gapGaps, fitnessFinal] =
measureOfSimilarity(final, numRows, numColumns);
    a = fopen('alignment.txt', 'w');
    for i = 1 : numOfSeq
        fprintf(a, final(i, :));
        fprintf(a, '\n');
    end
    status = fclose(a);
    msa_ExecutionTime = (t2_msa-t1_msa) * 24 *60 *60;
    ts_time = (t2_ts-t1_ts) * 24 *60 *60;
    disp('Total Execution Time in seconds:');
    disp(msa_ExecutionTime);
    disp('TS Execution Time in seconds:');
    disp(ts_time);
    disp('Fitness Value of the final alignment: ');
    disp(fitnessFinal);
    disp('Number of matches in the final alignment: ');
    disp(matches);
    disp('Initial Seed: ');
    disp(initialSeed);
    plotFitness(f_ga, f_ts);
    %end
    a = fopen('alignmentDetails.txt', 'w');
    fprintf(a, 'FITNESS GA');
    fprintf(a, '\n');
    for k = 1 : size(f_ga, 2)
        fprintf(a, '%d', f_ga(k));
        fprintf(a, '\n');
    end
    fprintf(a, 'FITNESS TS');
    fprintf(a, '\n');
    for k = 1 : size(f_ts, 2)
        fprintf(a, '%d', f_ts(k));
        fprintf(a, '\n');
    end
    fprintf(a, '\n');
    fprintf(a, 'Final Fitness: ');
    fprintf(a, '%d', fitnessFinal);
    fprintf(a, '\n');
    fprintf(a, 'Matches: ');
    fprintf(a, '%d', matches);
    fprintf(a, '\n');
```

```
    fprintf(a, 'Mismatches: ');
    fprintf(a, '%d', mismatches);
    fprintf(a, '\n');
    fprintf(a, 'TS Execution time: ');
    fprintf(a, '%d', ts_time);
    fprintf(a, '\n');
    fprintf(a, 'TS seed: ');
    fprintf(a, '%d', seed_TS);
    fprintf(a, '\n');
    fprintf(a, 'GA_Iteration: ');
    fprintf(a, '%d', iterationCount);
    status = fclose(a);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function seq = intRepresentation(sequence)
    for j = 1 : size(sequence, 2);
        if(sequence(j) == 65)
            seq(j) = 1;
        elseif(sequence(j) == 67)
            seq(j) = 2;
        elseif(sequence(j) == 71)
            seq(j) = 3;
        else
            seq(j) = 4;
        end
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function par = popInitialization(seq, length, col, numOfSeq)
    for i = 1 : numOfSeq
        gaps(i) = col - length(i);
        row = insetGaps(seq{i}, gaps(i), length(i), col);
        par(i, :) = row;
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function sq = insetGaps(seq, numGaps, length, col)
 row = zeros(1, col);
 ptr = 1;
 while(ptr <= numGaps)
        randomPos = randint(1,1,[1, col]);
        if(row(randomPos) ~= 5)
            row(randomPos) = 5;
            ptr = ptr +1;
        end
 end
k = 1;
for j = 1 : col
   if k <= length;
       if(row(j) == 0)
           row(j) = seq(k);  % insert the nucleotides (A T G C)
           k = k + 1;
       end
    end
end
sq = row;
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function cleaned = clean(individual)
    i =1;
    col = size(individual,2);
         while(i <= col)
              cg = 1;
              for j = 1 : size(individual,1)
                  if(individual(j, i) == 5)
                       cg = 0;
                  else
                      cg = 1;
                      break;
                  end
               end
               if cg == 0
                   col = col - 1;
                   individual(:, i) = [];
               end
              i = i + 1;
         end
      cleaned = individual;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [fitness, match] = fitnessCalculation(s, row, col)
        value = 0;
        counter = 0;
        match = 0;
        for j = 1 : row -1
            for k = j+1 : row
                for m = 1 : col
                    if((s(j,m) == 5) && (s(k,m) == 5))
                         %value = value - 0;
                         counter = counter + 1;
                    elseif(s(j,m) == s(k,m))
                         value = value + 8;
                         match = match +1;
                         counter = counter + 1;
                    elseif((s(j,m) == 5) || (s(k,m)== 5))
                         value = value + 1;
                         counter = counter + 1;
                    elseif(s(j,m) ~= s(k,m))
                         value = value  + 2;
                         counter = counter + 1;
                    end
                end
            end
        end
 fitness = value/counter;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [child1 child2]= crossOver(par1,par2)
    col = size(par1,2);
    row = size(par1,1);
    randLoc = randint(1,1,[2,col-2]);
    child1_front = par1(:, 1 : randLoc);
    child2_tail = par1(:, randLoc+1 : end);
```

```
   for i = 1 : row
        nucleotides(i) = 0;
        for j = 1 : randLoc
            if (par1(i, j) ~= 5)
                 nucleotides(i) = nucleotides(i) +1;  % nucleotides in each
row of the fragment1
            end
        end
    end

   for i = 1 : row
        index(i) = 0;
        counter = 0;
        for j = 1 : size(par2, 2)  % till row numbers of p2
            if (par2(i, j) ~= 5)
                counter = counter + 1;
                if(counter == nucleotides(i))
                   index(i) = j;
                   break;
                end
            end
        end
   end

   % Forming child1
   for i = 1 : row
       for j= min(index)+1 : size(par2, 2)
           if(j <= index(i))
               child1_tail(i, j - min(index)) = 5;
           else
               child1_tail(i, j - min(index)) = par2(i, j);
           end
       end
   end
   child1 = [child1_front child1_tail];

   %Forming child2
   for i = 1 : row
       for j = 1 : max(index);
           if(j <= index(i))
               child2_front(i, j) = par2(i, j);
           else
               child2_front(i, j) = 5;
           end
       end
   end
   child2 = [child2_front child2_tail];
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function mutant = mutation(Offspring, randRow)
  col = size(Offspring, 2);
  ctr = 0;
  for j = 1 : col
      if(Offspring(randRow, j) == 5)
          ctr = ctr +1;
          gapPos(ctr) = j;
```

```
      end
  end
  if(ctr > 0)
    index = randint(1,1, [1, ctr]);
    randCol = gapPos(index);
    if (randCol == 1)
        Offspring(randRow, randCol) = Offspring(randRow, randCol +1);
        Offspring(randRow, randCol +1) = 5;
    elseif(randCol == col)
        Offspring(randRow, randCol) = Offspring(randRow, randCol -1);
        Offspring(randRow, randCol -1) = 5;
    else
        if(Offspring(randRow, randCol -1) ~= 5)
            Offspring(randRow, randCol) = Offspring(randRow, randCol -1);
            Offspring(randRow, randCol -1) = 5;
        else
            Offspring(randRow, randCol) = Offspring(randRow, randCol +1);
            Offspring(randRow, randCol +1) = 5;
        end
    end
  end
  mutant = Offspring;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function saveSequencesFromGA(ss)
    numColumns = size(ss, 2);
    numOfSeq = size(ss, 1);
    for i = 1 : numOfSeq
        for j = 1 : numColumns
            if (ss(i,j) == 1)
                seq_msa(i,j) = double('A');
            elseif(ss(i,j) == 2)
                seq_msa(i,j) = double('C');
            elseif(ss(i,j) == 3)
                seq_msa(i,j) = double('G');
            elseif(ss(i,j) == 4)
                seq_msa(i,j) = double('T');
            elseif(ss(i,j) == 5)
                seq_msa(i,j) = double('-');
            else
                seq_msa(i,j) = double(' ');
            end
        end
    end
    final = char(seq_msa);
     a = fopen('alignmentFromGA.txt', 'w');
     for i = 1 : numOfSeq
         fprintf(a, final(i, :));
         fprintf(a, '\n');
     end
     status = fclose(a);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [csOUT, f, c] = tabuSearch(csIN, seed, row)
  CS = csIN;
  col = size(CS, 2)/row;
```

```
  maxIteration = 500;
  tabu_tenture = 10; % Iterations after which a Move in TL can be applied.
  TL(1, :) = CS;
  TT(1) = tabu_tenture - 1;
  num_TL = 1;
  maxFitness = 0;    % used as aspiration criterion
  iteration = 1;
  f_steps = 0;
  while(iteration <= maxIteration)
       num_ML = 0;
       col_limit = 0;
       for j = 1 : col
           if(CS(j) ~= 0)
               col_limit = col_limit +1;
           end
       end
       [f_threshold, comp, m_threshold] = TS_fitnessCalculation(CS, row,
col_limit);
        %SSM generation and evaluation
        for i = 1: row
            [SS_move, seed] = TS_SSM(CS, i, seed, row);
            col_limit = 0;
           for j = 1 : col
               if(SS_move(j) ~= 0)
                   col_limit = col_limit +1;
               end
           end
            [fitness(i), comp, matches(i)] = TS_fitnessCalculation(SS_move,
row, col_limit);
             if(matches(i) > m_threshold)
                 num_ML = num_ML +1;
                 ML(num_ML, :)  = SS_move;
                 fitness_ML(num_ML) = fitness(i);
                 Matches_ML(num_ML) = matches(i);
             end
        end
        clear fitness;
        %BSM generation and evaluation
        for i = 1: row -1
            [BS_move, seed] = TS_BSM(CS, i, seed, row);
             col_limit = 0;
              for j = 1 : col
                   if(BS_move(j) ~= 0)
                       col_limit = col_limit +1;
                   end
             end
             [fitness(i), comp, matches(i)] = TS_fitnessCalculation(BS_move,
row, col_limit);
             if(matches(i) > m_threshold)
                 num_ML = num_ML +1;
                 ML(num_ML, :) = BS_move;
                 fitness_ML(num_ML) = fitness(i);
                 Matches_ML(num_ML) = matches(i);
             end
        end
        clear fitness;
        if(num_ML > 0)
```

```
            %fitnessVals = fitness_ML;
            matchValues = Matches_ML;
            %[sortedFitnessVals, index] = sort(fitnessVals);
            [sortedMatches, index] = sort(matchValues, 'descend');
            %indexOfBest = index(size(fitnessVals, 2));
            indexOfBest = index(1);
            maxFitness = fitness_ML(indexOfBest);  % For Aspiration Criteria
            maxMatches = Matches_ML(indexOfBest);
        else
            maxFitness = f_threshold;
            maxMatches = m_threshold;
        end

        clear fitnessVals;
        % Selecting best move and apply
        if(num_TL > 0)
            num = 1;
            moveApplied = false;
            while(num <= num_ML && moveApplied ==false)
                available = false;
                for j = 1 : num_TL
                    ctr = 0;
                    for q = 1 : 64
                        if(ML(index(num), q) == TL(j, q)) %checking its
availability in TL
                            ctr = ctr +1;
                        end
                    end
                    if(ctr == 64)
                        available = true;
                        pt = j;
                    end
                end
                if(available == true)
                    if ((fitness_ML(num) > maxFitness)||(TT(pt) < 1))
                        %disp('Available but applied');
                        CS = ML(index(num), :);
                        TL(num_TL+1, :) = ML(index(num), :);
                        num_TL = num_TL +1;
                        TT(num_TL) = tabu_tenture;
                        moveApplied = true;
                    end
                else
                    %disp('Applied since not available in TL');
                    CS = ML(index(num), :);
                    TL(num_TL+1, :) = ML(index(num), :);
                    num_TL = num_TL +1;
                    TT(num_TL) = tabu_tenture;
                    moveApplied = true;
                end
                num = num +1;
            end
        else
            if(num_ML > 0)
                %disp('Apply the best move (no TL)');
                CS = ML(index(1), :);
                TL(num_TL+1, :) = ML(index(1), :);
```

```
                num_TL = num_TL +1;
                TT(num_TL) = tabu_tenture;
                %moveApplied = true;
            end
        end
        clear ML;
        clear fitness_ML;
        CS = TS_clean(CS, row);
        col_limit = 0;
        for j = 1 : col
           if(CS(j) ~= 0)
               col_limit = col_limit +1;
           end
        end
        [CS_fitness(iteration), comp, CS_matches(iteration)] =
TS_fitnessCalculation(CS, row, col_limit);
        f = CS_fitness(iteration);
        m = CS_matches(iteration);
        if(iteration ==1 || rem(iteration, 10) == 0)
            f_steps = f_steps +1;
            f_ts(f_steps) = f;
            c_ts(f_steps) = comp;
        end
        [TL, TT, num_TL] = TS_tabuUpdate(TL, TT, num_TL);
        iteration = iteration +1;
  end
  TS_iterationCount = iteration -1
   f = f_ts;
   c = c_ts;
   csOUT = CS;
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [fitness, comp, matches] = TS_fitnessCalculation(s, row, col_limit)
    col = 64/row;
    sq = zeros(row, col);
    for i = 1 : row
        for j = 1 : col
            pos = ((i-1)*col) + j;
            sq(i, j) = s(pos);
        end
    end
    value = 0;
    counter = 0;
    matches = 0;
        for j = 1 : row -1
            for k = j+1 : row
                for m = 1 : col_limit
                    if((sq(j,m) == 5) && (sq(k,m) == 5))
                         %value = value - 0;
                         counter = counter + 1;
                    elseif(sq(j,m) == sq(k,m))
                         value = value + 8;
                         matches = matches +1;
                         counter = counter + 1;
                    elseif((sq(j,m) == 5) || (sq(k,m)== 5))
                         value = value + 1;
                         counter = counter + 1;
                    elseif(sq(j,m) ~= sq(k,m))
```

```
                        value = value  + 2;
                        counter = counter + 1;
                    end
                end
            end
        end
 fitness = value/counter;
 comp = counter;
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [move, seed]= TS_SSM(CS, ptr, seed, row)
    col = 64/row;
    sq = zeros(row, col);
    for i = 1 : row
        for j = 1 : col
            pos = ((i-1)*col) + j;
            sq(i, j) = CS(pos);
        end
    end
    success = false;
    counter = 0;
    index = 0;
    gaps = 0;  %% requires forced quantization
    upperLimit = col - 1;
    while(success == false && counter < 50) %upto 20 trial to find a gap
         [rp, seed]= randomGenerator(seed, upperLimit);
         if(sq(ptr, rp) == 5)       % only one chance is given
             counter = counter +1;
             success = true;
             gaps = 1;
             index = rp;
             j = rp+1;
             enough = false;
             while(j < col && enough == false)
                 if(sq(ptr, j) == 5)
                    gaps = gaps +1;
                 else
                   enough = true;
                 end
                 j = j+1;
             end
         else
             counter = counter +1;
         end
    end
    if (gaps > 0)
       if((index + gaps) <= col)
           sq(ptr, index) = sq(ptr, index + gaps);
           sq(ptr, index + gaps) = 5;
       end
    end
    for i = 1 : row
        for j = 1 : col
            pos = ((i-1)*col) + j;
            sqq(pos) = sq(i, j);
        end
    end
    move = sqq;
```

```
%end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [move, seed] = TS_BSM(CS, ptr, seed, row)
    col = 64/row;
    sq = zeros(row, col);
    for i = 1 : row
        for j = 1 : col
            pos = ((i-1)*col) + j;
            sq(i, j) = CS(pos);
        end
    end
    gaps = 0;
    success = false;        % traces whether there is a move or not
    %gaps - numbergaps in sq ptr exluding last column);
    for j = 1 : row
        if(j <= col -1)
            if(sq(ptr, j) == 5)
                gaps = gaps +1;
                gapPos(gaps) = j;
            end
        end
    end
    upperLimit = gaps;
    counter = 1;
    while (counter <= gaps && success == false)
        depth = 0;
        [rp, seed]= randomGenerator(seed, upperLimit);
        target_col = gapPos(rp);
         if(sq(ptr +1, target_col) == 5)
            success = true;
         end
        if(success == true)
            k = ptr +1;
            enough = false;
            while(k <= 4 && enough == false)
                if (sq(k, target_col) == 5)
                    depth = depth +1;
                else
                    enough = true;
                end
                k = k+1;
            end
            curr_solution = sq;
            if(depth > 0)
                for m = ptr : ptr + depth
                    curr_solution(m, target_col) = sq(m, target_col +1);
                    curr_solution(m, target_col+1) = 5;
                end
            end
            if (success == true)
                move = curr_solution;
            end
        end
        counter = counter +1;
    end
   if (success == false)
       move = CS; % what if this condition left out
```

```
   else
       for i = 1 : row
            for j = 1 : col
                pos = ((i-1)*col) + j;
                sqq(pos) = sq(i, j);
            end
       end
       %move = TS_clean(sqq, row);
       move = sqq;
   end

 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 function [randVal, seed] = randomGenerator(seed, upperLimit)
    m = 3001;
    a = 14;
    x = 0;
    while(x < 1)
        temp = a * seed;
        seed = mod(temp, m);
        temp1 = seed/m;
        temp2 = upperLimit * temp1;
        randVal = round(temp2);
        x = randVal;
    end
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function index = TS_indexOfSortedFitness(fitness, n)
  index = zeros(1, 8);
  pos = 1;
  for i = 1 : 8;
    if(i <= n)
        maxValue = 0.005;
        for j = 1: 8
            if(maxValue < fitness(j) && j <= n)
                maxValue = fitness(j);
                pos = j;
            end
        end
        fitness(pos) = 0;
        index(i) = pos;
    end
  end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [TL, TT, num_TL] = tabuUpdate(TL, TT, num_TL)
    for i = 1 : num_TL
        TT(i) = TT(i) -1;
    end
    ptr = 1;
    updated = false;
    while(ptr <= num_TL && updated == false)
        if(TT(ptr) < 1)
            k = ptr;
            for j = k : num_TL - 1
              TL(j, :) = TL(j+1, :);
              TT(j) = TT(j+1);
            end
            updated = true;
        else
```

```
            ptr = ptr +1;
        end
    end
    if(updated == true)
        num_TL = num_TL -1;
    end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function S = TS_clean(S, row)
    col = 64/row;
    individual = zeros(row, col);
    for i = 1 : row
        for j = 1 : col
            pos = ((i-1)*col) + j;
            individual(i, j) = S(pos);
        end
    end
    colOfGaps = 1;
    while(colOfGaps ~= 0)
        colOfGaps = 0;
        for j = 1 : col
            if(individual(:, j) == 5)
                colOfGaps = j;
            end
        end
        if(colOfGaps ~= 0)
            for j = 1 : col -1
                if(j >= colOfGaps)
                    individual(:, j) = individual(:, j+1);
                end
            end
            individual(:, col) = 0;
        end
    end

    for i = 1 : row
        for j = 1 : col
            pos = ((i-1)*col) + j;
            S(pos) = individual(i, j);
        end
    end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [matches, mismatches, base_Gaps, gap_Gaps, fitness] =
measureOfSimilarity(s, row, col)
        matches = 0;
        mismatches = 0;
        base_Gaps = 0;
        gap_Gaps = 0;
        value = 0;
        counter = 0;
        for j = 1 : row -1
            for k = j+1 : row
                for m = 1 : col
                    if((s(j,m) == '-') && (s(k,m) == '-'))
                         gap_Gaps = gap_Gaps +1;
                         %value = value - 0;
                         counter = counter + 1;
                    elseif(s(j,m) == s(k,m))
```

```
                        matches = matches + 1;
                        value = value + 8;
                        counter = counter + 1;
                    elseif((s(j,m) == '-') || (s(k,m)== '-'))
                        base_Gaps = base_Gaps +1;
                        value = value + 1;
                        counter = counter + 1;
                    elseif(s(j,m) ~= s(k,m))
                        mismatches = mismatches +1;
                        value = value  + 2;
                        counter = counter + 1;
                    end
                end
            end
        end
       fitness = value/counter;
  end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function plotFitness(f1, f2)
limit1 = size(f1, 2);
limit2 = size(f2, 2);
for i = 1 : limit1
    x1(i) = i;
    y1(i) = f1(i);
end
a = x1(3);
b = f1(a);
plot(x1,y1,'--rs','LineWidth',2,...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor','g',...
                'MarkerSize',2)
text(a, b,'\leftarrow for GA phase',...
                'HorizontalAlignment','left')
hold all

for i = 1 : limit2
    x2(i) = i;
    y2(i) = f2(i);
end
a = x2(3);
b = f2(a);
plot(x2,y2,'--bs','LineWidth',2,...
                'MarkerEdgeColor','k',...
                'MarkerFaceColor','y',...
                'MarkerSize',2)
text(a, b,'\leftarrow for TS phase',...
                'HorizontalAlignment','left')

title('Fitness value vs iteration')
xlabel('iteration')
ylabel('Fitness')
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

# DECLARATION

This thesis is the presentation of my work. Whenever the contributions of others are involved, efforts to clearly indicate those contributions are made, with due reference to the literature.

The work was done under the guidance of Dr. Kumudha Raimond, at Addis Ababa Institute of Technology, Addis Ababa, Ethiopia.

Addisu Galassa Guddissa

Name of the candidate | Signature of the candidate

In my capacity as a advisor of the candidate's thesis, I certify that the above statements are true to the best of my knowledge.

Dr. Kumudha Raimond

Name of the advisor | Signature of the advisor

Date: October 21, 2011