M.Sc. Engg. Thesis

Optimal Placement of Unique Restriction Sites in Synthetic Genomes

by

Mahfuza Sharmin
Student No.: 0409052026P

Submitted to
Department of Computer Science & Engineering
in partial fulfillment of the requirements for the degree of
Masters in Science in Computer Science and Engineering

Department of Computer Science & Engineering
Bangladesh University of Engineering & Technology(BUET)
Dhaka-1000.

July 2012

Declaration of Authorship

This is to certify that the work presented in this thesis entitled "Optimal Placement of Unique Restriction Sites in Synthetic Genomes" is the outcome of the investigation carried out by me under the supervision of Dr. M. Sohel Rahman, Associate Professor, Department of Computer Science and Engineering(BUET), Dhaka. It is also declared that niether this thesis nor any part thereof has been submitted or being currently submitted anywhere else for the award of any degree or diploma.

(Author)
Mahfuza Sharmin
Student No.: 0409052026P
Department of Computer Science and Engineering (BUET),
Dhaka-1000

Board of Examiners

The thesis titled "Optimal Placement of Unique Restriction Sites in Synthetic Genomes," submitted by Mahfuza Sharmin, Roll No. 0409052026P, Session April 2009, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory in partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Engineering and approved as to its style and contents. Examination held on July 03, 2012.

Dr. M. Sohel Rahman Associate Professor Department of Computer Science and Engineering BUET, Dhaka 1000	Chairman (Supervisor)
Dr. Abu Sayed Md. Latiful Hoque Professor & Head Department of Computer Science and Engineering BUET, Dhaka 1000	Member (Ex-officio)
Dr. M. Kaykobad Professor Department of Computer Science and Engineering BUET, Dhaka 1000	Member
Dr. Masud Hasan Associate Professor Department of Computer Science and Engineering BUET, Dhaka 1000	Member
Dr. Mohammad Nurul Huda Associate Professor Department of Computer Science and Engineering United International University, Dhaka	Member (External)

Acknowledgements

All praises due to Allah, the most benevolent and the most merciful.

I show my heartfelt gratitude toward my supervisor Dr. M. Sohel Rahman, who was very helpful during the entire span of my research work. Without his direction, support and advice, this work would not have been possible. I am especially grateful to him for allowing me greater freedom in choosing the problems to work on, for his encouragement at times of disappointment, and for his patience with my wildly sporadic work habits.

I would like to thank Bangladesh University of Engineering and Technology for its generous support and research grant. The university also provided me with its library facilities and online resource facilities. I would also like to thank Department of Computer Science and Engineering for its support with resources and materials during the research work.

I also remember my teachers and colleagues who earnestly provided me with encouragement and inspiration for achieving this goal.

Last but not the least, I am thankful to my parents, family and friends for their support and tolerance.

Abstract

There has been a number of research that has investigated the function of genes in a sequence and how to synthesize genome sequence according to user specification. The purpose of this thesis is to find a genome sequence which provides maximum amount of flexibility and independence to biologists to run experiment with the sequence. Another aim is to give opportunity for investigation of vaccine invention. To this end, we propose to apply metaheuristics process for making a genome sequence uniquely prone to enzymes.

We have applied a family of local and global search techniques to investigate which search technique is better applicable for the problem under consideration. All implementation of our algorithms are incorporated in the existing sequence design tool, namely, $PRESTO$. Finally in our study, the process are simulated on a number of viral sequences and the outcome is examined by statistical means. Through our extensive experimental and statistical analysis, we have found that our local search techniques perform better than the existing heuristics. Our findings also include that some global search techniques do not perform as expected, even though they explores the search space more. We have also found that multi-objective pareto optimization gives best output in the current context.

Contents

Declaration of Authorship i

Board of Examiners ii

Acknowledgements iii

Abstract iv

List of Figures viii

List of Tables x

1 Introduction 1
- 1.1 Synthetic Biology . . . 1
  - 1.1.1 Computer Science and Synthetic Biology . . . 2
- 1.2 Problem Background . . . 3
- 1.3 Contribution in this Thesis . . . 3
- 1.4 Organization of the Thesis . . . 4
  - 1.4.1 Preliminaries . . . 4
  - 1.4.2 Literature Survey . . . 4
  - 1.4.3 URSPP in Synthetic Genomes by Metaheuristics . . . 5
  - 1.4.4 Experimental Results . . . 5
  - 1.4.5 Conclusion . . . 5

2 Preliminaries 6
- 2.1 Synthetic biology . . . 6
  - 2.1.1 Viral Genome Synthesis . . . 7
  - 2.1.2 Refactoring: Genome Vs. Software . . . 7
  - 2.1.3 Restriction Enzymes and Restriction Sites . . . 8
  - 2.1.4 Protein and Amino acid . . . 8
  - 2.1.5 Genetic Code: Codon . . . 9

2.1.6 Subcloning . . . 10
2.2 Problem Statement . . . 11
2.3 Definition of Metaheuristics . . . 12
2.4 Single State Methods . . . 13
2.4.1 Hill Climbing . . . 13
2.4.2 Steepest Ascent Hill Climbing . . . 14
2.4.3 Steepest Ascent Hill Climbing with Replacement . . . 14
2.5 Population Methods . . . 15
2.5.1 Genetic Algorithm . . . 15
2.5.2 Genetic Algorithm with Elitism . . . 16
2.5.3 Steady State Genetic Algorithm . . . 17
2.6 Evolutionary Multiobjective Optimization . . . 18
2.7 Selection Procedure . . . 19
2.7.1 Tournament Selection . . . 19
2.7.2 Multiobjective Tournament Selection . . . 19
2.8 Non-Dominated Sorting Genetic Algorithm . . . 21
2.9 Summary . . . 22

3 Literature survey 23
3.1 Previous Results on *URSPP* . . . 23
3.2 Related Works on Gene Synthesis . . . 25
3.3 Summary . . . 30

4 URSPP in Synthetic Genomes by Metaheuristics 31
4.1 Motivation behind Application of Metaheuristics . . . 31
4.2 Proposed Methodology . . . 32
4.2.1 Candidate Solution Representation . . . 33
4.2.2 Breeding Operators . . . 34
4.2.3 Quality Assessment . . . 36
4.3 Algorithms . . . 38
4.3.1 Local Search Techniques . . . 38
4.3.2 Hybrid Genetic algorithm . . . 38
4.3.3 Non-Dominated Sorting Genetic Algorithm . . . 39
4.4 Summary . . . 40

5 Experimental Results 41
5.1 Statistical Test . . . 41
5.1.1 Paired or Unpaired Test . . . 41
5.1.2 The T Test . . . 42
5.1.3 Concept of null hypothesis . . . 42
5.1.4 Significance levels . . . 43
5.1.5 One or Two sided P Value . . . 43
5.1.6 Confidence Interval (C.I.) . . . 44
5.2 Simulation Results . . . 45
5.2.1 Experimental set up and Representation . . . 45

5.2.2 Results summary and Analysis 47
5.3 Summary 84

6 Conclusion **85**
6.1 Findings 85
6.2 Future Direction 86

List of Figures

2.1 EcoRV enzyme cuts at GATATC. 8
2.2 Insertion of a restriction site. 10
2.3 Sparsity of Individual B is higher than Individual A 21

3.1 Moving window algorithm. 26
3.2 SiteFind Screenshots. 27
3.3 KLF4 R390S mutant has a novel BglII restriction site. 27
3.4 KLF4 K225/229R mutant has a novel NheI restriction site. 28
3.5 User Interface of GeneJax 28

4.1 Restriction enzyme, RE_1 is already inserted 31
4.2 Deletion of RE_1 might allow to insert RE_2 and RE_3 32
4.3 Mutation 1 34
4.4 Mutation 1 35
4.5 Mutation 2 35
4.6 Pareto ranks 37
4.7 Selection of best solution from front 39

5.1 Confidence Interval 45
5.2 Convergence of Objective, f_3 47
5.3 Hill Climbing Vs Existing Heuristics for λ Phage Virus 48
5.4 Steepest Ascent HC Vs Existing Heuristics for λ Phage Virus 48
5.5 Steepest Ascent HC with Replacement Vs Existing Heuristics for λ Phage Virus 48
5.6 Hybrid GA Vs Existing Heuristics for λ Phage Virus 52
5.7 Hybrid GA with Elitism Vs Existing Heuristics for λ Phage Virus . . . 58
5.8 Hybrid Steady State GA Vs Existing Heuristics for λ Phage Virus . . . 58
5.9 Non-Domonated Sorting GA Vs Existing Heuristics for λ Phage Virus . 59
5.10 Non-Domonated Sorting GA Vs Local Search algorithms for λ Phage Virus 61
5.11 Hill Climbing Vs Existing Heuristics for Polio Virus 62
5.12 Steepest Ascent HC Vs Existing Heuristics for Polio Virus 64
5.13 Steepest Ascent HC with Replacement Vs Existing Heuristics for Polio Virus 64
5.14 Hybrid GA Vs Existing Heuristics for Polio Virus 64
5.15 Hybrid GA with Elitism Vs Existing Heuristics for Polio Virus 68

5.16 Hybrid Steady State GA Vs Existing Heuristics for Polio Virus 68
5.17 Non-Domonated Sorting GA Vs Existing Heuristics for Polio Virus . . 68
5.18 Non-Domonated Sorting GA Vs Local Search algorithms for Polio Virus 72
5.19 Hill Climbing Vs Existing Heuristics for Equine Arteritis Virus 73
5.20 Steepest Ascent HC Vs Existing Heuristics for Equine Arteritis Virus . 73
5.21 Steepest Ascent HC with Replacement Vs Existing Heuristics for Equine Arteritis Virus 73
5.22 Non-Domonated Sorting GA Vs Existing Heuristics for Equine Arteritis Virus 77
5.23 Non-Domonated Sorting GA Vs Local Search algorithms for Equine Arteritis Virus 77
5.24 Hill Climbing Vs Existing Heuristics for Measles Virus 77
5.25 Steepest Ascent HC Vs Existing Heuristics for Measles Virus 79
5.26 Steepest Ascent HC with Replacement Vs Existing Heuristics for Measles Virus 81
5.27 Non-Domonated Sorting GA Vs Existing Heuristics for Measles Virus . 82
5.28 Non-Domonated Sorting GA Vs Local Search algorithms for Measles Virus 83

List of Tables

2.1 Standard Genetic Code 9

4.1 One Candidate Solution 33

5.1 Algorithms 46
5.2 Prerence of Objective difference 46
5.3 Hill Climbing Vs Existing Heuristics for λ Phage Virus 49
5.4 Steepest Ascent HC Vs Existing Heuristics for λ Phage Virus 50
5.5 Steepest Ascent HC with Replacement Vs Existing Heuristics for λ Phage Virus 51
5.6 Hybrid GA (low local improver) Vs Existing Heuristics for λ Phage Virus 53
5.7 Hybrid GA (moderate local improver) Vs Existing Heuristics for λ Phage Virus 54
5.8 Hybrid GA (high local improver) Vs Existing Heuristics for λ Phage Virus 55
5.9 Hybrid GA with Elitism Vs Existing Heuristics for λ Phage Virus . . . 56
5.10 Hybrid Steady State GA Vs Existing Heuristics for λ Phage Virus . . . 57
5.11 Non-Domonated Sorting GA Vs Existing Heuristics for λ Phage Virus . 60
5.12 Non-Domonated Sorting GA Vs Local Search algorithms for λ Phage Virus 61
5.13 Hill Climbing Vs Existing Heuristics for Polio Virus 63
5.14 Steepest Ascent HC Vs Existing Heuristics for Polio Virus 65
5.15 Steepest Ascent HC with Replacement Vs Existing Heuristics for Polio Virus 66
5.16 Hybrid GA Vs Existing Heuristics for Polio Virus 67
5.17 Hybrid GA with Elitism Vs Existing Heuristics for Polio Virus 69
5.18 Hybrid Steady State GA Vs Existing Heuristics for Polio Virus 70
5.19 Non-Domonated Sorting GA Vs Existing Heuristics for Polio Virus . . 71
5.20 Non-Domonated Sorting GA Vs Local Search algorithms for Polio Virus 72
5.21 Hill Climbing Vs Existing Heuristics for Equine Arteritis Virus 74
5.22 Steepest Ascent HC Vs Existing Heuristics for Equine Arteritis Virus . 75
5.23 Steepest Ascent HC with Replacement Vs Existing Heuristics for Equine Arteritis Virus 76
5.24 Non-Domonated Sorting GA Vs Existing Heuristics for Equine Arteritis Virus 78

5.25 Non-Domonated Sorting GA Vs Local Search algorithms for Equine Arteritis Virus 79
5.26 Hill Climbing Vs Existing Heuristics for Measles Virus 80
5.27 Steepest Ascent HC Vs Existing Heuristics for Measles Virus 80
5.28 Steepest Ascent HC with Replacement Vs Existing Heuristics for Measles Virus 81
5.29 Non-Domonated Sorting GA Vs Existing Heuristics for Measles Virus . 82
5.30 Non-Domonated Sorting GA Vs Local Search algorithms for Measles Virus 83

Chapter 1

Introduction

The use of biological techniques has grown explosively in the past few decades and shows no sign of slowing down. The result of this growth is that the number of sources of products, services, and information has increased to the point that keeping track of (or locating) the numerous providers has become extremely time consuming. The sheer volume of biological data makes it impossible to analyze them by hand. As a result, computers have become indispensable to biological research. This chapter introduces the reader with the state of the art of biological research and provides an overview of what we have done in our thesis. In particular, we give an overview of a popular branch of bioinformatics, namely, synthetic biology. It is the latter sub discipline of bioinformatics, which the subject of this thesis belongs to.

1.1 Synthetic Biology

Synthetic biology is an emerging and exciting field in bioinformatics and genetic engineering that involves redesigning of existing, natural biological systems for new purposes as well as for the creation of entirely novel artificial living things. For several decades, genetic modification technologies have been used to move genes from one species and splice them into the DNA of another. Here the ultimate goal is to create transgenic plants/animals/microorganisms with new and improved characteristics. The focus in synthetic biology, however, is to create novel living systems (with a similar goal as above) from standardized genetic parts. The research here poses various newer algorithmic and computational problems and challenges. Viral genome synthesis, i.e.,

designing novel living organisms at the genetic level, is an important research area in this field.

1.1.1 Computer Science and Synthetic Biology

In simple words, Synthetic biology is nothing but putting engineering into biology. An engineered genetic toggle switch [1] developed by Tim Gardner and Jim Collins is a good example of how engineering principles are driving the boat of synthetic biology [1]. Researchers are now trying to adapt concepts developed in area of programming language development and software engineering for synthetic biology applications. For example, a recent paper in *PLoS Computational Biology* shows how methods used by computer scientists to develop programming languages can be applied to DNA sequences [2]. Authors in [2] report an attribute grammar based formalism to model the structure-function relationships in synthetic DNA sequences. An attribute grammar is constructed as an extension of a context-free grammar and in computer science it is commonly used to translate the text of a program source code or the syntax tree directly into the computational operations or machine level instructions.

Another example of the study and research under the hood of synthetic biology is re-designing Bacteria. Bacteria are the simplest known objects from the natural world that are capable of replicating when provided with only simpler components (e.g., broth). Still, bacteria are far from simple. Bacteria also provide the basic environment in which synthetic biological systems exist and act; in some sense, they are like the power supply and chassis of a computer. By re-designing/refactoring a simple living system we hope to learn how to better couple (and decouple) our designed systems from their host environment. As engineers, we are much better at thinking and designing digital systems. One reason we are better at digital system design is that such systems create an 'abstraction barrier' between the detailed device physics level and the system design and operation levels.

[1] Genetic toggle switch is a synthetic, bistable gene-regulatory network-in an organism e.g. Escherichia coli. The toggle is flipped between stable states using transient chemical or thermal induction and exhibits a nearly ideal switching threshold.

1.2 Problem Background

Our problem has basically come from synthetic biology research. A large part of synthetic biology research deals with genome synthesis and designing synthetic gene. Genome synthesis has many interesting applications. One of such application is drug or vaccine design. Our problem has direct relationship with viral vaccine design. As an example, when a virus attacks the genome sequence os virus is combined with the our genome sequence. This combination is the cause of our disease. To design a vaccine for that virus what biologists do is compare two genome sequence and a part of viral genome sequence is disturbed. The disturbance is done in such a way that the partly disturbed genome sequence can act as a potential vaccine. Here, biologists need to analyze and take some decision on which part of viral sequence should be disturbed and which part should be kept intact. For making this decision computer science plays a role, in particular it assists with genome refactoring which means creating a new genome sequence which is functionally same to the original sequence but is much easier to manipulate in the laboratory for chemical and biological test. The problem under consideration has directly come from such a lab project. Here, our target is to find a genome sequence from an existing genome sequence so that it becomes uniquely responsive to some given chemical agents. this uniqueness is particularly prized, because if the new but equivalent sequence is uniquely responsive to a chemical agent, the function of the affected genetic part can be unambiguously understood.

1.3 Contribution in this Thesis

The main target of this thesis is to aid in drug designing, in particular vaccine designing. To design an effective vaccine, we need to analyze functionality of genes and their parts contained in a viral genome sequence. To better understand what a specific part in the sequence does, what is does not, it is a biologically popular way to keep that specific part in the sequence and temporarily remove/abrupt/make de-functional others using some chemicals (enzymes). Synthesizing viral sequences in such a way so that specific part of the sequence is responsive to some enzymes and others are non-responsive, is a computationally hard problem. In this thesis, we apply high level search techniques, to find such synthesized genome sequences. To achieve this, we first map the problem into a multi-objective optimization problem and define problem specific genetic operators so that efficient meta-heuristic techniques can be applied. We have designed suitable

hybrid meta-heuristic algorithms powered by local search techniques to solve the problem. Our implementation can serve as a sequence design tool for synthesized genomes. Additionally, we have conducted extensive experiments to analyze the performance of the designed algorithms using viral genome/DNA sequence. To the best of our knowledge, the synthesized genome designed by this tool are most attractive to the biologists because of cost and other criteria.

1.4 Organization of the Thesis

A brief overview of the organization of the rest of this thesis is presented in this section.

1.4.1 Preliminaries

A fundamental challenge in bioinformatics is transforming a biological problem into a computational problem. Before such conversion, it is quite necessary to understand the biological context so that they can smartly be applied in algorithmic context. Smart application of one context to another refers to the fact that the rules-regulation and conditions of the former branch does not violate due to pulling the problem into the latter; rather such usage or pulling eases the original problem to solve. Chapter 2 deals with some relevant concepts and notions meeting this challenge. Here, we also presents basics of Metaheuristics. We discuss several types of local and global search algorithms which we use later. Additionally, some selection procedure to find good solutions are briefed.

1.4.2 Literature Survey

This chapter provides the short description of the work of Montes [3] on optimization of restriction site placement. Some related works on genome synthesis are also discussed in detail.

1.4.3 URSPP in Synthetic Genomes by Metaheuristics

As has been mentioned above in this thesis, we have tried to construct new genome sequences applying different heuristic techniques. This chapter first presents our motivation behind using such search techniques. Then all the required notions to apply the global search methods are introduced at length. Finally, we formally present our algorithms in this chapter.

1.4.4 Experimental Results

We also design and conduct extensive experiments to analyze the performance of our algorithms and provide insightful discussion based on the comparisons done with state of the art algorithms on the same problem. Section 5 is dedicated to present the experimental results. Here, we have given our implementation set up and the supporting software tools used for the simulating the algorithms. At the end, we have provided summary results along with insightful discussions about the outcome followed by a comparative analysis with other related results in the literature.

1.4.5 Conclusion

This chapter concludes our thesis and puts our results into the context of the state of the art of the literature. Some future research directions are also identified and briefly discussed.

Chapter 2

Preliminaries

In this chapter, we discuss some terminology, concepts and relevant notations that are used throughout the thesis. The terms discussed here, belong to both biological and algorithmic studies. The discussion in this chapter will help the readers to grasp the significance and importance of our study and will aid in properly comprehending the meaning of studied problem which are discussed throughout the subsequent chapters.

2.1 Synthetic biology

A brief description of synthetic biology has been given in Chapter 1. Since our focus is on this particular branch of bioinformatics, here, we discuss about it more elaborately. The new buzzword 'Synthetic Biology' entered the vocabulary of the scientific community only a few years ago [4–7]. A consensus definition drafted by a group of European experts [8] defined Synthetic Biology as follows:

> Synthetic biology is the engineering of biology: the synthesis of complex, biologically based (or inspired) systems, which display functions that do not exist in nature. This engineering perspective may be applied at all levels of the hierarchy of biological structures from individual molecules to whole cells, tissues and organisms. In essence, synthetic biology will enable the design of biological systems' in a rational and systematic way.

Synthetic biology refers to both: (a) the design and fabrication of biological components and systems that do not already exist in the natural world and (b) the re-design and

fabrication of existing biological systems. There are two types of synthetic biologists. The first group uses unnatural molecules to mimic natural molecules with the goal of creating artificial life. The second group uses natural molecules and assembles them into a system that acts unnaturally. In general, the goal is to solve problems that are not easily understood through analysis and observation alone and it is only comprehensible by the manifestation of new models.

Synthetic biology studies how to build artificial biological systems for engineering applications, using many of the same tools and experimental techniques. But the work is fundamentally an engineering application of biological science, rather than an attempt to do more science. The focus is often on ways of taking parts of natural biological systems, characterizing and simplifying them, and using them as a component of a highly unnatural, engineered, biological system.

2.1.1 Viral Genome Synthesis

Gene synthesis has become an important tool in many fields of recombinant DNA technology including vaccine development, gene therapy and molecular engineering. The synthesis of nucleic acid sequences is often more economical than classical cloning and mutagenesis procedures. Gene synthesis is the process of synthesizing a gene without the need for initial template DNA samples. Genome synthesis technique is largely used for virus vaccine development. For example, a team of molecular biologists and computer scientists at Stony Brook University has modified the polio virus to create a weakened version, which, when injected, went on to effectively immunize lab mice [9]. They used a novel algorithm to sort through potential recordings of the genome that would produce the desired proteins. The technique may lead to practical, systematic methods of developing future viral vaccines.

2.1.2 Refactoring: Genome Vs. Software

Refactoring [10] is a software engineering term for redesigning a program to improve its internal structure for better ease of maintenance while leaving its external behavior unchanged. Genome synthesis technology enables us to refactor biological organisms: we seek to restructure the genome of an organism into a sequence which is functionally equivalent (i.e., behaves the same in its natural environment) while being easier to manipulate [11, 12].

2.1.3 Restriction Enzymes and Restriction Sites

Restriction enzymes are reagents widely used by molecular biologists for genetic manipulation and analysis. For diagnosing/synthesizing DNA sequence restriction enzymes offer nonparallel opportunities. These special types of enzymes *recognize* and *cut* specific nucleotide sequences in DNA molecules. For example, the bacterium Hemophilus aegypticus produces an enzyme named HaeIII that cuts DNA wherever it encounters the sequence 5'GGCC3'/3'CCGG5'. The pattern, GGCC/CCGG, is called the *restriction enzyme recognition site* or *restriction site* or *recognition site*. *Unique restriction enzyme* cuts the DNA at exactly one place. Due to their unambiguous recognition property, unique restriction enzymes are more interesting, useful and appealing to the scientists. A sequence containing unique restriction sites at regular intervals is easier to manipulate in the laboratory. That is why, in many cases, before experimenting the original sequence obtained from a living organism, such unique sites are artificially inserted and/or deleted in the DNA sequence keeping it functionally equivalent to the original one. Often, inserting (deleting) a recognition site for a restriction enzyme is referred to as inserting (deleting) a restriction enzyme. Figure 2.1 shows an example of this concept for EcoRv enzyme which can recognize the sequence, GATATC.

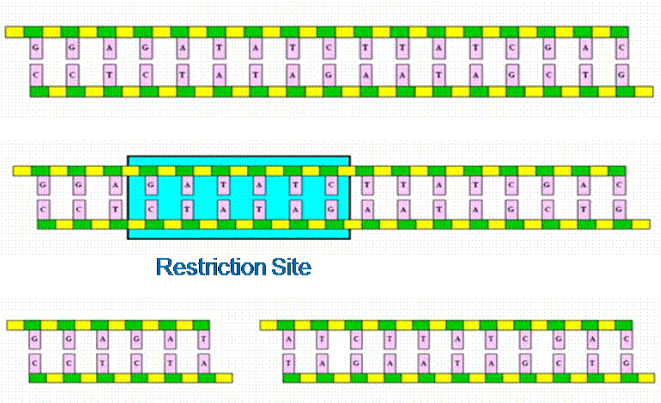

FIGURE 2.1: EcoRV enzyme cuts at GATATC.

2.1.4 Protein and Amino acid

Proteins are one of the building blocks of the body and Amino acids serve as the building blocks of proteins, which are basically, linear chains of amino acids. Amino acids can be linked together in varying sequences to form a vast varieties of proteins. A total of 20 different kinds of amino acids form proteins. These 20 amino acids are encoded by the universal genetic code. To restructure the genome of an organism into a functionally

equivalent sequence, the amino acid sequence of it must be preserved. The redundancy of the genetic code (to be described in the following section) plays an important role in preserving the amino acid sequences or proteins even after the insertion of a new restriction site at a certain place of the genome or after the removal of any restriction site from it.

2.1.5 Genetic Code: Codon

Each amino acid is encoded by a series of three adjacent bases, called *Codon*. Codons specify which amino acid will be added next during protein synthesis. With some exceptions, a three-nucleotide codon in a nucleic acid sequence specifies a single amino acid. The genetic code has redundancy but no ambiguity (see the Table 2.1). For example, although codons GAA and GAG both specify glutamic acid (redundancy), neither of them specifies any other amino acid (no ambiguity). There are three amino acids encoded by six different codons: serine, leucine, and arginine. Only two amino acids are specified by a single codon. One of these is the amino-acid methionine, specified by the codon ATG, which also specifies the start of translation; the other is tryptophan, specified by the codon TGG.

TABLE 2.1: Standard Genetic Code

1st base	2nd base								3rd base
	T		C		A		G		
T	TTT	Phe/F Phenylalanine	TCT	Ser/S Serine	TAT	Tyr/Y Tyrosine	TGT	Cys/C Cysteine	T
	TTC		TCC		TAC		TGC		C
	TTA	Leu/L Leucine	TCA		TAA	Stop	TGA	Stop	A
	TTG		TCG		TAG	Stop	TGG	Trp/W Tryptophan	G
C	CTT		CCT	Pro/P Proline	CAT	His/H Histidine	CGT	Arg/R Arginine	T
	CTC		CCC		CAC		CGC		C
	CTA		CCA		CAA	Gln/Q Glutamine	CGA		A
	CTG		CCG		CAG		CGG		G
A	ATT	IIe/I Isoleucine	ACT	Thr/T Threonine	AAT	Asn/N Asparagine	AGT	Ser/S Serine	T
	ATC		ACC		AAC		AGC		C
	ATA		ACA		AAA	Cys/k Lysine	AGA	Arg/R Arginine	A
	ATG	Met/M Methionine	ACG		AAG		AGG		G
G	GTT	Val/V Valine	GCT	Ala/A Alanine	GAT	Asp/D Aspartic acid	GGT	Aly/G Glycine	T
	GTC		GCC		GAC		GGC		C
	GTA		GCA		GAA	Glu/E Glutamic Acid	GGA		A
	GTG		GCG		GAG		GGG		G

Degeneracy results because there are more codons than encodable amino acids. There are only 20 different kinds of amino acids found in the proteins of living organisms whereas 64 ($4^3 = 64$) possible codons are available. Multiple codons representing the same amino acid are called *Synonymous codons*. These properties of the genetic code make it more *fault-tolerant* for point mutations. For successful insertion and deletion of restriction sites, synonymous codons must be used while placing one codon in lieu

of another. Figure 2.2 shows an example of this concept. GAG and GAA both are the code for the aminoacid glutamic acid. Here a single nucleotide change introduces the EcoRI restriction site, without modifying the amino acid sequence.

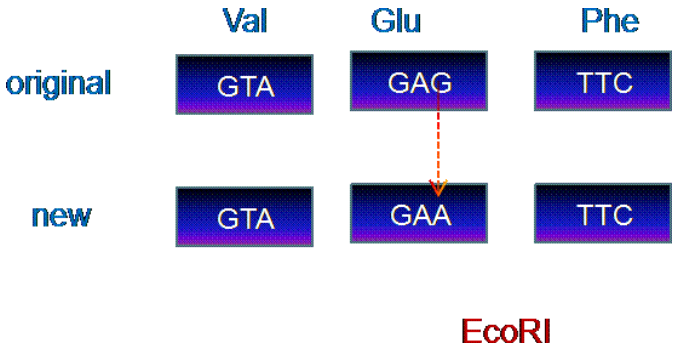

FIGURE 2.2: Insertion of a restriction site.

2.1.6 Subcloning

In molecular biology, subcloning is a technique used to move a particular gene of interest from a parent vector to a destination vector in order to further study its functionality. Restriction enzymes are used to excise the gene of interest (the insert) from the parent. Simultaneously, the same restriction enzymes are used to digest (cut) the destination. The insert and the destination vector are then mixed together with certain ratio. After letting the reaction mixture sit for a set amount of time at a specific temperature, the insert should become successfully incorporated into the destination plasmid.

2.2 Problem Statement

The problem of our interest originates from the laboratory of viral genome synthesis. The original problem takes as input a viral plasmid sequence and a set of restriction enzymes. The goal then is to find a new plasmid sequence containing unique recognition sites for the given set of restriction enzymes such that (1) the number of unique sites inserted is maximum, (2) amount of sequence editing required is as less as possible and (3) the placement of the sites are as even as possible. This problem is referred to as the *Unique Restriction Site Placement Problem* (*URSPP*) in the literature [3].

While editing the sequence some practical restrictions need to be considered. To retain the original functionality of the genome, it's amino acid sequence must remain unaltered. That's why, synonymous codons are used to introduce a site to and/or delete a site from the sequence through the change of one or more bases. During such deletions and insertions no accidental occurrence of any of the restriction sites should be introduced in the sequence. The difficulty of the problem, among others, arises from the task of keeping track of exponential number of possibilities in the order of considering restriction enzymes for insertion and/or deletion. We must also decide on which occurrence of a restriction site to keep so that the site serves as a unique one. This also adds to the difficulty.

Another restriction is that a site in the so called *locked region* can't be deleted, nor a site can be inserted there. *Locked region* is the part of a genome without which the virus dies/can never function as expected, though the actual function of such a region is still a mystery to the biologists. Such *locked region*s include the places where RNA Secondary structures, Multiple Open Reading Frames[1] are found.

[1]The places where transcription process is paused.

In the rest of this chapter, we present basic definitions of metaheuristics and some search techniques. Here, our discussion on metaheuristics in subsequent sections is confined within the techniques that have been applied to solve the problem under consideration.

2.3 Definition of Metaheuristics

Metaheuristics have been established as one of the most practical approach to simulate optimization and are designed to tackle complex optimization problems where other optimization methods have failed to be either effective or efficient. These techniques are often (though not necessarily) inspired by processes occurring in nature, e.g. Darwinian Natural Selection, Annealing, Collective behavior of ants etc.

The term metaheuristic, first introduced in [13], derives from the composition of two Greek words. 'Heuristic' derives from the verb heuriskein (euriskein) which means "to find", "art of discovering new", while the suffix meta means "beyond, in an upper level". Before this term was widely adopted, metaheuristics were often called modern heuristics [14]. Below we give some recent definitions of metaheuristics [15–17],

> "A metaheuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions."

> "A metaheuristic is an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a construction method."

In summary, metaheuristics are high level strategies for exploring search spaces by using different methods. Within the application of these techniques, there is a dynamic balance between diversification and intensification: on one side we have to quickly identify regions in the search space with high quality solutions and on the other side we must not waste too much time in regions of the search space which are either already explored or

which are unlikely to provide high quality solutions. The term diversification generally refers to the exploration of the search space, whereas the term intensification refers to the exploitation of the accumulated search experience [18–20]. The balance between diversification and intensification is important in terms of time required to reach an acceptable solution, usage of memory and other resources.

Typically, metaheuristics are approximation algorithms and hence they cannot always produce provably optimal solutions. But they do have the potential to produce good solutions in short amount of time (if used appropriately). Metaheuristics are different from exact optimization algorithms in that they do not guarantee the optimality of the obtained solutions. On the other hand, unlike approximation algorithms, nor does it define how close are the obtained solutions from the optimal ones.

2.4 Single State Methods

This methods are often called local search techniques. Local Search metaheuristics belong to an emerging class of methods for tackling combinatorial search and optimization problems. It has been shown to be very effective for a large number of combinatorial problems. These techniques are based on the iterative exploration of a solution space: at each iteration, the algorithm steps from one solution to one of its neighbors, i.e., solutions that are (in some sense) close to the former.

2.4.1 Hill Climbing

Hill climbing is a single state method. This technique is related to gradient ascent, but it does not require to know the strength of the gradient or even its direction. New candidate solutions are iteratively tested in the region of the current candidate and the new ones are adopted if they are better. This enables to climb up the hill until the local optima is reached. The basic structure of Hill climbing is outlined in Algorithm 1.

Algorithm 1 HILL CLIMBING

```
1: S ← some initial candidate solution
2: repeat
3:     R ← Tweak(Copy(S))
4:     if Quality(R) > Quality(S) then
5:         S ← R
6: untill S is the ideal solution or we have run out of time
7: return S
8: end
```

2.4.2 Steepest Ascent Hill Climbing

Steepest Ascent Hill Climbing is another single state method and almost similar to Hill climbing. However, this algorithm is a little more aggressive than Hill Climbing. It creates n tweaks to a candidate solution all at one time, and then adopt the best one. The naming of this modified algorithm comes from the fact that it samples all around the original candidate solution and then picks the best, i.e., it essentially samples the gradient and marches straight up it. Algorithm 2 illustrates Steepest Ascent Hill Climbing techniques.

Algorithm 2 STEEPEST ASCENT HILL CLIMBING

```
1:  n ← number of tweaks desired to sample the gradient
2:  S ← some initial candidate solution
3:  repeat
4:     R ← Tweak(Copy(S))
5:     for n times do
6:        W ← Tweak(Copy(S))
7:        if Quality(W) > Quality(R) then
8:           R ← W
9:     if Quality(R) > Quality(S) then
10:       S ← R
11: untill S is the ideal solution or we have run out of time
12: return S
13: end
```

2.4.3 Steepest Ascent Hill Climbing with Replacement

This single state method is a popular variation of Hill Climbing and it allows more exploration. Unlike Algorithm 2, Algorithm 3 replaces S directly with R while sampling the gradient and marching straight up it. Of course, this runs the risk of losing the best solution of the run. So the algorithm is augmented to keep the best-discovered-so-far solution stashed away, in a reserve variable called $Best$. At the end of the run, we return $Best$.

Algorithm 3 STEEPEST ASCENT HILL CLIMBING WITH REPLACEMENT

```
1: n ← number of tweaks desired to sample the gradient
2: S ← some initial candidate solution
3: Best ← S
4: repeat
5:    R ← Tweak(Copy(S))
6:    for n times do
7:       W ← Tweak(Copy(S))
8:       if Quality(W) > Quality(R) then
9:          R ← W
10:   S ← R
11:   if Quality(S) > Quality(Best) then
12:      Best ← S
13: untill S is the ideal solution or we have run out of time
14: return S
15: end
```

2.5 Population Methods

Evolutionary Algorithms (EA) are popular approaches to solving single and multi objective optimization problems [21–23]. These are search methods that take inspirations from natural phenomenon of selection and survival of the fittest in the biological world. They differ from the more traditional optimization techniques in that they employ a search involving a "population" of solutions instead of a single point. Each iteration of the algorithm involves a competitive selection that weeds out poor solutions. The solutions with high "fitness" or "quality" are recombined with other solutions by swapping parts of a solution with another. Solutions are also mutated by making a small change to a single element of the solution. *Recombination* and *mutation* are used to generate new solutions that are biased towards the regions of the search space at which good solutions have already been seen. EAs are well suited for a wide range of combinatorial and continuous problems, though the different variations are tailored towards specific domains. The variant, namely, *Genetic algorithm*s are well suited for optimizing combinatorial problems.

2.5.1 Genetic Algorithm

Genetic Algorithm is a common evolutionary algorithm which keep around a sample of candidate solutions rather than a single candidate solution. The Genetic Algorithm (GA), often referred to as genetic algorithms, was invented by John Holland at the University of Michigan in the 1970s [24]. It is similar to a (μ, λ)[2] Evolution Strategy in many respects: it iterates through fitness assessment, selection and breeding, and

[2]Here, in each iteration, λ number of individuals comprise the population and only μ fittest individuals participate to produce next generation through mutation.

population reassembly. The primary difference is in how selection and breeding takes place. Whereas Evolution Strategies select the parents and then creates the children, the Genetic Algorithm little-by-little selects a few parents and generates children until enough children have been created. To breed, we begin with an empty population of children. We then select two parents from the original population, copy them, cross them over with one another, and mutate the results. This forms two children, which we then add to the child population. We repeat this process until the child population is entirely filled. Algorithm 4 provides the pseudo code for *Genetic Algorithm*.

Algorithm 4 GENETIC ALGORITHM

```
popsize ← desired population size {This is basically . Make it even.}
P ← {}
for popsize times do
    P ← P∪ new random individual
Best ← null
repeat
    for each individual P_i ∈ P do
        AssessFitness(P_i)
        if Best = nullor Fitness(P_i) > Fitness(Best) then
            Best ← P_i
    Q ← {}
    for popsize/2 times do
        Parent P_a ← SelectWithReplacement(P)
        Parent P_b ← SelectWithReplacement(P)
        Children C_a , C_b ← Crossover(Copy(P_a), Copy(P_b))
        Q ← Q∪ Mutate(C_a), Mutate(C_b)
    P ← Q
until Best is the ideal solution or we have run out of time
return Best
end
```

2.5.2 Genetic Algorithm with Elitism

The idea of Elitism is simple: we augment the Genetic Algorithm to directly inject into the next population the fittest individual or individuals from the previous population [25]. These individuals are called the elites. By keeping the best individual (or individuals) around in future populations, this algorithm begins to resemble $(\mu + \lambda)$, and has similar exploitation properties. This exploitation can cause premature convergence. To overcome this problem different ideas like, increasing the mutation and crossover noise, weakening the selection pressure, reducing how many elites, etc are employed. Elitism is effective where exploitation can help to reach near optima. Algorithm 5 illustrates Elitism.

Algorithm 5 GENETIC ALGORITHM WITH ELITISM

```
1:  popsize ← desired population size
2:  n ← desired number of elite individuals
3:  P ← {}
4:  for popsize times do
5:     P ← P∪ new random individual
6:  Best ← null
7:  repeat
8:     for each individual P_i ∈ P do
9:        AssessFitness(P_i)
10:       if Best = nullor Fitness(P_i) > Fitness(Best) then
11:          Best ← P_i
12:    Q ← { the n fittest individuals in P, breaking ties at random }
13:    for popsize/2 times do
14:       Parent P_a ← SelectWithReplacement(P)
15:       Parent P_b ← SelectWithReplacement(P)
16:       Children C_a , C_b ← Crossover(Copy(P_a), Copy(P_b))
17:       Q ← Q∪ Mutate(C_a), Mutate(C_b)
18:    P ← Q
19: until Best is the ideal solution or we have run out of time
20: return Best
21: end
```

2.5.3 Steady State Genetic Algorithm

An alternative to applying a traditional generational approach in Genetic Algorithm is to use a steady-state approach, updating the population in a piecemeal fashion rather than all at one time. This approach was popularized by the Darrell Whitley and Joan Kauths GENITOR system [26]. The idea here, is to iteratively breed a new child or two, assess their fitness, and then reintroduce them directly into the population itself, killing off some preexisting individuals to make room for them. Algorithm 6 is a version of Steady State Genetic Algorithm which uses crossover and generates two children at a time.

Algorithm 6 STEADY STATE GENETIC ALGORITHM

```
1:  popsize ← desired population size
2:  P ← {}
3:  for popsize times do
4:     P ← P∪ new random individual
5:  Best ← null
6:  repeat
7:     Parent P_a ← SelectWithReplacement(P)
8:     Parent P_b ← SelectWithReplacement(P)
9:     Children C_a , C_b ← Crossover(Copy(P_a), Copy(P_b))
10:    C_a ← Mutate(C_a)
11:    C_b ← Mutate(C_b)
12:    AssessFitness(C_a)
13:    if Fitness(C_a) > Fitness(Best) then
14:       Best ← C_a
15:    AssessFitness(C_b)
16:    if Fitness(C_b) > Fitness(Best) then
17:       Best ← C_b
18:    Individual P_d ← SelectForDeath(P)
19:    Individual P_e ← SelectForDeath(P)
20:    P ← P − {P_d, P_e}
21:    P ← P ∪ {C_a, C_b}
22: until Best is the ideal solution or we have run out of time
23: return Best
24: end
```

2.6 Evolutionary Multiobjective Optimization

The process of optimizing systematically and simultaneously a collection of objective functions is called multiobjective optimization. The underlying mechanisms of evolutionary algorithms are simple. Still they have proven themselves as a general, robust and powerful search mechanism [27]. In particular, they possess several characteristics that are desirable for problems involving, i) multiple conflicting objectives, and ii) intractably large and highly complex search spaces. The rapidly growing interest in the area of multiobjective evolutionary algorithms (MOEAs) is reflected by, e.g., a conference series [28] and two recent books dedicated to this subject [21, 29]. For MOEAs we need to do four things:

- Identify the objectives involved (the set of values that interest us),

- Determine how they interact (the fitness mapping or interconnections),

- Generate viable alternatives (the population of possible choices or niches), and

- Identify the best compromise (the Pareto optimum) in the current context.

2.7 Selection Procedure

In the metaheuristics algorithms the selection procedure plays a significant role. The selection procedure decides which candidate solutions to retain and which to reject as it wanders through the space of possible solutions to the problem. In what follows, we discuss different selection procedure techniques and algorithms with specific focus to the ones we use to solve our problem. The description that follows has mainly been borrowed from [23] after slight modification.

2.7.1 Tournament Selection

This is a non-parametric selection algorithm which is both simple and robust (tunable) [30]. It throws away the notion that fitness values mean anything other than bigger is better, and just considers their rank ordering. The algorithm returns the fittest individual of some t individuals picked at random, with replacement, from the population [Algorithm 7].

Algorithm 7 TOURNAMENT SELECTION

```
1: P ← Population
2: t ← tournament size, t ≥ 1
3: Best ← individual picked at random from P with replacement
4: for i = 2 to t do
5:     Next ← individual picked at random from P with replacement
6:     if Fitness(Next) > Fitness(Best) then
7:         Best ← Next
8: return Best
9: end
```

2.7.2 Multiobjective Tournament Selection

In a problem where there is a bundle of objectives, we can tie them into a single fitness using some kind of linear function. But this requires us to come up with the degree to which one objective is worth another. This is hard to do, and may be close to impossible if the objectives are nonlinear. To solve the problem (having to come up with weights), we could instead abandon linear functions and simply treat the objectives as uncomparable functions. Some variants of such methods are Multiobjective Lexicographic Tournament Selection [Algorithm 8], Multiobjective Ratio Tournament Selection [Algorithm 9] and Multiobjective Majority Tournament Selection [Algorithm 10].

Algorithm 8 MULTIOBJECTIVE LEXICOGRAPHIC TOURNAMENT SELECTION

```
1:  Best ← individual picked at random from population with replacement
2:  O ← {O_1, ..., O_n} Objectives to assess with {In lexicographic order, most to least preferred.}
3:  t ← tournament size, t ≥ 1
4:  for i = 2 to t do
5:     Next ← individual picked at random from P with replacement
6:     for j = 1 to n do
7:        if ObjectiveValue(O_j, Next) > ObjectiveValue(O_j,Best) then {clearly superior}
8:           Best ← Next
9:           break from inner for
10:       else if ObjectiveValue(O_j, Next) < ObjectiveValue(O_j,Best) then {clearly superior}
11:          break from inner for
12: return Best
13: end
```

Algorithm 9 MULTIOBJECTIVE RATIO TOURNAMENT SELECTION

```
1:  Best ← individual picked at random from population with replacement
2:  O ← {O_1, ..., O_n} Objectives to assess with {In lexicographic order, most to least preferred.}
3:  t ← tournament size, t ≥ 1
4:  j ← random number picked uniformly from 1 to n
5:  for i = 2 to t do
6:     Next ← individual picked at random from P with replacement
7:     if ObjectiveValue(O_j, Next) > ObjectiveValue(O_j,Best) then {Clearly superior}
8:        Best ← Next
9:  return Best
10: end
```

Algorithm 10 MULTIOBJECTIVE MAJORITY TOURNAMENT SELECTION

```
1:  Best ← individual picked at random from population with replacement
2:  O ← {O_1, ..., O_n} Objectives to assess with {In lexicographic order, most to least preferred.}
3:  t ← tournament size, t ≥ 1
4:  for i = 2 to t do
5:     Next ← individual picked at random from P with replacement
6:  c ← 0
7:     for each objective O_j ∈ O do
8:        if ObjectiveValue(O_j, Next) > ObjectiveValue(O_j,Best) then
9:           c ← c + 1
10:       else if ObjectiveValue(O_j, Next) < ObjectiveValue(O_j,Best) then {clearly superior}
11:          c ← c − 1
12:    if c > 0 then
13:       Best ← Next
14: return Best
15: end
```

In *Multiobjective Lexicographic Tournament Selection* [Algorithm 8], basically when comparing two individuals, we run through the objectives (most important to least important) until we find one clearly superior to the other in that objective. Here, we have an Objective Value (objective, individual) function which tells us the quality of individual with regards to the given objective. Algorithm 9 picks an objective at random each time to use for fitness for this selection only and Algorithm 10 use voting: an individual is preferred if it is ahead in more objectives.

2.8 Non-Dominated Sorting Genetic Algorithm

Before going into details of such Genetic Algorithm, we need some definitions. Individual A *Pareto dominates* Individual B if A is at least as good as B in every objective and better than B in at least one objective. Computing pareto domination and binary tournament selection based on pareto domination is illustrated in Algorithm 11.

Algorithm 11 PARETO DOMINATION

```
1:  A ← individual A
2:  B ← individual B
3:  O ← {O_1, ..., O_n} Objectives to assess with
4:  a ← false
5:  j ← random number picked uniformly from 1 to n
6:  for each objective O_j ∈ O do
7:     if ObjectiveValue(O_j, A) > ObjectiveValue(O_j,B) then
8:        a ← true
9:     else if ObjectiveValue(O_j, A) < ObjectiveValue(O_j,B) then
10:       return false
11: return a
12: end
```

The set of individuals that can not pareto dominate each other lies in the same pareto front and individual on same pareto front has the same rank. The lower the rank the better the candidate solution is. So the best candidate solutions in a population have pareto front rank 1.

Non-Dominated Sorting is first proposed by N. Srinivas and Kalyanmoy Deb [31]. In this sorting approach, the population P is first partitioned into ranks, with each rank (a group of individuals) stored in the vector F. Then, a rank number is assigned to an individual (perhaps the individual gets it written internally somewhere). So the fitness of ith individual is, $fitness(i) = \frac{1}{1+ParetoFrontRank(i)}$.

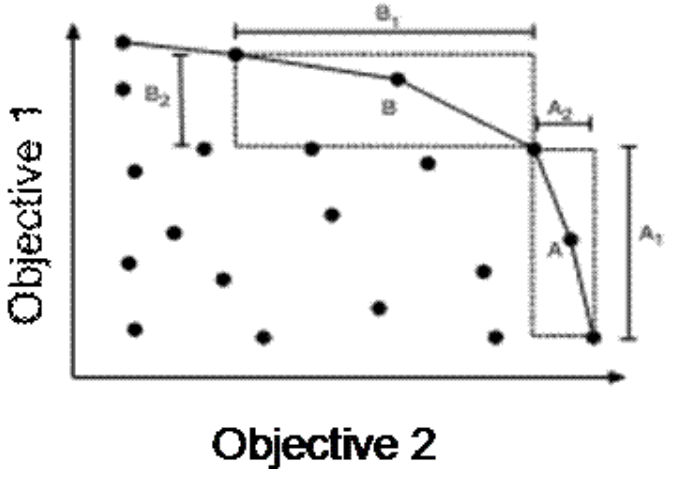

FIGURE 2.3: Sparsity of Individual B is higher than Individual A

While selection in NSGA, along with the fitness defined above another measure, namely, *sparsity* is used. It is assumed that, if the individuals in the population are being spread more evenly across the front, the population can serve the purpose of searching a good solution in a better way. The sparsity of an individual employs the following notion: an individual is in a more sparse region if the closest individuals on either side of it in its Pareto Front Rank are not too close to it (Please see the Figure 2.3). Individuals at the far ends of the Pareto Front Rank are assigned an infinite sparsity. As a selection procedure, the tournament selection is used where based on Pareto Front Rank individuals are selected first. If there is a tie, it is broken by using sparsity. These helps to get the individuals which are not only close to the true Pareto front, but also nicely spread out along it. Non-Dominated Sorting Lexicographic Tournament Selection With Sparsity is shown in Algorithm 12.

Algorithm 12 NON-DOMINATED SORTING LEXICOGRAPHIC TOURNAMENT SELECTION WITH SPARSITY

$P \leftarrow$ Population with Pareto Front Ranks assigned
$best \leftarrow$ individual picked at random from P with replacement
$t \leftarrow$ tournament size, $t \geq 1$
for $i = 2$ to t **do**
  $Next \leftarrow$ individual picked at random from P with replacement
  if ParetoFrontRank($Next$) < ParetoFrontRank($Best$) **then**
    $Best \leftarrow Next$
  else if ParetoFrontRank($Next$) = ParetoFrontRank($Best$) **then**
    if Sparsity($Next$) > Sparsity($Best$) **then**
      $Best \leftarrow Next$
return $Best$
end

Non-Dominated Sorting Genetic Algorithm keeps around all the best known individuals so far, in a sort of $(\mu + \lambda)$ or elitist notion. The general idea is to hold in P an archive of the best n individuals discovered so far. Then breeding a new population Q from P, and everybody in P and Q gets to compete for who gets to stay in the archive.

2.9 Summary

In summary, Chapter 2 has provided the readers an introduction to biological terms which are directly related to our problem. This introduction has prepared the readers to perceive the problem and related works on the same properly as briefed in the next chapter. Also in this chapter we discuss various metaheuristics techniques. The discussion of metaheuristics of this chapter also aids the reader to understand the meaning of terms and notations used in Chapter 4.

Chapter 3

Literature survey

The goal of this chapter is to present and discuss the related literature. Here, in our discussion we include previous works on the original problem we tackle in this thesis as well as some other works related to gene synthesis.

3.1 Previous Results on *URSPP*

URSPP is a real life problem. This has originated very recently from the laboratory while conducting viral genome synthesis. The computational version of the problem has come into the computer science surface through the very recent work of [3]. And to the best of our knowledge this is the only work on this problem in the literature. In this paper, the authors describes *URSPP* from the combinatorial viewpoint which is can be stated as follows. Given a text and a set of patterns, find a new text with maximum number of patterns inserted, ensuring minimum editing of letters and that the maximum distance between two consecutive inserted patterns (considering all insertions) is minimum. This is called the optimization version of URSPP. The decision version of the same problem, as defined in [3], asks whether it is possible to insert all patterns from a given set with gaps between patterns being at most K for a given integer, K.

The decision version of *URSPP* problem has been proved to be NP complete and it has been shown that the optimization version can not be approximated within a factor of 3/2 unless P=NP [3]. The best known result for the optimization version is a 2-approximation algorithm [3]. Also a *Dynamic Programming* (*DP*) algorithm with an exponential running time of $O(n^2 2^r)$ and some heuristic algorithms have been proposed

in [3]. Here, n is the number of *events*[1] and r is the number of unused restriction enzymes.

The authors in [3] provided two types of approximate implementations of the DP algorithm because of its exponential running time and memory requirement. In both of them the DP algorithm is run in turns considering consecutive blocks of enzymes one after another. In particular, at first the optimal placement is sought for the first block (set) of enzymes and then for the following block, and so on. However, in one implementation the order of considering the enzymes gives the highest possible insertion points (*forward* implementation) whereas the other gives the lowest possible insertion points (*backward* implementation).

The other heuristic versions are implemented first by eliminating all but one restriction site for each enzyme that appear in the genome initially and then inserting new restriction sites for enzymes which do not exist in the initial genome. To insert the enzymes they considered a *greedy* approach and the *maximal bipartite matching* technique. The *greedy* heuristic insertion selects unused restriction enzymes with least number of potential insertion points. The other approach, to insert enzymes, applies Hungarian algorithm [32] of *Weighted Bipartite Matching* on the weighted bipartite graph $G = (X \bigcup Y, E)$, where X is a set of unused restriction enzymes and Y is a list of ideal places where enzymes should be inserted. The weight of an edge $e \in E$ depends on the distance between an ideal place, $y \in Y$ and the location where $x \in X$ can be inserted.

The authors also built a sequence design tool called $PRESTO$ which contains implementation of four heuristics, *forward*, *backward*, *greedy* and *matching*. This tool takes genome sequence, location of genes and locked region as input. Along with the new sequence $PRESTO$ gives list of base pair changes and list of enzymes that are cutting. In each list, the location inside the genome sequence are also included.

[1] An event corresponds to either a location where a restriction enzyme is currently cutting or a place where an unused restriction enzyme can be inserted [3].

3.2 Related Works on Gene Synthesis

Although *URSPP* is a very recent problem, the works on synthetic gene design are not new in the literature. So, here we give a brief literature review on some of these works. Designing synthetic genes by hand is a time-consuming and error-prone process. In the past, researchers used to send off their requirements of the separate steps of synthetic gene design to a black box provided by a gene synthesis company and let it use its proprietary programs to design genes. Now-a-days these syntheses are relatively inexpensive commercialized services. It has become the most cost-effective, time and resource-saving method for obtaining nearly any desired DNA construct, outperforming conventional molecular biology techniques in many aspects from time and economization to expression performance, stability, and quality.

Blue Heron [33], OriGene [34], GeneArt [35] are some of the leading commercial vendors who continue to innovate with technologies to meet the growing demands of the researchers for gene synthesis. Also, a large number of tools and algorithms are being designed to provide a platform for synthetic gene design according to the user requirements.

Since 1999, Blue Heron has delivered tens of millions of base pairs of perfectly accurate genes to thousands of customers worldwide. Blue Heron can deliver to the customers one gene or one thousand according to the order from them. The options for such delivery ranges from the simplest sequence to comprehensive codon substitutions creating variants across hundreds of regions, . Now-a-days Blue Heron is a part of Origene.

OriGene Technologies was founded as a research tool company focused on the creation of the largest commercial collection of full-length human cDNAs[2] in a standard expression vector. OriGene Technologies uses high-throughput, genome wide approach to develop products for pharmaceutical, biotechnology, and academic research. Their flagship product is the cDNA clone collection, a searchable gene bank of over 30,000 human full-length TrueClone cDNA collection and over 25,000 TrueORF cDNA clones.

SiteFind is a free web-based software tool that enables the user to introduce a novel restriction site into the mutation primers without changing the peptide nucleotide sequence so that the site can be used as a marker for successful mutation [36]. In this software, there is an option for the user to choose a specific amino acid that should be

[2]Complementary DNA

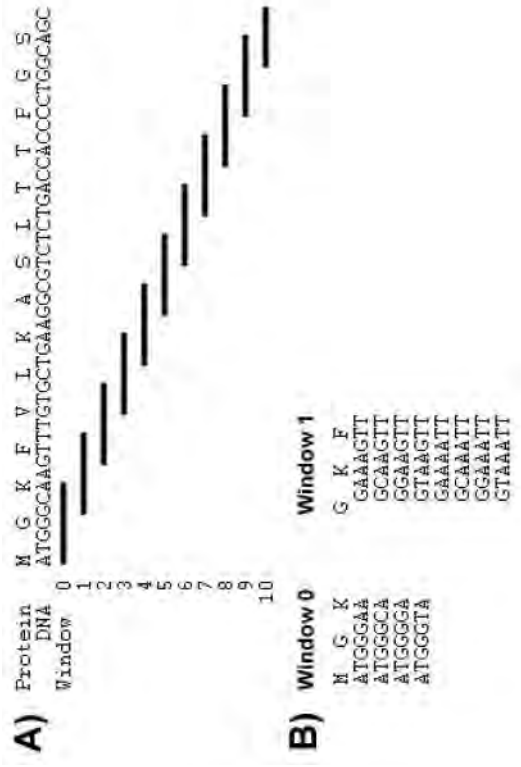

FIGURE 3.1: Moving window algorithm. a) Example of how the algorithm is implemented with a 4 nucleotide restriction site. Each window is therefore 7 nucleotides and each successive window is shifted forward 4 nucleotides, ensuring minimal overlap. b) Example of all the possible sequences generated for each of the first two search windows using the moving window algorithm. [This figure has been borrowed from [36].]

changed and to select the potential restriction site closest to the point mutation. Figures 3.1 to 3.3 show the interactive interface of the process

GeneJax is a JavaScript web application CAD tools for the parts extraction and visualization stages of the genome re-designing/refactoring process [37]. At this point we note that, unlike the introduction of a single restriction site, the problem we study in this thesis tries to introduce as many restriction sites as possible as part of the redesign process of a whole genome sequence. GeneJax has been inspried by google maps. It can map a local region from a large data set and parts of the sequence can also be manipulated (e.g. created, deleted, renamed, exported). Figure 3.5 shows the GeneJax user interface.

Refactored genomes offers a promising technological advance to better understand the structures and functions of DNA sequences [11]. Redesign of bacteriophage genomes is a recent approach that has been deployed to the T7 [12] and M13 phage (please refer

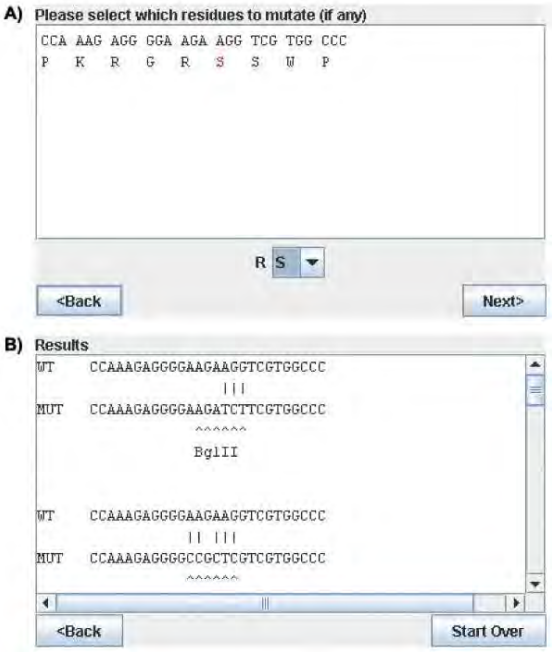

FIGURE 3.2: SiteFind Screenshots. a) Sample input, showing translated nucleotide sequence and a mutant residue highlighted in red. b) Sample output, showing a novel BglII site discovered within the sequence. [This figure has been borrowed from [36].]

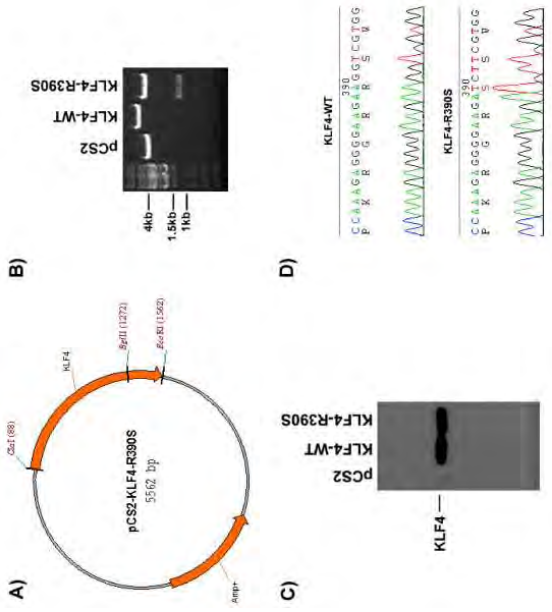

FIGURE 3.3: KLF4 R390S mutant has a novel BglII restriction site. a) pCS2-KLF4-R390S construct diagram. b) ClaI / BglII Restriction digest of both wild-type and successfully mutated plasmid DNA. c) a-Flag Western blot showing expression of mutant construct in 293T cells. d) Sequencing result of the mutation, mutated residue is highlighted in red. [This figure has been borrowed from [36].]

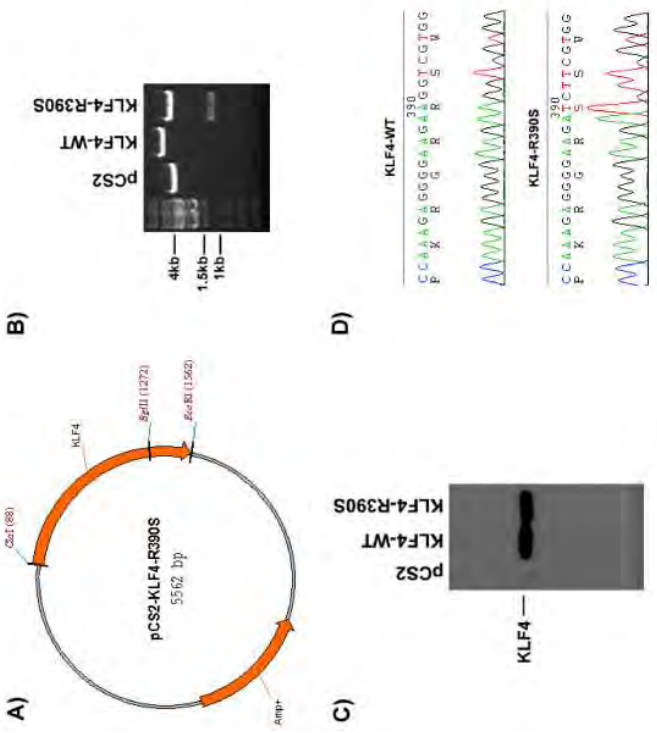

FIGURE 3.4: KLF4 K225/229R mutant has a novel NheI restriction site. e) pCS2-KLF4-K225/229R construct diagram. f) NheI / EcoRI Restriction digest of both wild-type and successfully mutated plasmid DNA. g) a-Flag Western blot showing expression of mutant construct in 293T cells. h) Sequencing result of the mutation, mutated residues are highlighted in red. [This figure has been borrowed from [36].]

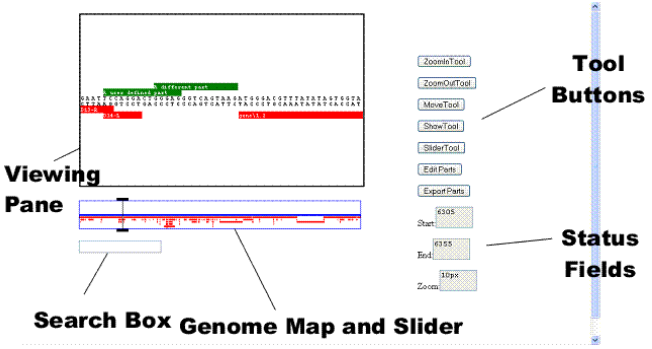

FIGURE 3.5: User Interface of GeneJax

to [11] and references therein). In [38] the author proposes an algorithm for optimally removing restriction sites against sets of cutter sequences.

3.3 Summary

To summarize, this chapter has introduced the readers to what this thesis deals with. Additionally, the previous works have been discussed here. We also have presented a gist of works conducted on gene synthesis. In the next chapter, we will present how we are going to solve *URSPP* using idea of metaheuristics techniques.

Chapter 4

URSPP in Synthetic Genomes by Metaheuristics

In this chapter, we present our algorithms. In particular, the application of various search techniques to obtain the desired genome sequence is illustrated in this chapter. At first we briefly present our motivation to apply metaheuristics. Then we elaborately discuss all the aspects of our algorithms: how we represent our solution, problem specific breeding operator, how to asses the fitness of a solution and the algorithmic steps.

4.1 Motivation behind Application of Metaheuristics

A disadvantage of heuristic methods is that they either generate only a very limited number of different solutions, or stop at poor quality local optima, which is the case for iterative improvement methods. In this thesis, to solve the *URSPP*, metaheuristics have been proposed with a goal to overcome these problems. A metaheuristic can be seen as a general-purpose heuristic method focusing towards the promising regions of the search space containing high-quality solutions. Metaheuristics have capability to escape being stuck at a local minimum.

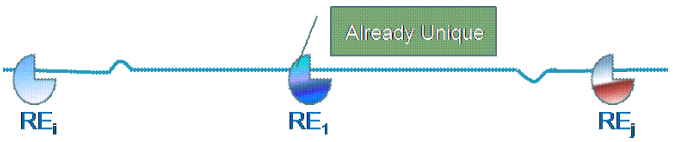

FIGURE 4.1: Restriction enzyme, RE_1 is already inserted

FIGURE 4.2: Deletion of RE_1 might allow to insert RE_2 and RE_3

To further understand our motivation to apply a metaheuristic below we discuss how heuristic approaches in the literature fails to cover many areas of the search space. We can divide the given set of restriction enzymes into three groups based on the existence of restriction sites in the given sequence. The first group consists of the enzymes that already have unique recognition sites in the original sequence. An enzyme having multiple sites in the sequence belongs to the second group and the enzymes belonging to the third group have no such sites in the given sequence. The heuristic algorithms of [3] considers the different groups of enzymes as follows. The first group of enzymes are considered as inserted already. For an enzyme in the second group, all but one occurrences of the restriction sites are deleted so that it gets a unique restriction site. Finally, for an enzyme belonging to the third group, one occurrence of the site is created which serve as the unique recognition site for it. Clearly, in such heuristic approaches, various (exponential number of) possibilities are completely ignored. For example, a deletion of enzyme of the first group may lead to more than one insertion of enzymes from the second and/or third group. Metaheuristics do not completely preclude the consideration of these possibilities and hence are more likely to achieve better solutions. This concept is illustrated in Figure 4.1 and 4.1. RE_1 in the only restriction enzyme between RE_i and RE_j and RE_1 belongs to the first group of enzymes, i.e., it has already been there in the genome sequence. None of the previous heuristics delete this enzyme. But due to the presence of this enzyme any other enzymes might not be allowed to insert. Now, if RE_1 is deleted it is possible that two other unique sites for RE_2 and RE_3 enzymes are being inserted. Here, the latter two enzymes belongs to third group. Hence, we have been strongly motivated to apply concepts of metaheuristics to solve *URSPP*.

4.2 Proposed Methodology

Before starting the main algorithm, we pre-process the given set of restriction enzymes to construct the *Restriction Map* [3]. *Restriction Map* is a data structure used to keep track of the list of restriction enzymes, each with its name and its recognition site.The map is built using a dictionary-matching algorithm, namely the Aho-Corasick algorithm [39] to efficiently find all occurrences of a finite set of patterns P in a given text. When

we get the sequence further processing is done as follows. A simple $O(nm)$ time algorithm is implemented to find all the possible places where a given restriction enzyme can be inserted, where n is the length of the sequence and m is the length of the recognition site. Now, for each restriction enzyme, all the occurrences from *Restriction Map* and all the possible insertion points are listed. This list is used as the potential insertion list while applying breeding operators on candidate solutions as will be described in the following sections.

4.2.1 Candidate Solution Representation

We represent each individual in a simple manner by a direct encoding of a fixed sized double vector. One sub-vector, namely $\mathcal{B}$, is boolean, which denotes the absence/presence of restriction enzymes and the other, namely $\mathcal{L}$, keeps track of the possible locations to insert the enzymes. Each position of the whole vector is dedicated for a particular restriction enzyme, referred to as a gene. The ith gene is denoted by g_i. We follow a left to right order while traversing a vector which means that the enzymes are inserted from left to right.

Because of the constraint about a locked region, it has to be avoided and the sites in locked regions are handled differently as follows. If the site in the locked region occurs once for a restriction enzyme, it is taken as inserted. If for any restriction enzyme more than one sites reside in the locked region, then, following the strategy of [3] it is assumed that the enzyme 'can never be inserted'. However, such different dealing of the sites in the locked region follows from the fact that we can not alter any base in these region. Rest of the restriction enzymes are used to construct candidate solutions. The size, N, of each candidate solution is equal to the number of restriction enzymes that are going to be inserted.

According to the sample candidate solution shown in Table 4.1, restriction enzyme, RE_1 is present and its location is ℓ_1. Note carefully that RE_2 in Table 4.1 is absent and hence the location ℓ_2 is currently insignificant. However, if the corresponding presence bit is turned on (e.g., due to mutation), ℓ_2 will be RE_2's location.

TABLE 4.1: One Candidate Solution

Meaning of notation	RE_1,	RE_2,	...	RE_i,	...
Gene for RE's	g_1	g_2	...	g_i	...
Presence or Absence, $\mathcal{B}$	1,	0,	...	1,	...
Possible location, $\mathcal{L}$	ℓ_1,	ℓ_2,	...	ℓ_i,	...

Here, the vector size, N is less than or equal to the cardinality of the given enzyme set, S. As we have discussed earlier that the restriction site occurring in the locked region is assumed to be either 'inserted' or 'can never be inserted' based on number of occurrence in the locked region. Rest of the restriction enzymes are used to construct candidate solutions and hence $N \leq |S|$.

4.2.2 Breeding Operators

FIGURE 4.3: Mutation 1

For breeding of new candidate solutions we use both recombination/crossover and mutation. For recombination we apply the standard two point crossover technique as follows. Say, N is the size of candidate solution vector. In *two point crossover*, we pick two numbers i and j, where $1 \leq i < j \leq N$ and swap the genes between them. Figure 4.3 shows an example of *two point crossover*. From biology we know that, properties, specially degeneracy, of the genetic code make a genome sequence more fault-tolerant for point mutations [40]. Hence, for mutation we apply a variation of the *point mutation*. Our mutation operators are problem specific and are described in Algorithms 13 and 14.

Algorithm 13 MUTATION1

```
1:  G ← < g_1, g_2, ..., g_N > vector to be mutated
2:  r ← random integer picked uniformly from 1 to N
3:  α ← probability of bit flip of presence
4:
5:  p ← pick a random value from 0.0 to 1 inclusive
6:  if p < α
7:      flip the presence bit, b_r
8:  else
9:      pick another random location from the list of potential insertion points
10:     update the location, l_r with newly chosen location
11: end
```

The process of mutation operations is further illustrated by Figure 4.4 and 4.5. According to Figure 4.4, an enzyme is randomly selected and with a certain probability the corresponding presence is flipped and/or another certain probability it's current location in genome sequence is updated. According to Figure 4.4, along with the flipping

Algorithm 14 MUTATION2

$G \leftarrow < g_1, g_2, \ldots, g_N >$ vector to be mutated
$r \leftarrow$ random integer picked uniformly from 1 to N
$0 : \alpha \leftarrow$ probability of bit flip of *presence*
$\alpha : \beta \leftarrow$ probability of modifying location *presence*
$\beta : 1 \leftarrow$ probability of changing the order of restriction site insertion

$p \leftarrow$ pick a random value from 0.0 to 1 inclusive
if $(0 < p < \alpha)$
    flip the presence bit, b_r
else if $(\alpha < p < \beta)$
    pick another random location from the list of potential insertion points
    update the location, l_r with newly chosen location
else
    construct new candidate solution, $G \leftarrow < g_r, g_{r+1}, \ldots, g_N, g_1, g_2, \ldots, g_{r-1} >$
end

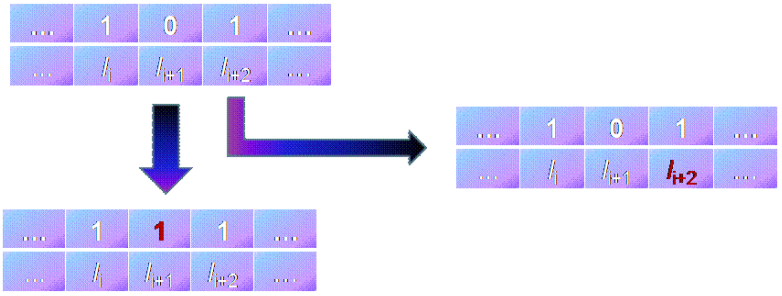

FIGURE 4.4: Mutation 1

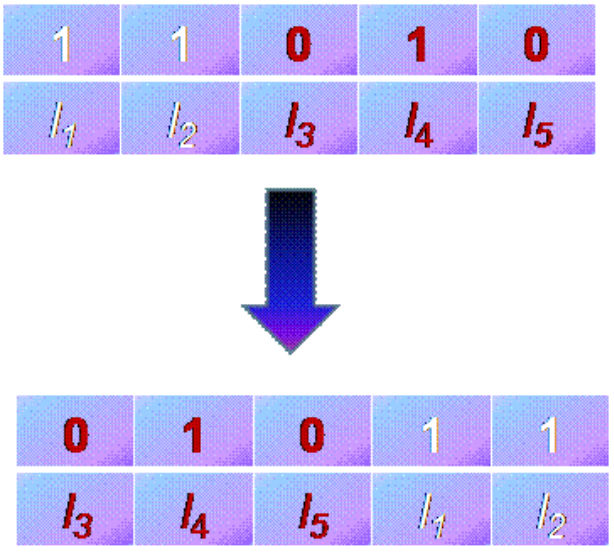

FIGURE 4.5: Mutation 2

of presence bit flip and/or updating of position vector, the order of enzyme insertions is also changed.

Here, the version of mutation described in Algorithm 14 considers the possibility of changing the order of restriction sites to be inserted. Therefore, it might allow a larger space to search. Notably, higher exploration can have both positive and negative effects on the results based on the intensity of exploration.

4.2.3 Quality Assessment

Recall that, our goal is to place unique restriction sites for as many restriction enzymes as possible allowing minimum number of base changes and to minimize the maximum gap between the consecutive sites. So, to determine the quality of a candidate solution, we have to consider three criteria: (1) number of unique sites, (2) number of base changes from the original genome sequence and (3) the maximum gap between the consecutive sites. We define, the following notations for these criteria:

f_1 = number of 1's in the boolean sub-vector, $\mathcal{B}$.

f_2 = number of positions where the original sequence differs from the synthesized sequence.

f_3 = the maximum among the distances between two consecutive restriction sites.

Fitness or quality of solution is higher when f_1 is higher and f_2 and f_3 are lower. A naive way to assess fitness could be to define the quality of a solution as a weighted sum of how well it meets various objectives. This approach is used to locally find a solution of better quality.

However, the linear parsimony pressure, which, in contrast to non-parametric parsimony pressure [41], is created by selection function when candidates are selected based on the size of fitness, and the difficulty of finding the degree to which one objective is worth another encourages us to treat the objectives as incomparable functions. To this end we plan to use variants of *tournament selection* to select highly qualified solution from a population. Basically, *tournament selection* is a non-parametric selection algorithm which returns the fittest ones among some t individuals picked at random, with replacement, from the population. Here, t is called the *tournament size*. The versions used for our algorithms are, *Multiobjective Lexicographic Tournament Selection*

(mlts), *Multiobjective Majority Tournament Selection (mmts)* and *Multiobjective Ratio Tournament Selection (mrts)*.

For *Multiobjective Lexicographic Tournament Selection*, the objectives are assumed to be ranked. A candidate solution is better if it is better with respect to a higher ranked objective. For this purpose, we assume that the rank ordering as, $f_3 > f_1 > f_2$. In brief the motivation behind this ordering is as follows. Clearly, to facilitate manipulation we need to insert large number of unique sites. However, if the insertion are uneven, then large area of the sequence may remain unexplored experimentally. These two statements leads us to the inequality that, $f_1 > f_2$ and $f_3 > f_1$. Combining them we get, $f_3 > f_1 > f_2$.

In *Multiobjective Majority Tournament Selection*, the candidate solution which is fitter with respect to most of the objectives is selected as more qualified and the *Multiobjective Ratio Tournament Selection* selects the better candidate solution with respect to a randomly chosen objective.

The other kind of fitness assessment we apply is the *Pareto domination*. Recall that, Individual A pareto dominates Individual B if A is at least as good as B in every objective and better than B in at least one objective. All individuals who can not pareto dominate each other forms pareto front of same rank (Please see the Figure 4.6). Pareto front of lowest rank gives the best individuals. So the fitness of ith individual is, $fitness(i) = \frac{1}{1+ParetoFrontRank(i)}$.

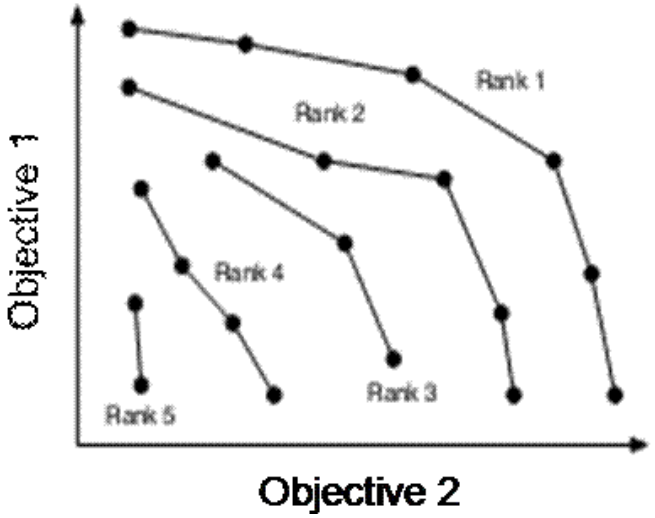

FIGURE 4.6: Pareto ranks

4.3 Algorithms

URSPP has more than one goal to achieve. As has been discussed in Chapter 4, to achieve more than one objectives, where the objectives are conflicting, use of MOEA is most effective. However, before applying MOEA we tested some local search techniques and hybrid genetic algorithms for our problem settings to probe the issues like, falling in local optima by local search techniques, jumping out of local optima using global search or exploration techniques etc.

4.3.1 Local Search Techniques

The local search algorithms considered in this work are simple Hill Climbing, Steepest Ascent hill Climbing and Steepest Ascent Hill Climbing with Replacement. Hill Climbing is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by incrementally changing a single element of the solution. If the change produces a better solution, an incremental change is made to the new solution, repeating until no further improvements can be found. In simple hill climbing, the first closer node is chosen, whereas in steepest ascent hill climbing all successors are compared and the closest to the solution is chosen. Steepest ascent hill climbing is similar to best-first search, which tries all possible extensions of the current path instead of only one. Both forms fail if there is no closer node, which may happen if there are local maxima in the search space which are not solutions. This failure can be avoided sometimes by Steepest Ascent Hill Climbing with Replacement technique. Unlike the former, the latter always replaces the current solution by the closest successor.

4.3.2 Hybrid Genetic algorithm

The genetic algorithms applied in URSPP are basic GA, GA with elitism and steady state GA. Each version is hybridized by a popular local improver, namely, hill-climbing. Among these, basic GA little-by-little selects a few parents and generates children until enough children have been created. To breed, we begin with an empty population of children. We then select two parents from the original population, copy them, cross them over with one another, and mutate the results. This forms two children, which we

then add to the child population. We repeat this process until the child population is entirely filled.

In GA with elitism we directly inject the fittest individual or individuals from the previous population into the next population. Finally, the steady-state approach updates the population in a piecemeal fashion rather than all at one time. The idea here is to iteratively breed a new child or two, assess their fitness, and then reintroduce them directly into the population itself, killing off some pre-existing individuals to make room for them.

4.3.3 Non-Dominated Sorting Genetic Algorithm

The previous algorithms attempt to merge objectives into one single fitness value by trading off one objective for another in some way. In NSGA we try to find which candidate solution pareto dominates other and thus extract only solutions consisting the pareto front. Since we return the lowest pareto front ranked solutions with sparsest individual, most of the local and global optima individuals can be found.

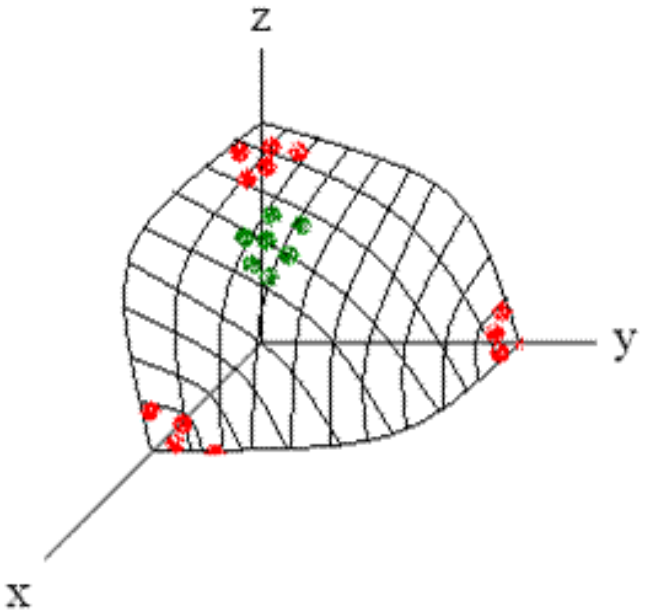

FIGURE 4.7: Selection of best solution from front

Recall that, in NSGA, the notion of sparsity has been injected with an aim to push the solutions towards the front. However, the *URSPP* has two conflicting goals which are f_1 and f_2. Therefore, if we try to increase the value of f_1, it is likely that the value of f_2 will increase also whereas our's target is to decrease f_2. Due to the presence of conflicting

goals, size of front becomes comparatively larger. Unlike traditional algorithm, we do not select a candidate solution arbitrarily from $BestFront$ as our best candidate. As shown in Figure 4.7, we always discard solutions (red colored) which has infinite sparsity. Instead, we return the solutions which lies somewhat middle of the front.

Note that, when a potential insertion point for a recognition site of a given enzyme is picked we may accidentally create a recognition site for another enzyme. Similar situation may happen when an enzyme with multiple occurrences are turned on and all but the selected locations are deleted. To avoid this undesirable effect, each crossover and mutation is accompanied/followed by a validate operation. In the validation operation, if the creation of restriction site for so far inserted enzyme is detected, we choose next possible insertion point from potential insertion list. If no such possibility can be found, the presence bit of corresponding restriction enzyme is turned off. The other constraint, like avoiding modifying locked regions and maintaining the amino acid sequence of genes, are handled while constructing the potential insertion point list of enzymes.

4.4 Summary

In this chapter we have presented our algorithms along with a clear description of how we represent candidate solution how to perform breeding and their assessment process. In the next chapter, we will manifest the performance of our algorithms in compared to other heuristics which exist in literature.

Chapter 5

Experimental Results

We have conducted extensive experiments to analyze the performance of our algorithm and to compare it with the other state of the art algorithm. This chapter presents our simulation results and related analysis with insight. In addition to a simple figurative comparison, we also investigate the statistical significance of our results with respect to other results. We start this chapter with a brief discussion of different statistical tests performaned here. Then we go to the simulations results.

5.1 Statistical Test

A statistical test provides a mechanism for making quantitative decisions about a process or processes. The intent is to determine whether there is enough evidence to "reject" a conjecture or hypothesis about the process. The conjecture is called the null hypothesis. Not rejecting may be a good result if we want to continue to act as if we *"believe"* that the null hypothesis is true. Or it may be a disappointing result, possibly indicating we may not yet have enough data to "prove" something by rejecting the null hypothesis.

5.1.1 Paired or Unpaired Test

When comparing two groups, it should be decided whether to use a paired test. When comparing three or more groups, the term paired is not apt and the term repeated measures is used instead.

An unpaired test should be used to compare groups when the individual values are not paired or matched with one another. A paired or repeated-measures test should be selected when values represent repeated measurements on one subject (before and after an intervention) or measurements on matched subjects. The paired or repeated-measures tests are also appropriate for repeated laboratory experiments run at different times, each with its own control.

We should select a paired test when values in one group are more closely correlated with a specific value in the other group than with random values in the other group. It is only appropriate to select a paired test when the subjects were matched or paired before the data were collected.

5.1.2 The T Test

To compare two paired values (such as in a before-after situation) where both observations are taken from the same or matched subjects, we can perform a paired t-test. The t-test assesses whether the means of two groups are statistically different from each other. This analysis is appropriate whenever you want to compare the means of two groups. We need to construct a null hypothesis - an expectation - which the experiment was designed to test.

5.1.3 Concept of null hypothesis

A classic use of a statistical test occurs in process control studies. For example, suppose we are interested in ensuring that photo masks in a production process have mean line widths of 500 micrometers. The null hypothesis, in this case, is that the mean line width is 500 micrometers. Implicit in this statement is the need to flag photo masks which have mean line widths that are either much greater or much less than 500 micrometers. This translates into the alternative hypothesis that the mean line widths are not equal to 500 micrometers. This is a two-sided alternative because it guards against alternatives in opposite directions; namely, that the line widths are too small or too large.

The testing procedure works this way. Line widths at random positions on the photo mask are measured using a scanning electron microscope. A test statistic is computed from the data and tested against pre-determined upper and lower critical values. If the test statistic is greater than the upper critical value or less than the lower critical value,

the null hypothesis is rejected because there is evidence that the mean line width is not 500 micrometers.

Null and alternative hypotheses can also be one-sided. For example, to ensure that a lot of light bulbs has a mean lifetime of at least 500 hours, a testing program is implemented. The null hypothesis, in this case, is that the mean lifetime is greater than or equal to 500 hours. The complement or alternative hypothesis that is being guarded against is that the mean lifetime is less than 500 hours. The test statistic is compared with a lower critical value, and if it is less than this limit, the null hypothesis is rejected. Thus, a statistical test requires a pair of hypotheses; namely,

H_0: a null hypothesis

H_a: an alternative hypothesis.

5.1.4 Significance levels

The null hypothesis is a statement about a belief. We may doubt that the null hypothesis is true, which might be why we are "testing" it. The alternative hypothesis might, in fact, be what we believe to be true. The test procedure is constructed so that the risk of rejecting the null hypothesis, when it is in fact true, is small. Here, the risk is often referred to as the significance level of the test and is denoted by α. By having a test with a small value of α, we feel that we have actually "proved" something when we reject the null hypothesis.

5.1.5 One or Two sided P Value

In statistical significance testing, the p-value is the probability of obtaining a test statistic at least as extreme as the one that was actually observed, assuming that the null hypothesis is true. One often "rejects the null hypothesis" when the p-value is less than the significance level α (Greek alpha), which is often 0.05 or 0.01. When the null hypothesis is rejected, the result is said to be statistically significant.

With many tests, we have to choose to calculate either a one- or two-sided P value (same as one- or two-tailed P value). As we know, the P value is calculated for the null hypothesis that the two population means are equal, and any discrepancy between the two sample means is due to chance. If this null hypothesis is true, the one-sided P

value is the probability that two sample means would differ as much as was observed (or further) in the direction specified by the hypothesis just by chance, even though the means of the overall populations are actually equal. The two-sided P value also includes the probability that the sample means would differ that much in the opposite direction (i.e., the other group has the larger mean). The two-sided P value is twice the one-sided P value.

A one-sided P value is appropriate when it can be stated with certainty (and before collecting any data) that there either will be no difference between the means or that the difference will go in a direction that is specified in advance (i.e., it is specified that which group will have the larger mean). If such specification can not be made the direction of any difference before collecting data, then a two-sided P value is more appropriate. If in doubt, a two-sided P value is selected.

5.1.6 Confidence Interval (C.I.)

In statistical estimation, a confidence interval (C.I.) is a kind of interval estimate which is used to indicate the reliability of an estimate. The selection of a confidence level for an interval determines the probability that the confidence interval produced will contain the true parameter value. Common choices for the confidence level C are 0.90, 0.95, and 0.99. These levels correspond to percentages of the area of the normal density curve. For example, a 95% confidence interval covers 95% of the normal curve – the probability of observing a value outside of this area is less than 0.05. Because the normal curve is symmetric, half of the area is in the left tail of the curve, and the other half of the area is in the right tail of the curve. As shown in the Figure 5.1, for a confidence interval with level C, the area in each tail of the curve is equal to (1-C)/2. For a 95% confidence interval, the area in each tail is equal to 0.05/2 = 0.025. Here, Z_H is upper critical level and $-Z_H$ is upper critical level.

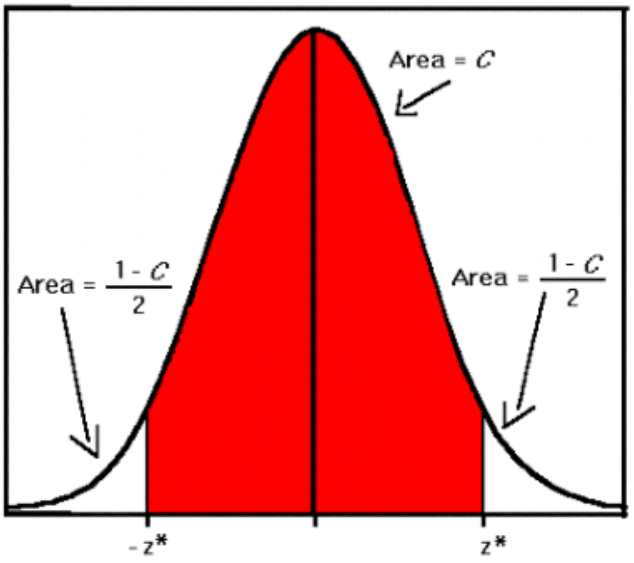

FIGURE 5.1: Confidence Interval

5.2 Simulation Results

5.2.1 Experimental set up and Representation

Our experiments were conducted on a computer having 3GHz Intel Pentium 4 processor with 1GB DDR3 Memory. The program was written in Java (JDK Version 1.6.0_26) and compiled on Netbeans IDE 7.0.1. We have run our experiments on a number of viral sequences obtained from the website of the National Center for Biotechnology Information [42] and a set of 145 restriction enzymes from the REBase restriction enzyme database [43]. Each algorithm with different types of tournament selection strategy is compared with previous heuristics with respect to the three objectives by taking average of multiple runs (around 20 times). To analyze the results, a paired two-sample t-tests has also been performed. The t-test assesses whether the means of two groups are statistically (significantly) different from each other. To compare two paired values (such as in a before-after situation) where both observations are taken from the same or matched subjects, a paired t-test is applied. A small P-value indicates that the result is significant [44, 45].

We present our results in a condensed form in the Tables 5.3 to 5.20 shown below. In these tables, we use some abbreviated names (e.g. existing heuristics, our proposals) whose meaning are given in the Table 5.1. Here, the existing algorithms from [3] have not been implemented by us, rather we have collected the software package from the authors [46]. Also viruses selected for our experiment are suggested by [3].

TABLE 5.1: Algorithms

Abbreviated Name	Explanation
Greedy	Heuristic algorithm of [3] using greedy search technique
Matching	Heuristic algorithm of [3] using weighted bipartite matching
Forward	Approximate Implementation of DP algorithm of [3] where enzymes of highet possible insertion points are considered first
Backward	Approximate Implementation of DP algorithm [3] where enzymes of least possible insertion points are considered first
HC	Hill Climbing
SAHC	Steepest Ascent Hill Climbing
SAHCwR	Steepest Ascent Hill Climbing with Replacement
Mutation type 1	Algorithm 13 of Chapter 5
Mutation type 2	Algorithm 14 of Chapter 5

TABLE 5.2: Difference between Objective value from proposed algorithm and objective value from existing algorithm

Objective Difference	Positive Value	Negative Value
Δf_1	Preferable	Not Preferable
Δf_2	Not Preferable	Preferable
Δf_3	Not Preferable	Preferable

The convention followed to present the results is as follows. For each objective, the difference between the objective values obtained by the proposed algorithm and that obtained by an existing algorithm is computed ($\Delta f_i, 1 \leq i \leq 3$). Then the mean and standard deviation (STD) of these differences are calculated and presented in the tables. For example, in Table 5.3 compares our Hill climbing algorithm with the different heuristics of [3] on λ Phage virus. Here, for mutation type 1, the first row gives the difference of mean values e.g. $\Delta f_1 = 6.415$ means Hill Climbing can insert 6.415 more restriction sites than *Greedy* heuristics, $\Delta f_2 = 12.32$ denotes Hill Climbing causes 12.32 base changes more than *Greedy* heuristics and $\Delta f_3 = -11.11$ indicates Hill Climbing provides genome sequence with 11.11 less f_3 than *Greedy* heuristic.

Along with the tabular representation, figures are drawn for better visualization. In all figures we have ploted mean values of each objectives for comparison.

Recall that, we want to insert larger number of enzymes with lower number of base changes having lower maximum gap between consecutive enzymes. So, our aim is to have higher f_1 and lower f_2 and f_3. Therefore, in Δf_1 column, positive mean value is preferred which means that the proposed algorithm can insert larger number of restriction sites (Please see Table 5.2). On the other hand, negative mean values are preferable

for Δf_2 and Δf_3. To elaborate, negative value in Δf_2 (Δf_3) column denotes that the proposed algorithm costs lower in terms of base changes (offers a synthesized sequence which has lower maximum distance of inserted enzymes). In the tables, along with the mean values and standard deviations of the objective value differences, we also present the confidence interval (C.I.) around the mean value using a 95% confidence level. Additionally, P-values are provided which indicates the statistical significance of our results.

5.2.2 Results summary and Analysis

Note that, the timing requirement of *forward* and *backward* implementations of DP algorithm ([3]) (4-5 minutes) has been found to be is much higher than that of other techniques (1-2 seconds). Therefore, in terms of timing, all other heuristics and meta-heuristics are superior to *forward* and *backward* implementation of DP algorithm. Since *URSPP* is an offline problem, the issue of performance is bigger than that of timing. Hence, we did not compare the algorithms from running time point of view. We write however that, the number of iterations used in each meta-heuristics are controlled in such a way that the average running time remains less than twice the running time of *greedy* and *matching* heuristics. Since within this period, the solution is converged. In Figure 5.2 an example of such convergence is shown.

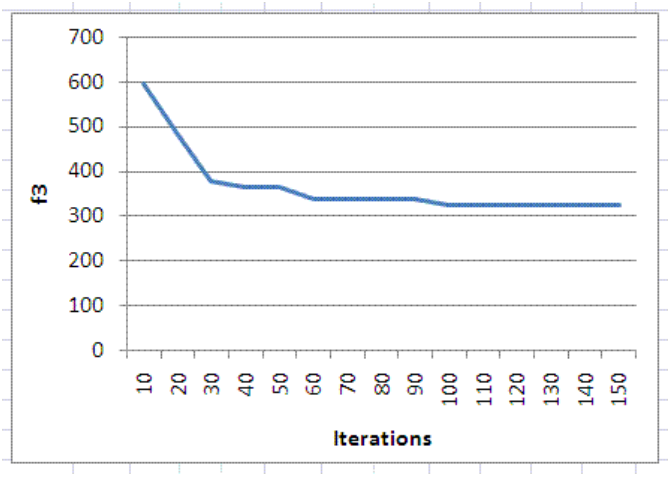

FIGURE 5.2: Convergence of Objective, f_3 for rubella virus in NSGA algorithm

The experimental results for λ Phage virus are summarized in Tables 5.3 through 5.12. Among these, Tables 5.3 to 5.5 presents comparison of local search algorithms with the existing heuristics of [3]. In these tables we find that, local search has provided us with a better synthesized sequence in terms of number of inserted restriction sites and

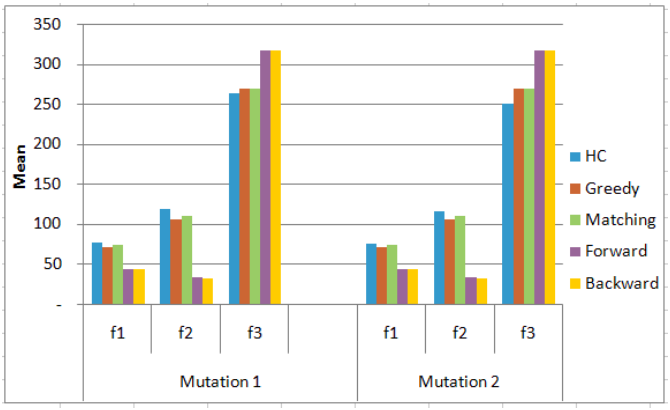

FIGURE 5.3: Comparison of Hill Climbing (HC) Algorithm with existing heuristics for λ Phage Virus (Also see Table 5.3)

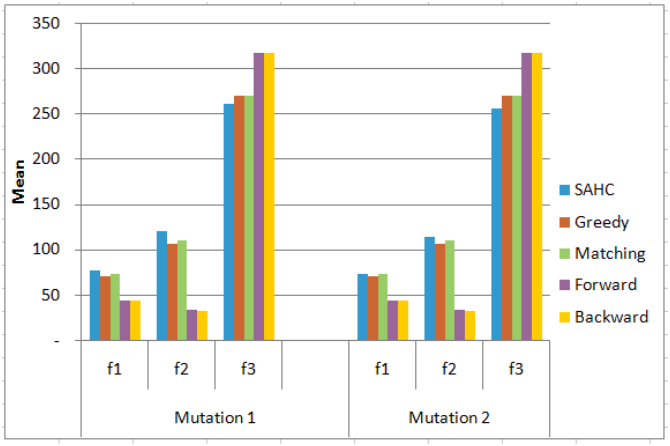

FIGURE 5.4: Comparison of Steepest Ascent Hill Climbing (SAHC) Algorithm with existing heuristics for λ Phage Virus (Also see Table 5.4)

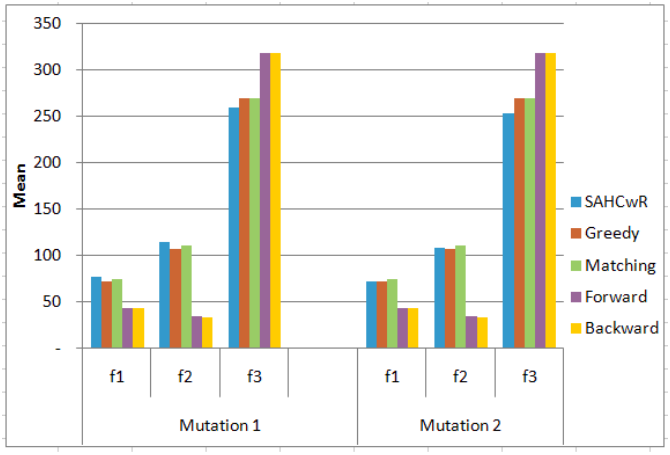

FIGURE 5.5: Comparison of Steepest Ascent Hill Climbing with Replacement (SAHCwR) Algorithm with existing heuristics for λ Phage Virus (Also see Table 5.5)

TABLE 5.3: Comparison of Hill Climbing (HC) Algorithm with existing heuristics for λ Phage Virus (Also see Figure 5.3)

Heuristic	Mutation type		Δf_1	Δf_2	Δf_3
Greedy	1	mean	6.415	12.32	-11.11
		STD	4.736	15.44	164.622
		C.I.	3.388	11.04	117.763
		P Value	0.002	0.033	0.83576
	2	mean	3.915	9.52	-39.11
		STD	4.102	10.73	159.833
		C.I.	2.935	7.679	114.338
		P Value	0.015	0.021	0.4589
Matching	1	mean	3.775	8.4	-11.05
		STD	4.509	13.19	144.7
		C.I.	3.226	9.433	103.512
		P Value	0.027	0.075	0.81459
	2	mean	1.275	5.6	-39.05
		STD	4.555	10.68	187.571
		C.I.	3.258	7.641	134.18
		P Value	0.399	0.132	0.5268
Forward	1	mean	34.2	85.1	-106.8
		STD	5.731	16.95	282.62
		C.I.	4.1	12.12	202.174
		P Value	2E-08	7E-08	0.26263
	2	mean	31.7	82.3	-134.8
		STD	5.519	14.86	309.109
		C.I.	3.948	10.63	221.123
		P Value	2E-08	3E-08	0.20118
Backward	1	mean	34.3	85.8	-106.8
		STD	5.599	15.98	282.62
		C.I.	4.005	11.43	202.174
		P Value	1E-08	4E-08	0.26263
	2	mean	31.8	83	-134.8
		STD	5.391	13.63	309.109
		C.I.	3.857	9.75	221.123
		P Value	2E-08	1E-08	0.20118

TABLE 5.4: Comparison of Steepest Ascent Hill Climbing (SAHC) Algorithm with existing heuristics for λ Phage Virus (Also see Figure 5.4)

Heuristic	Mutation type		Δf_1	Δf_2	Δf_3
Greedy	1	mean	6.215	14.22	-16.6
		STD	5.591	23.47	138.4
		C.I.	4	16.79	98.99
		P Value	0.007	0.088	0.713
	2	mean	1.515	7.62	-28.3
		STD	3.19	11.93	177.4
		C.I.	2.282	8.535	126.9
		P Value	0.167	0.074	0.626
Matching	1	mean	3.575	10.3	-16.6
		STD	4.66	20.69	152.7
		C.I.	3.334	14.8	109.2
		P Value	0.038	0.15	0.74
	2	mean	-1.13	3.7	-28.3
		STD	1.97	9.28	172.9
		C.I.	1.409	6.639	123.7
		P Value	0.104	0.239	0.618
Forward	1	mean	34	87	-112
		STD	4.295	16.52	206.8
		C.I.	3.072	11.82	147.9
		P Value	1E-09	5E-08	0.12
	2	mean	29.3	80.4	-124
		STD	2.869	14.7	303.9
		C.I.	2.053	10.51	217.4
		P Value	1E-10	3E-08	0.229
Backward	1	mean	34.1	87.7	-112
		STD	4.383	17.74	206.8
		C.I.	3.135	12.69	147.9
		P Value	1E-09	8E-08	0.12
	2	mean	29.4	81.1	-124
		STD	2.633	13.47	303.9
		C.I.	1.884	9.636	217.4
		P Value	6E-11	1E-08	0.229

TABLE 5.5: Comparison of Steepest Ascent Hill Climbing with Replacement (SAHCwR) Algorithm with existing heuristics for λ Phage Virus (Also see Figure 5.5)

Heuristic	Mutation type		Δf_1	Δf_2	Δf_3
Greedy	1	mean	5.915	8.02	-20.41
		STD	6.491	13.85	101.254
		C.I.	4.643	9.908	72.4328
		P Value	0.018	0.1	0.53972
	2	mean	0.015	1.32	-32.61
		STD	4.363	8.648	83.8268
		C.I.	3.121	6.186	59.9661
		P Value	0.992	0.641	0.24981
Matching	1	mean	3.275	4.1	-20.35
		STD	6.264	14.51	87.3954
		C.I.	4.481	10.38	62.5189
		P Value	0.133	0.395	0.48028
	2	mean	-2.63	-2.6	-32.55
		STD	3.237	10.11	85.5634
		C.I.	2.315	7.23	61.2084
		P Value	0.03	0.437	0.25967
Forward	1	mean	33.7	80.8	-116.1
		STD	7.379	17.95	220.404
		C.I.	5.279	12.84	157.667
		P Value	2E-07	2E-07	0.13011
	2	mean	27.8	74.1	-128.3
		STD	4.442	9.562	252.297
		C.I.	3.178	6.84	180.482
		P Value	1E-08	2E-09	0.14227
Backward	1	mean	33.8	81.5	-116.1
		STD	7.239	17.04	220.404
		C.I.	5.178	12.19	157.667
		P Value	1E-07	1E-07	0.13011
	2	mean	27.9	74.8	-128.3
		STD	4.408	9.09	252.297
		C.I.	3.154	6.502	180.482
		P Value	9E-09	9E-10	0.14227

maximum distance between consecutive sites. For instance, in Table 5.3, against greedy heuristic and Mutation type 1, the mean difference of f_1 (mean=6.415, STD=4.736, P Value=0.002) is significantly greater than zero. A 95% confidence interval (C.I.=3.388) around the mean of Δf_1 is [3.027, 9.803]. Both the lowest and highest values (lower and upper critical values) of Δf_1 are positive, i.e., the Δf_1 remains positive irrespective of the value of the variance. Recall that, positive Δf_1, negative Δf_2 and negative Δf_3 are preferable (Please see the Table 5.2).

Similarly, from values of Δf_2 we can say that, local search techniques find solution which is costly in terms of base changes. With respect to Δf_3 we also get better results though with lower significance. For example, in case of comparison with Greedy and Matching heuristics the P Value is high which does not reject the null hypothesis that the mean difference is zero. But when comparison is made with forward and backward implementation, the P value is much lower which proves higher significance of mean difference.

Notably, in most cases, local search algorithms using the Algorithm 14 (i.e., MUATION2) as the mutation operator shows better performance than those using the other version of it (Algorithm 13). This is due to the fact that Mutation type 2 allows more exploration. It is also worth-mentioning that, *SAHCwR* shows better performance that *SAHC* which is better than *HC*. This can also be attributed to increased exploration.

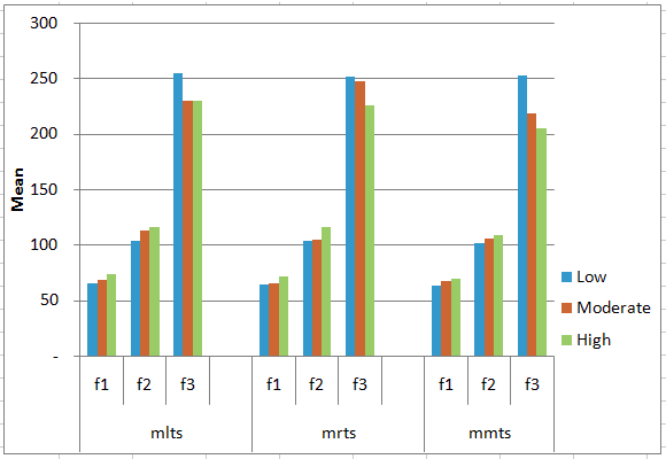

FIGURE 5.6: Comparison of Hybrid Genetic Algorithm using different level of local improver with existing heuristics for λ Phage Virus (Also see from Table 5.6 to 5.8)

For λ Phage virus, Tables 5.6 through 5.8 present the performance of Genetic Algorithm (GA) against the heuristics of [3] differing the level of local improver. The results show that even with multiobjective tournament selection, GA can not outperform others with respect to all objectives. In each of the cases GA offers better sequence only with

TABLE 5.6: Comparison of Hybrid Genetic Algorithm using low local improver with existing heuristics for λ Phage Virus (Also see Figure 5.6)

Using Low Local Improver					
Heuristic	Selection Type		Δf_1	Δf_2	Δf_3
Greedy	mlts	mean	-5.69	-2.78	-30
		STD	7.228	15.09	79.55
		C.I.	5.171	10.8	56.91
		P Value	0.035	0.575	0.263
	mrts	mean	-6.29	-2.48	-36.1
		STD	3.883	11.96	117.1
		C.I.	2.778	8.559	83.74
		P Value	6E-04	0.529	0.355
	mmts	mean	-7.29	-4.78	-33
		STD	4.554	14.17	124.2
		C.I.	3.257	10.14	88.82
		P Value	7E-04	0.314	0.422
Matching	mlts	mean	-8.33	-6.7	-30
		STD	7.123	15.31	82.79
		C.I.	5.095	10.95	59.23
		P Value	0.005	0.2	0.282
	mrts	mean	-8.93	-6.4	-36.1
		STD	4.18	9.814	117.1
		C.I.	2.99	7.021	83.79
		P Value	8E-05	0.069	0.356
	mmts	mean	-9.93	-8.7	-33
		STD	3.501	11.92	138.7
		C.I.	2.505	8.525	99.23
		P Value	9E-06	0.046	0.472
Forward	mlts	mean	22.1	70	-126
		STD	6.437	12.75	273.6
		C.I.	4.605	9.124	195.7
		P Value	2E-06	3E-08	0.18
	mrts	mean	21.5	70.3	-132
		STD	6.115	13.16	276.5
		C.I.	4.374	9.412	197.8
		P Value	1E-06	4E-08	0.166
	mmts	mean	20.5	68	-129
		STD	3.408	11.5	296.7
		C.I.	2.438	8.226	212.3
		P Value	1E-08	2E-08	0.203
Backward	mlts	mean	22.2	70.7	-126
		STD	6.596	13.7	273.6
		C.I.	4.719	9.797	195.7
		P Value	2E-06	5E-08	0.18
	mrts	mean	21.6	71	-132
		STD	5.967	12.1	276.5
		C.I.	4.268	8.657	197.8
		P Value	1E-06	2E-08	0.166
	mmts	mean	20.6	68.7	-129
		STD	3.307	10.49	296.7
		C.I.	2.365	7.503	212.3
		P Value	1E-08	2E-08	0.203

TABLE 5.7: Comparison of Hybrid Genetic Algorithm using moderate local improver with existing heuristics for λ Phage Virus (Also see Figure 5.6)

Using Moderate Local Improver					
Heuristic	Selection Type		Δf_1	Δf_2	Δf_3
Greedy	mlts	mean	-2.39	6.12	-77.8
		STD	4.327	6.434	139.3
		C.I.	3.096	4.603	99.64
		P Value	0.115	0.026	0.111
	mrts	mean	-5.79	-1.08	-43.7
		STD	3.892	13.92	50.19
		C.I.	2.784	9.956	35.9
		P Value	0.001	0.812	0.022
	mmts	mean	-5.83	-4.9	-101
		STD	4.944	10.24	68.15
		C.I.	3.537	7.324	48.75
		P Value	0.005	0.164	0.001
Matching	mlts	mean	-5.03	2.2	-77.8
		STD	4.786	8.215	146.3
		C.I.	3.424	5.876	104.7
		P Value	0.009	0.409	0.127
	mrts	mean	-8.43	-5	-43.7
		STD	3.399	9.943	62.93
		C.I.	2.432	7.113	45.02
		P Value	3E-05	0.146	0.056
	mmts	mean	-5.83	-4.9	-101
		STD	4.944	10.24	68.15
		C.I.	3.537	7.324	48.75
		P Value	0.005	0.164	0.001
Forward	mlts	mean	25.4	78.9	-174
		STD	4.671	9.41	292.7
		C.I.	3.342	6.731	209.4
		P Value	3E-08	1E-08	0.094
	mrts	mean	22	71.7	-139
		STD	3.464	11.62	249.9
		C.I.	2.478	8.315	178.8
		P Value	9E-09	1E-08	0.112
	mmts	mean	24.6	71.8	-197
		STD	6.769	15.17	214.3
		C.I.	4.842	10.85	153.3
		P Value	1E-06	1E-07	0.017
Backward	mlts	mean	25.5	79.6	-174
		STD	4.453	7.734	292.7
		C.I.	3.186	5.533	209.4
		P Value	2E-08	2E-09	0.094
	mrts	mean	22.1	72.4	-139
		STD	3.381	10.66	249.9
		C.I.	2.419	7.625	178.8
		P Value	7E-09	5E-09	0.112
	mmts	mean	24.7	72.5	-197
		STD	6.684	14.34	214.3
		C.I.	4.782	10.26	153.3
		P Value	1E-06	6E-08	0.017

TABLE 5.8: Comparison of Hybrid Genetic Algorithm using high local improver with existing heuristics for λ Phage Virus (Also see Figure 5.6)

Using High Local Improver					
Heuristic	Selection Type		Δf_1	Δf_2	Δf_3
Greedy	mlts	mean	3.015	12.72	-79.2
		STD	7.595	17.9	121
		C.I.	5.433	12.81	86.58
		P Value	0.241	0.098	0.068
	mrts	mean	0.415	105.7	395.2
		STD	4.489	32.48	203.3
		C.I.	3.212	23.24	145.4
		P Value	0.777	0.052	0.005
	mmts	mean	-1.19	2.72	-129
		STD	4.079	9.395	75.77
		C.I.	2.918	6.721	54.2
		P Value	0.382	0.384	4E-04
Matching	mlts	mean	0.375	8.8	-79.2
		STD	7.299	14.59	103.9
		C.I.	5.221	10.44	74.3
		P Value	0.875	0.16	0.039
	mrts	mean	-2.23	6	-86.2
		STD	3.36	11.6	68.8
		C.I.	2.403	8.3	49.22
		P Value	0.066	0.136	0.003
	mmts	mean	-3.83	-1.2	-129
		STD	4.176	8.983	69.31
		C.I.	2.988	6.426	49.58
		P Value	0.018	0.683	2E-04
Forward	mlts	mean	30.8	85.5	-175
		STD	8.297	18.08	193.5
		C.I.	5.935	12.93	138.4
		P Value	9E-07	2E-07	0.019
	mrts	mean	28.2	82.7	-182
		STD	3.994	10.4	236.2
		C.I.	2.857	7.442	168.9
		P Value	3E-09	1E-09	0.038
	mmts	mean	26.6	75.5	-225
		STD	5.562	13.65	259.1
		C.I.	3.979	9.763	185.3
		P Value	1E-07	3E-08	0.023
Backward	mlts	mean	30.9	86.2	-175
		STD	8.13	16.83	193.5
		C.I.	5.816	12.04	138.4
		P Value	8E-07	6E-08	0.019
	mrts	mean	28.3	83.4	-182
		STD	3.831	10.12	236.2
		C.I.	2.741	7.242	168.9
		P Value	2E-09	9E-10	0.038
	mmts	mean	26.7	76.2	-225
		STD	5.376	12.15	259.1
		C.I.	3.846	8.695	185.3
		P Value	8E-08	1E-08	0.023

TABLE 5.9: Comparison of Hybrid GA with Elitism with existing heuristics for λ Phage Virus (Also see Figure 5.7)

Heuristic	Selection		f1	f2	f3
Greedy	mlts	mean	-11.3	-24.6	65.99
		STD	5.088	12.95	127.831
		C.I.	3.64	9.261	91.445
		P Value	6E-05	1E-04	0.13702
	mrts	mean	-10.4	-16.5	22.19
		STD	2.839	9.804	104.043
		C.I.	2.031	7.014	74.4281
		P Value	1E-06	5E-04	0.51698
	mmts	mean	-10.8	-21.9	13.59
		STD	4.447	8.987	94.1173
		C.I.	3.181	6.429	67.3274
		P Value	3E-05	3E-05	0.65877
Matching	mlts	mean	-13.9	-28.5	66.05
		STD	4.707	10.52	93.9951
		C.I.	3.367	7.528	67.24
		P Value	6E-06	7E-06	0.05338
	mrts	mean	-13	-20.4	22.25
		STD	2.605	9.321	101.939
		C.I.	1.864	6.668	72.9227
		P Value	7E-08	7E-05	0.50746
	mmts	mean	-13.4	-25.8	13.65
		STD	3.78	6.746	83.2349
		C.I.	2.704	4.826	59.5427
		P Value	1E-06	7E-07	0.61655
Forward	mlts	mean	16.5	48.2	-29.7
		STD	3.923	8.904	271.913
		C.I.	2.806	6.37	194.515
		P Value	3E-07	6E-09	0.73773
	mrts	mean	17.4	56.3	-73.5
		STD	4.789	11.02	272.094
		C.I.	3.426	7.88	194.644
		P Value	1E-06	6E-08	0.41514
	mmts	mean	17	50.9	-82.1
		STD	6.394	10.43	193.616
		C.I.	4.574	7.461	138.505
		P Value	1E-05	0.012	0.0004
Backward	mlts	mean	16.6	48.9	-29.7
		STD	4.061	9.146	271.913
		C.I.	2.905	6.543	194.515
		P Value	4E-07	1E-08	0.73773
	mrts	mean	17.5	57	-73.5
		STD	4.649	10.42	272.094
		C.I.	3.326	7.457	194.644
		P Value	8E-07	3E-08	0.41514
	mmts	mean	17.1	51.6	-82.1
		STD	6.226	9.812	193.616
		C.I.	4.454	7.019	138.505
		P Value	1E-05	5E-08	0.21281

TABLE 5.10: Comparison of Hybrid Steady State GA with existing heuristics for λ Phage Virus (Also see Figure 5.8)

Heuristic	Selection		f1	f2	f3
Greedy	mlts	mean	-4.49	1.32	-114
		STD	4.592	6.689	55.58
		C.I.	3.285	4.785	39.76
		P Value	0.013	0.602	1E-04
	mrts	mean	-7.89	-3.08	21.09
		STD	3.657	6.873	138.3
		C.I.	2.616	4.917	98.92
		P Value	8E-05	0.19	0.641
	mmts	mean	-7.29	-4.98	-28.2
		STD	3.598	9.269	186.5
		C.I.	2.574	6.631	133.4
		P Value	1E-04	0.124	0.644
Matching	mlts	mean	-7.13	-2.6	-114
		STD	4.441	12.46	56.17
		C.I.	3.177	8.916	40.18
		P Value	7E-04	0.604	1E-04
	mrts	mean	-10.5	-7	21.15
		STD	3.511	8.223	144.4
		C.I.	2.511	5.882	103.3
		P Value	6E-06	0.025	0.654
	mmts	mean	-9.93	-8.9	-28.2
		STD	2.653	6.935	209.1
		C.I.	1.898	4.961	149.6
		P Value	9E-07	0.003	0.68
Forward	mlts	mean	23.3	74.1	-210
		STD	4.523	12.92	218.7
		C.I.	3.235	9.244	156.5
		P Value	5E-08	2E-07	0.014
	mrts	mean	19.9	69.7	-74.6
		STD	5.065	8.982	211.6
		C.I.	3.623	6.425	151.4
		P Value	6E-07	1E-09	0.294
	mmts	mean	20.5	67.8	-124
		STD	5.642	11.2	300.6
		C.I.	4.036	8.014	215
		P Value	1E-06	1E-08	0.225
Backward	mlts	mean	23.4	74.8	-210
		STD	4.452	11.9	218.7
		C.I.	3.185	8.51	156.5
		P Value	5E-08	1E-07	0.014
	mrts	mean	20	70.4	-74.6
		STD	4.83	7.336	211.6
		C.I.	3.456	5.248	151.4
		P Value	4E-07	2E-10	0.294
	mmts	mean	20.6	68.5	-124
		STD	5.502	9.969	300.6
		C.I.	3.936	7.132	215
		P Value	9E-07	4E-09	0.225

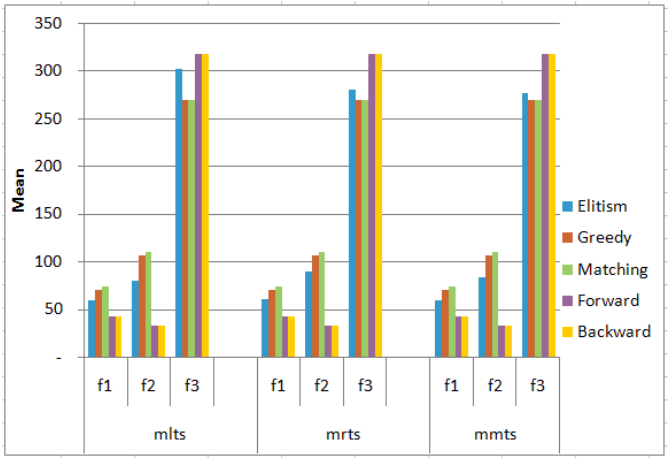

FIGURE 5.7: Comparison of Hybrid GA with Elitism with existing heuristics for λ Phage Virus (Also see Table 5.9)

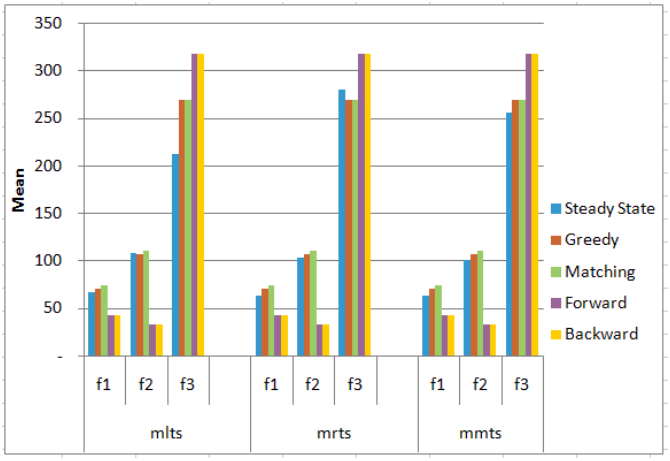

FIGURE 5.8: Comparison of Hybrid Steady State GA with existing heuristics for λ Phage Virus (Also see Table 5.10)

respect to f_3 which is our prime concern (Please see Subsection 4.2.3 of Chapter 4 for explanation).

Although global search technique like GA should outperform local search techniques like Hill Climbing, in our experiment we have found the reverse result. This might be attributed to the fact that GA lacks in the degree of exploitation and *URSPP* requires more exploitation than GA can provide. *URSPP* is a discrete optimization problem. For discrete optimization problem the extent of exploration required to avoid premature convergence is not very high. Notably, in our experiments, the explorative components of local searches (e.g., Mutation type 2 and Steepest Ascent versions) has provided quite good results (so as to outperform GA), we believe that the extent of exploration here does not exceed the requirement which is necessary to avoid premature convergence for *URSPP*. Hence, we also apply more genetic algorithm which exploits the search space

more that the other genetic algorithms. In particular, we inject the notion of elitism (to increase exploitation) and also employ the steady state version of GA.

We show the results using GA with Elitism and Steady State GA in Tables 5.9 and 5.10 respectively. However, as can be realized, as unfortunate as it seems, the results are not satisfactory. Despite our attempt to inject exploitation, a global search technique like GA,From literature we know that global search techniques better perform than local search techniques. But our experiments deviates from this fact. This deviation might have the following reasons.

As breeding operator we use crossover in our global search, whereas local search techniques use point mutation. From biology we know that, properties, specially degeneracy, of the genetic code make a genome sequence more fault-tolerant for point mutations [40]. Hence subsequently we plan to apply a algorithm where only point mutation will be used as breeding operator.

The two objectives, f_1 and f_2 are inversely related to each other, i.e., higher insertion decreases the chance of lower costs (base change). Therefore, as we increase the number of insertions of restriction sites using the local improver, the performance with respect to f_1 increases whereas the same with respect to f_2 decreases. Due to this conflicting dependency, multi-objective tournament selection alone can not provide very high quality solutions. To get even better results we move to another approach, namely NSGA (as discussed in Chapter 5) where separate concentration on objectives are more prominent.

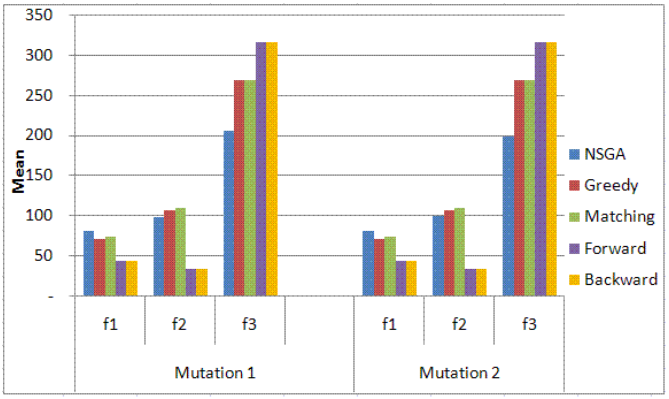

FIGURE 5.9: Comparison of Non-dominated Sorting Genetic Algorithm with existing heuristics for λ Phage virus (Also see Table 5.11)

Tables 5.11 and 5.12 present comparison of *NSGA* with the heuristics of [3] and different local search techniques of [47], namely, Hill Climbing (HC) and Steepest Ascent Hill Climbing with (SAHC) or without (SAHCwR) Replacement respectively.

TABLE 5.11: Comparison of Non-dominated Sorting Genetic Algorithm with existing heuristics for λ Phage virus (Also see Figure 5.9)

Heuristic	Mutation Type		Δf_1	Δf_2	Δf_3
Greedy	1	mean	9.615	-7.58	-126
		STD	6.038	11.13	77.3
		C.I.	4.319	7.96	55.3
		P Value	7E-04	0.06	6E-04
	2	mean	9.615	-6.38	-140
		STD	3.66	12.34	63.39
		C.I.	2.618	8.828	45.35
		P Value	2E-05	0.137	6E-05
Matching	1	mean	6.975	-11.5	-126
		STD	5.729	9.354	56.46
		C.I.	4.098	6.691	40.39
		P Value	0.004	0.004	6E-05
	2	mean	6.975	-10.3	-140
		STD	3.675	7.808	91.07
		C.I.	2.629	5.586	65.15
		P Value	2E-04	0.002	9E-04
Forward	1	mean	37.4	65.2	-222
		STD	6.535	7.052	217
		C.I.	4.675	5.045	155.3
		P Value	2E-08	3E-10	0.01
	2	mean	37.4	66.4	-236
		STD	4.169	7.792	256
		C.I.	2.982	5.574	183.2
		P Value	4E-10	6E-10	0.017
Backward	1	mean	37.5	65.9	-222
		STD	6.311	6.19	217
		C.I.	4.515	4.428	155.3
		P Value	2E-08	9E-11	0.01
	2	mean	37.5	67.1	-236
		STD	4.007	6.919	256
		C.I.	2.866	4.95	183.2
		P Value	3E-10	2E-10	0.017

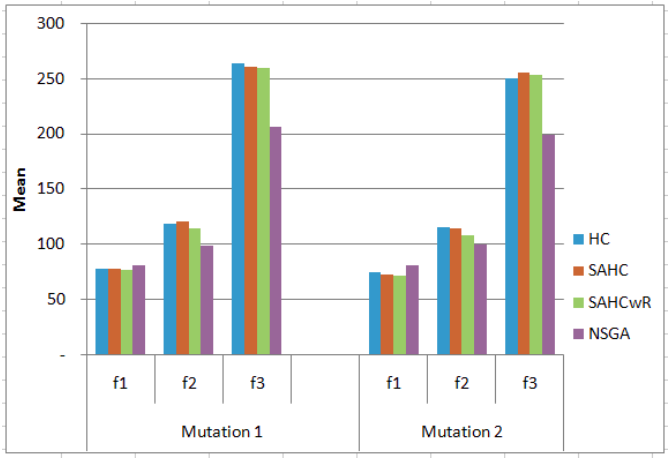

FIGURE 5.10: Comparison of Non-dominated Sorting Genetic Algorithm with local search techniques for λ Phage virus (Also see Table 5.12)

TABLE 5.12: Comparison of Non-dominated Sorting Genetic Algorithm with local search techniques for λ Phage virus (Also see Figure 5.10)

Heuristic	Mutation Type		Δf_1	Δf_2	Δf_3
HC	1	mean	3.2	-19.9	-115
		STD	4.917	13.7	152.2
		C.I.	3.517	9.799	108.8
		P Value	0.07	0.001	0.041
	2	mean	5.7	-15.9	-101
		STD	4.448	14.47	134.2
		C.I.	3.182	10.35	96.02
		P Value	0.003	0.007	0.042
SAHC	1	mean	3.4	-21.8	-109
		STD	8.072	18.55	157.8
		C.I.	5.774	13.27	112.8
		P Value	0.216	0.005	0.056
	2	mean	8.1	-14	-112
		STD	3.143	13.09	197.3
		C.I.	2.248	9.364	141.1
		P Value	2E-05	0.008	0.107
SAHCwR	1	mean	3.7	-15.6	-106
		STD	5.774	17.35	61.5
		C.I.	4.131	12.41	43.99
		P Value	0.073	0.019	4E-04
	2	mean	9.6	-7.7	-107
		STD	5.91	11.43	105.4
		C.I.	4.228	8.178	75.42
		P Value	6E-04	0.062	0.011

Table 5.11 shows that NSGA gives us better synthesized sequence than *greedy* and *matching* heuristics requiring lower number of base changes. Also the average number of inserted restrictions sites is higher and maximum gap is smaller. For example, against greedy heuristic and mutation type 1, the mean difference of f_1 (mean=9.615, STD=6.038, P Value=7E-04) is significantly greater than zero. A 95% confidence interval (C.I.=4.319) around the mean of Δf_1 is [5.296, 13.934]. Both the lowest and highest values of Δf_1 are positive, i.e., Δf_1 remains positive irrespective of the value of the variance. Similar inference holds for f_2 and f_3 except for f_2, a small part of the 95% confidence interval [0.38, -15.54] falls in the positive range. Recall that, positive Δf_1, negative Δf_2 and negative Δf_3 are better (Please see the Table 5.2). However, NSGA, is still costlier (in terms of base changes) than *forward* and *backward* implementation, though it is superior to those in terms of f_1, f_3 and execution time. Notably, although we did not compare the execution time, NAGS is found to be quite fast.

Table 5.12 shows that NSGA is less costlier (in terms of base changes) than local search techniques keeping a very good maximum gap between the restriction sites. The base changes are less in number because it intelligently inserts a slightly less number of restriction sites than others.

Tables 5.13 through 5.30 and Figure 5.11 through 5.28 focus on similar comparisons considering other viruses.

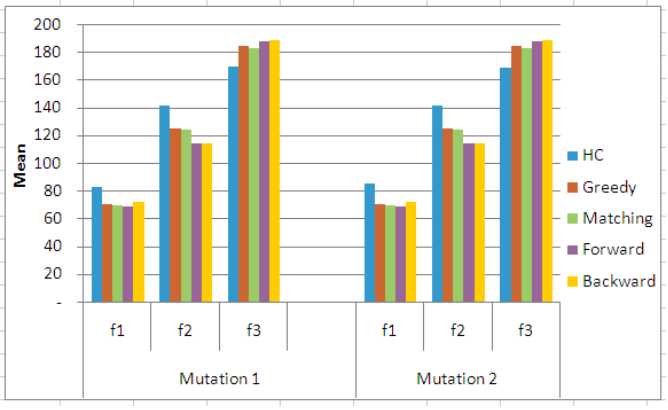

FIGURE 5.11: Comparison of Hill Climbing (HC) Algorithm with existing heuristics for Polio Virus (Also see Table 5.13)

TABLE 5.13: Comparison of Hill Climbing (HC) Algorithm with existing heuristics for Polio Virus (Also see Figure 5.11)

Heuristic	Mutation type		Δf_1	Δf_2	Δf_3
Greedy	1	mean	14.86	16.43	-30.075
		STD	2.947	8.217	33.753
		C.I.	2.108	8.512	24.1454
		P Value	7E-08	1E-04	0.02012
	2	mean	12.36	16.73	-29.475
		STD	4.363	11.97	51.9854
		C.I.	3.121	8.563	37.1881
		P Value	9E-06	0.002	0.10657
Matching	1	mean	15.34	17.34	-27.9
		STD	2.559	7.746	37.7271
		C.I.	1.831	7.429	26.9883
		P Value	1E-08	6E-05	0.04412
	2	mean	12.84	17.64	-27.3
		STD	4.586	11.94	28.3819
		C.I.	3.281	8.542	20.3032
		P Value	1E-05	0.001	0.01398
Forward	1	mean	16.3	26.9	-38.1
		STD	3.368	8.66	82.2117
		C.I.	2.409	8.075	58.8107
		P Value	9E-08	4E-06	0.17682
	2	mean	13.8	27.2	-37.5
		STD	5.203	13.68	93.0666
		C.I.	3.722	9.784	66.5758
		P Value	2E-05	1E-04	0.23451
Backward	1	mean	13.2	26.8	-39.9
		STD	9.283	8.324	83.4059
		C.I.	6.641	8.424	59.665
		P Value	0.001	3E-06	0.16463
	2	mean	10.7	27.1	-39.3
		STD	9.405	12.98	88.8683
		C.I.	6.728	9.287	63.5725
		P Value	0.006	1E-04	0.19548

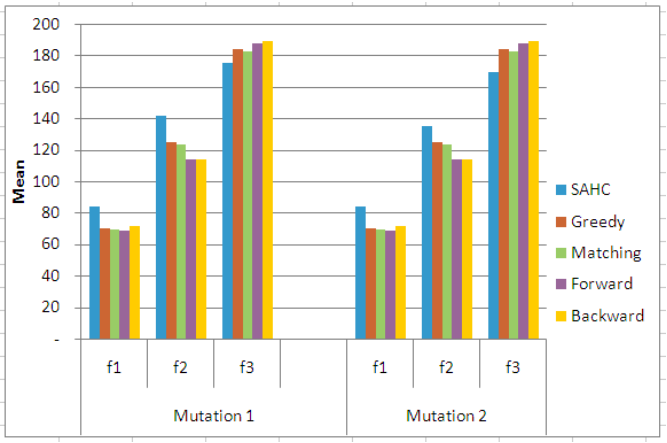

FIGURE 5.12: Comparison of Steepest Ascent Hill Climbing (SAHC) Algorithm with existing heuristics for Polio Virus (Also see Table 5.14)

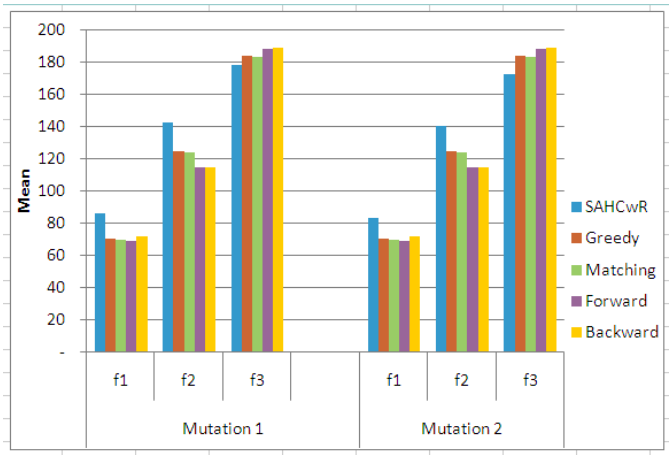

FIGURE 5.13: Comparison of Steepest Ascent Hill Climbing with Replacement (SAHCwR) Algorithm with existing heuristics for Polio Virus (Also see Table 5.15)

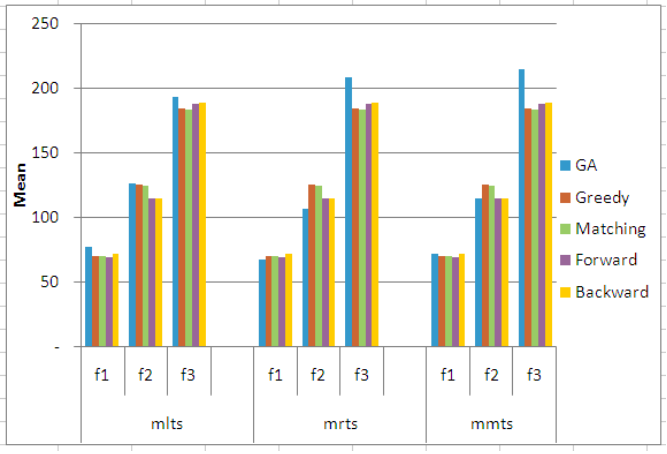

FIGURE 5.14: Comparison of Hybrid Genetic Algorithm using moderate local improver with existing heuristics for Polio Virus (Also see Table 5.16)

TABLE 5.14: Comparison of Steepest Ascent Hill Climbing (SAHC) Algorithm with existing heuristics for Polio Virus (Also see Figure 5.12)

Heuristic	Mutation type		Δf_1	Δf_2	Δf_3
Greedy	1	mean	13.66	10.53	-29.8
		STD	2.99	19.44	47.22
		C.I.	2.139	13.91	33.78
		P Value	2E-07	0.121	0.077
	2	mean	13.76	16.93	-18.1
		STD	3.418	7.469	49.26
		C.I.	2.445	5.343	35.24
		P Value	5E-07	5E-05	0.276
Matching	1	mean	14.14	11.44	-27.6
		STD	2.406	21.03	36.13
		C.I.	1.721	15.04	25.85
		P Value	2E-08	0.12	0.039
	2	mean	14.24	17.84	-15.9
		STD	3.826	7.563	47.89
		C.I.	2.737	5.41	34.26
		P Value	9E-07	4E-05	0.321
Forward	1	mean	15.1	21	-37.8
		STD	3.929	22.84	97.27
		C.I.	2.81	16.34	69.58
		P Value	7E-07	0.017	0.25
	2	mean	15.2	27.4	-26.1
		STD	3.645	5.232	85.18
		C.I.	2.608	3.743	60.93
		P Value	3E-07	5E-08	0.358
Backward	1	mean	12	20.9	-39.6
		STD	10.22	22.67	96.49
		C.I.	7.311	16.22	69.03
		P Value	0.005	0.017	0.227
	2	mean	12.1	27.3	-27.9
		STD	8.937	5.438	87.34
		C.I.	6.393	3.89	62.48
		P Value	0.002	7E-08	0.339

TABLE 5.15: Comparison of Steepest Ascent Hill Climbing with Replacement (SAHCwR) Algorithm with existing heuristics for Polio Virus (Also see Figure 5.13)

Heuristic	Mutation type		Δf_1	Δf_2	Δf_3
Greedy	1	mean	15.76	17.53	-12.075
		STD	2.585	8.941	38.8062
		C.I.	1.849	6.396	27.7602
		P Value	1E-08	2E-04	0.35084
	2	mean	13.26	15.43	-23.675
		STD	1.941	11.34	54.4423
		C.I.	1.388	8.112	38.9457
		P Value	5E-09	0.002	0.20234
Matching	1	mean	16.24	18.44	-9.9
		STD	2.508	10.1	31.4968
		C.I.	1.794	7.227	22.5314
		P Value	7E-09	3E-04	0.34621
	2	mean	13.74	16.34	-21.5
		STD	2.313	9.964	49.0692
		C.I.	1.655	7.128	35.102
		P Value	2E-08	6E-04	0.19925
Forward	1	mean	17.2	28	-20.1
		STD	2.7	8.192	89.9894
		C.I.	1.931	5.86	64.3746
		P Value	9E-09	2E-06	0.49786
	2	mean	14.7	25.9	-31.7
		STD	2.71	11.18	85.7776
		C.I.	1.939	7.998	61.3616
		P Value	4E-08	4E-05	0.27257
Backward	1	mean	14.1	27.9	-21.9
		STD	9.86	8.504	92.6924
		C.I.	7.053	6.084	66.3081
		P Value	0.001	3E-06	0.47405
	2	mean	11.6	25.8	-33.5
		STD	8.262	11.25	78.3429
		C.I.	5.911	8.05	56.0431
		P Value	0.002	5E-05	0.2093

TABLE 5.16: Comparison of Hybrid Genetic Algorithm using moderate local improver with existing heuristics for Polio Virus (Also see Figure 5.14)

Heuristic	Selection		Δf_1	Δf_2	Δf_3
Greedy	mlts	mean	7.155	1.225	17.13
		STD	7.181	14.61	48.74
		C.I.	5.137	10.45	34.86
		P Value	0.012	0.679	0.295
	mrts	mean	-2.95	-18.2	48.73
		STD	4.728	13.6	82.17
		C.I.	3.383	9.727	58.78
		P Value	0.08	0.002	0.094
	mmts	mean	2.035	-9.07	63.1
		STD	5.923	12.98	57.08
		C.I.	4.237	9.287	40.83
		P Value	0.306	0.055	0.007
Matching	mlts	mean	7.635	2.135	19.3
		STD	7.68	15.5	58.98
		C.I.	5.494	11.09	42.19
		P Value	0.012	0.615	0.328
	mrts	mean	-2.47	-17.3	50.9
		STD	4.331	9.582	67.53
		C.I.	3.098	6.855	48.31
		P Value	0.105	3E-04	0.041
	mmts	mean	2.035	-9.07	63.1
		STD	5.923	12.98	57.08
		C.I.	4.237	9.287	40.83
		P Value	0.306	0.055	0.007
Forward	mlts	mean	8.6	11.7	9.1
		STD	6.022	12.06	113.2
		C.I.	4.308	8.624	81
		P Value	0.001	0.027	0.805
	mrts	mean	-1.5	-7.7	40.7
		STD	4.743	10.88	106.2
		C.I.	3.393	7.786	75.99
		P Value	0.343	0.052	0.257
	mmts	mean	3	0.5	52.9
		STD	5.944	13.53	114.9
		C.I.	4.252	9.682	82.22
		P Value	0.145	0.91	0.18
Backward	mlts	mean	5.5	11.6	7.3
		STD	10.1	12.56	122
		C.I.	7.227	8.987	87.3
		P Value	0.119	0.03	0.854
	mrts	mean	-4.6	-7.8	38.9
		STD	10.6	10.91	105.5
		C.I.	7.58	7.806	75.47
		P Value	0.203	0.05	0.274
	mmts	mean	-0.1	0.4	51.1
		STD	6.607	13.31	114.9
		C.I.	4.727	9.521	82.17
		P Value	0.963	0.926	0.193

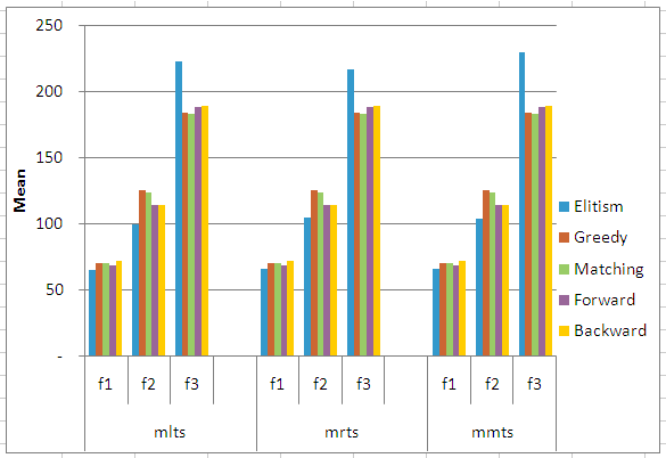

FIGURE 5.15: Comparison of Hybrid GA with Elitism with existing heuristics for Polio Virus (Also see Table 5.17)

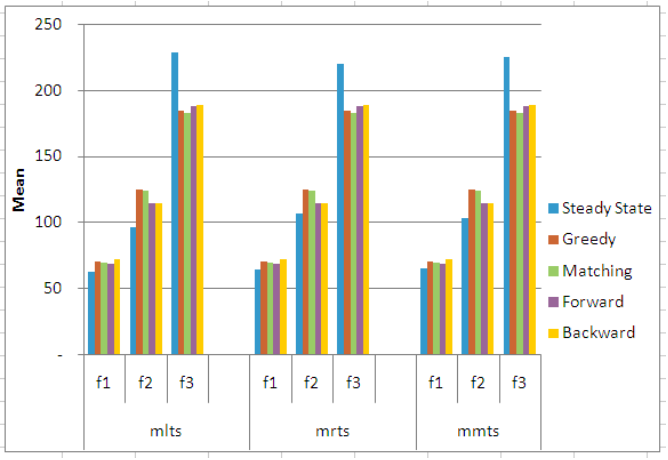

FIGURE 5.16: Comparison of Hybrid Steady State GA Algorithm with existing heuristics for Polio Virus (Also see Table 5.18)

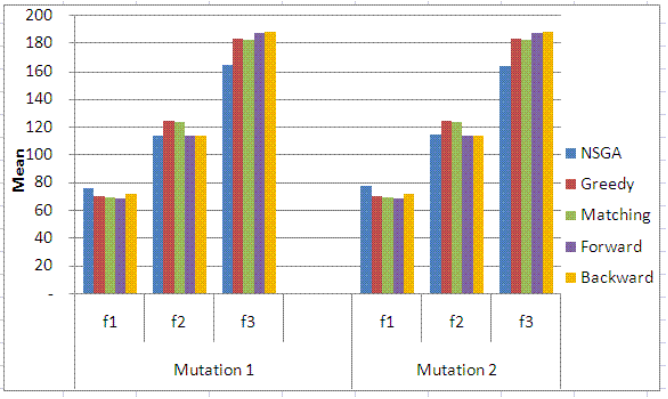

FIGURE 5.17: Comparison of Non-dominated Sorting Genetic Algorithm with existing heuristics for Polio virus (Also see Table 5.19)

TABLE 5.17: Comparison of Hybrid GA with Elitism with existing heuristics for Polio Virus (Also see Figure 5.15)

Heuristic	Selection		Δf_1	Δf_2	Δf_3
Greedy	mlts	mean	-5.25	-25	77.425
		STD	2.74	16.01	59.4845
		C.I.	1.96	11.46	42.5526
		P Value	2E-04	0.002	0.00261
	mrts	mean	-4.75	-20.7	65.225
		STD	4.849	13.02	51.3057
		C.I.	3.469	9.316	36.7019
		P Value	0.013	7E-04	0.00302
	mmts	mean	-4.15	-21.5	90.325
		STD	3.876	12.91	101.775
		C.I.	2.773	9.233	72.8052
		P Value	0.008	5E-04	0.02049
Matching	mlts	mean	-4.77	-24.1	79.6
		STD	3.511	12.56	77.9916
		C.I.	2.511	8.982	55.7918
		P Value	0.002	7E-04	0.01036
	mrts	mean	-4.27	-19.8	67.4
		STD	4.543	10.2	70.2483
		C.I.	3.25	7.296	50.2526
		P Value	0.016	2E-04	0.01415
	mmts	mean	-3.67	-20.6	92.5
		STD	3.439	11.99	110.111
		C.I.	2.46	8.58	78.7684
		P Value	0.008	4E-04	0.02619
Forward	mlts	mean	-3.8	-14.5	69.4
		STD	3.458	13.99	115.265
		C.I.	2.473	10.01	82.4557
		P Value	0.007	0.015	0.08932
	mrts	mean	-3.3	-10.2	57.2
		STD	4.945	10.27	104.635
		C.I.	3.538	7.348	74.8511
		P Value	0.064	0.012	0.11792
	mmts	mean	-2.7	-11	82.3
		STD	4.923	14.59	144.487
		C.I.	3.522	10.44	103.36
		P Value	0.117	0.041	0.10518
Backward	mlts	mean	-6.9	-14.6	67.6
		STD	9.073	14.01	120.146
		C.I.	6.491	10.02	85.9476
		P Value	0.04	0.016	0.10891
	mrts	mean	-6.4	-10.3	55.4
		STD	10.33	10.45	105.934
		C.I.	7.39	7.473	75.7807
		P Value	0.082	0.012	0.13256
	mmts	mean	-5.8	-11.1	80.5
		STD	11.42	14.18	152.727
		C.I.	8.169	10.14	109.254
		P Value	0.143	0.035	0.12991

TABLE 5.18: Comparison of Hybrid Steady State GA Algorithm with existing heuristics for Polio Virus (Also see Figure 5.16)

Heuristic	Selection		Δf_1	Δf_2	Δf_3
Greedy	mlts	mean	-7.55	-29.8	89.23
		STD	4.101	11.37	54.32
		C.I.	2.934	8.137	38.86
		P Value	3E-04	3E-05	6E-04
	mrts	mean	-5.55	-18.3	71.73
		STD	2.883	12.41	80.96
		C.I.	2.062	8.876	57.91
		P Value	2E-04	0.001	0.021
	mmts	mean	-5.15	-21.9	82.43
		STD	4.822	12.46	68.33
		C.I.	3.449	8.915	48.88
		P Value	0.008	4E-04	0.004
Matching	mlts	mean	-7.07	-28.9	91.4
		STD	4.418	10.53	61.62
		C.I.	3.16	7.534	44.08
		P Value	7E-04	2E-05	0.001
	mrts	mean	-5.07	-17.4	73.9
		STD	3.672	10.7	66.69
		C.I.	2.627	7.651	47.71
		P Value	0.002	6E-04	0.007
	mmts	mean	-4.67	-21	84.6
		STD	5.154	7.218	48.95
		C.I.	3.687	5.164	35.01
		P Value	0.019	7E-06	4E-04
Forward	mlts	mean	-6.1	-19.3	81.2
		STD	5.507	11.2	96.2
		C.I.	3.939	8.009	68.82
		P Value	0.007	0.001	0.026
	mrts	mean	-4.1	-7.8	63.7
		STD	3.573	11.91	118
		C.I.	2.556	8.523	84.44
		P Value	0.005	0.068	0.122
	mmts	mean	-3.7	-11.4	74.4
		STD	6.165	9.119	91.96
		C.I.	4.41	6.523	65.79
		P Value	0.09	0.003	0.031
Backward	mlts	mean	-9.2	-19.4	79.4
		STD	8.404	11.12	103.9
		C.I.	6.012	7.953	74.31
		P Value	0.007	0.001	0.039
	mrts	mean	-7.2	-7.9	61.9
		STD	9.114	10.8	108.6
		C.I.	6.52	7.723	77.69
		P Value	0.034	0.046	0.105
	mmts	mean	-6.8	-11.5	72.6
		STD	9.693	9.324	93.89
		C.I.	6.934	6.67	67.16
		P Value	0.054	0.004	0.037

TABLE 5.19: Comparison of Non-dominated Sorting Genetic Algorithm with existing heuristics for Polio virus (Also see Figure 5.17)

Heuristic	Mutation Type		Δf_1	Δf_2	Δf_3
Greedy	1	mean	5.855	-10.2	-39.2
		STD	2.606	12.61	31.41
		C.I.	1.865	9.017	22.47
		P Value	6E-05	0.031	0.003
	2	mean	7.455	-9.58	-39.6
		STD	3.672	12.47	43.52
		C.I.	2.627	8.918	31.57
		P Value	1E-04	0.038	0.018
Matching	1	mean	6.335	-9.27	-37
		STD	3.589	8.162	32.98
		C.I.	2.568	5.839	23.59
		P Value	3E-04	0.006	0.006
	2	mean	7.935	-8.67	-37.4
		STD	3.898	12.63	46.83
		C.I.	2.788	9.035	33.09
		P Value	1E-04	0.058	0.032
Forward	1	mean	7.3	0.3	-47.2
		STD	3.129	9.19	84
		C.I.	2.238	6.574	60.09
		P Value	4E-05	0.92	0.109
	2	mean	8.9	0.9	-47.6
		STD	3.107	12.59	76.08
		C.I.	2.223	9.007	51.8
		P Value	8E-06	0.826	0.079
Backward	1	mean	4.2	0.2	-49
		STD	8.867	8.967	89.14
		C.I.	6.343	6.414	63.77
		P Value	0.168	0.945	0.116
	2	mean	5.8	0.8	-49.4
		STD	7.33	12.2	76.85
		C.I.	5.244	8.727	51.84
		P Value	0.034	0.84	0.073

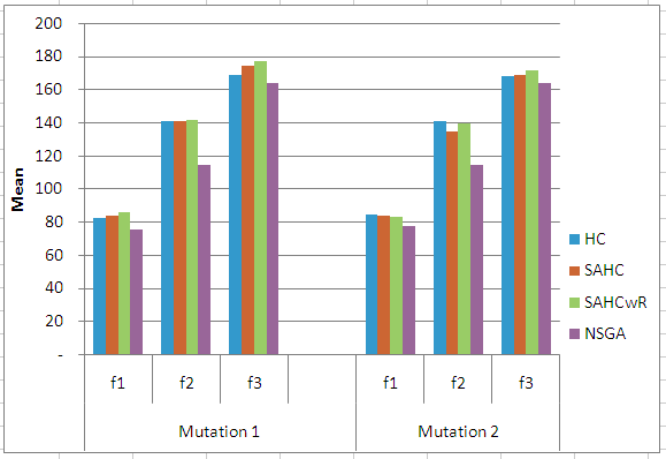

FIGURE 5.18: Comparison of Non-dominated Sorting Genetic Algorithm with local search techniques for Polio virus (Also see Table 5.20)

TABLE 5.20: Comparison of Non-dominated Sorting Genetic Algorithm with local search techniques for Polio virus (Also see Figure 5.18)

Heuristic	Mutation Type		Δf_1	Δf_2	Δf_3
HC	1	mean	-9	-26.6	-9.1
		STD	3.528	9.857	44.9
		C.I.	2.524	8.882	32.12
		P Value	2E-05	1E-05	0.538
	2	mean	-4.9	-26.3	-10.1
		STD	5.216	13.33	48.42
		C.I.	3.732	9.538	34.52
		P Value	0.016	2E-04	0.526
SAHC	1	mean	-7.8	-20.7	-9.4
		STD	4.341	24.15	35.56
		C.I.	3.105	17.27	25.44
		P Value	3E-04	0.024	0.425
	2	mean	-6.3	-26.5	-21.5
		STD	3.86	10.29	52.45
		C.I.	2.761	7.359	36.61
		P Value	6E-04	2E-05	0.227
SAHCwR	1	mean	-9.9	-27.7	-27.1
		STD	4.04	14.61	37.82
		C.I.	2.89	10.45	27.05
		P Value	3E-05	2E-04	0.05
	2	mean	-5.8	-25	-15.9
		STD	3.553	14.83	37.21
		C.I.	2.542	10.61	27.56
		P Value	6E-04	5E-04	0.21

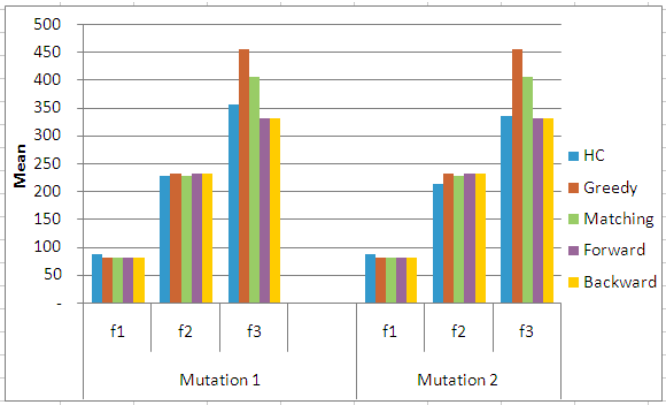

FIGURE 5.19: Comparison of Hill Climbing (HC) Algorithm with existing heuristics for Equine Arteritis Virus (Also see Table 5.21)

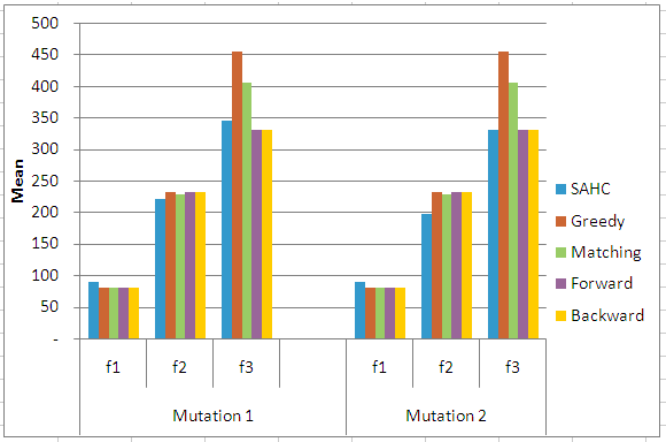

FIGURE 5.20: Comparison of Steepest Ascent Hill Climbing (SAHC) Algorithm with existing heuristics for Equine Arteritis Virus (Also see Table 5.22)

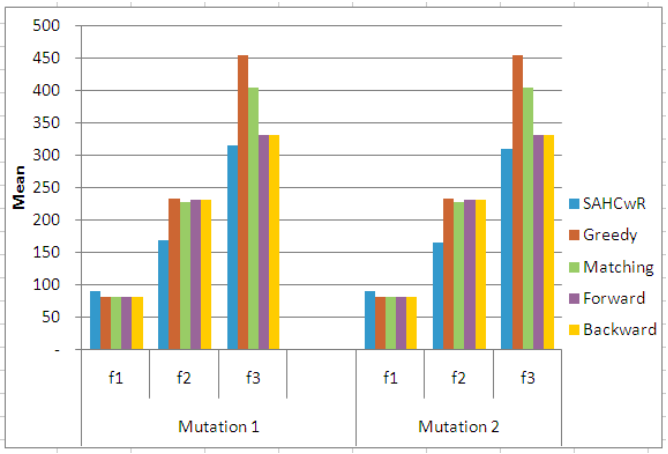

FIGURE 5.21: Comparison of Steepest Ascent Hill Climbing with Replacement (SAHCwR) Algorithm with existing heuristics for Equine Arteritis Virus (Also see Table 5.23)

TABLE 5.21: Comparison of Hill Climbing (HC) Algorithm with existing heuristics for Equine Arteritis Virus (Also see Figure 5.19)

Heuristic	Mutation type		Δf_1	Δf_2	Δf_3
Greedy	1	mean	6.1	-5.3	-199.5
		STD	2.558	98.93	246.856
		C.I.	2.505	70.77	176.59
		P Value	4E-05	0.869	0.03091
	2	mean	5.2	-18.5	-239
		STD	3.425	95.69	196.101
		C.I.	2.45	68.45	140.282
		P Value	1E-03	0.556	0.00388
Matching	1	mean	6.9	-0.6	-99.6
		STD	3.604	95.5	162.047
		C.I.	2.578	68.32	115.921
		P Value	2E-04	0.985	0.08382
	2	mean	6	-13.8	-139.1
		STD	3.916	92.22	123.292
		C.I.	2.801	65.97	88.198
		P Value	9E-04	0.647	0.00605
Forward	1	mean	6.8	-4.4	48.1
		STD	3.706	96.44	181.662
		C.I.	2.651	68.99	129.953
		P Value	3E-04	0.888	0.4241
	2	mean	5.9	-17.6	8.6
		STD	3.315	92.39	111.733
		C.I.	2.371	66.09	79.929
		P Value	3E-04	0.562	0.81315
Backward	1	mean	6.8	-4.4	48.1
		STD	3.706	96.44	181.662
		C.I.	2.651	68.99	129.953
		P Value	3E-04	0.888	0.4241
	2	mean	5.9	-17.6	8.6
		STD	3.315	92.39	111.733
		C.I.	2.371	66.09	79.929
		P Value	3E-04	0.562	0.81315

TABLE 5.22: Comparison of Steepest Ascent Hill Climbing (SAHC) Algorithm with existing heuristics for Equine Arteritis Virus (Also see Figure 5.20)

Heuristic	Mutation type		Δf_1	Δf_2	Δf_3
Greedy	1	mean	8.7	-10.7	-218
		STD	4.111	110.1	219.2
		C.I.	2.941	78.75	156.8
		P Value	9E-05	0.766	0.012
	2	mean	7.8	-35.5	-249
		STD	3.553	92.06	206.1
		C.I.	2.542	65.86	147.4
		P Value	7E-05	0.254	0.004
Matching	1	mean	9.5	-6	-118
		STD	5.563	106.2	82.49
		C.I.	3.979	75.97	59.01
		P Value	4E-04	0.862	0.001
	2	mean	8.6	-30.8	-149
		STD	4.351	88.29	85.04
		C.I.	3.113	63.16	60.83
		P Value	1E-04	0.299	4E-04
Forward	1	mean	9.4	-9.8	30
		STD	5.337	106.5	127.5
		C.I.	3.818	76.15	91.2
		P Value	3E-04	0.778	0.476
	2	mean	8.5	-34.6	-1.6
		STD	3.808	88.69	152.3
		C.I.	2.724	63.44	108.9
		P Value	6E-05	0.249	0.974
Backward	1	mean	9.4	-9.8	30
		STD	5.337	106.5	127.5
		C.I.	3.818	76.15	91.2
		P Value	3E-04	0.778	0.476
	2	mean	8.5	-34.6	-1.6
		STD	3.808	88.69	152.3
		C.I.	2.724	63.44	108.9
		P Value	6E-05	0.249	0.974

TABLE 5.23: Comparison of Steepest Ascent Hill Climbing with Replacement (SAHCwR) Algorithm with existing heuristics for Equine Arteritis Virus (Also see Figure 5.21)

Heuristic	Mutation type		Δf_1	Δf_2	Δf_3
Greedy	1	mean	7.8	-63.4	-280.5
		STD	3.048	109.7	256.771
		C.I.	2.18	78.47	183.683
		P Value	2E-05	0.101	0.00722
	2	mean	7.5	-67.1	-290.3
		STD	3.375	59.05	213.729
		C.I.	2.414	42.24	152.892
		P Value	6E-05	0.006	0.002
Matching	1	mean	8.6	-58.7	-180.6
		STD	4.377	108	78.3584
		C.I.	3.131	77.28	56.0542
		P Value	2E-04	0.12	4.6E-05
	2	mean	8.3	-62.4	-190.4
		STD	3.683	55.52	100.847
		C.I.	2.635	39.72	72.1413
		P Value	6E-05	0.006	0.00021
Forward	1	mean	8.5	-62.5	-32.9
		STD	3.923	108.6	133.469
		C.I.	2.806	77.7	95.4783
		P Value	7E-05	0.102	0.4557
	2	mean	8.2	-66.2	-42.7
		STD	3.49	55.42	103.82
		C.I.	2.496	39.64	74.2686
		P Value	4E-05	0.004	0.22571
Backward	1	mean	8.5	-62.5	-32.9
		STD	3.923	108.6	133.469
		C.I.	2.806	77.7	95.4783
		P Value	7E-05	0.102	0.4557
	2	mean	8.2	-66.2	-42.7
		STD	3.49	55.42	103.82
		C.I.	2.496	39.64	74.2686
		P Value	4E-05	0.004	0.22571

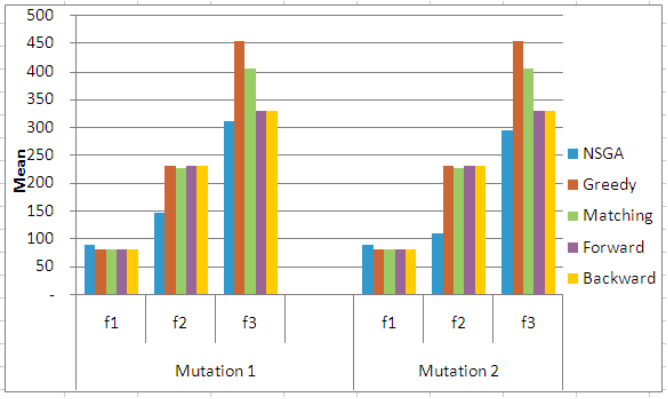

FIGURE 5.22: Comparison of Non-dominated Sorting Genetic Algorithm with existing heuristics for Equine Arteritis virus (Also see Table 5.24)

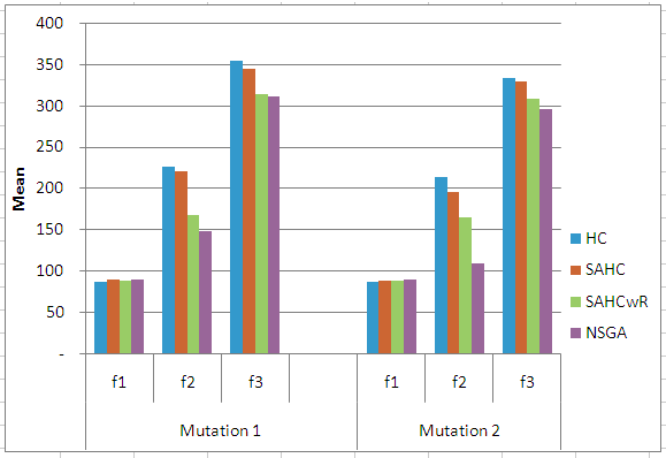

FIGURE 5.23: Comparison of Non-dominated Sorting Genetic Algorithm with local search techniques for Equine Arteritis virus (Also see Table 5.25)

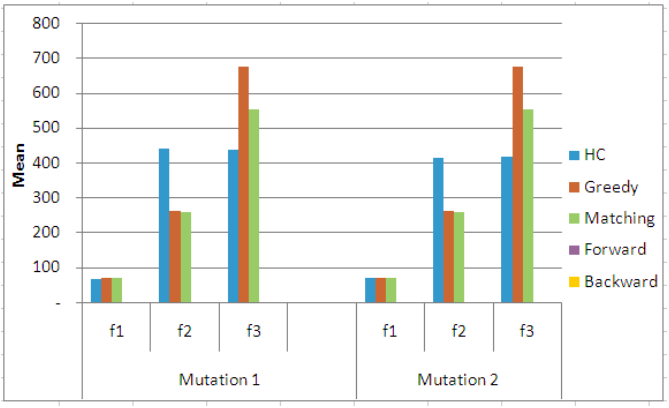

FIGURE 5.24: Comparison of Hill Climbing (HC) Algorithm with existing heuristics for Measles Virus (Also see Table 5.26)

TABLE 5.24: Comparison of Non-dominated Sorting Genetic Algorithm with existing heuristics for Equine Arteritis virus (Also see Figure 5.22)

Heuristic	Mutation type		Δf_1	Δf_2	Δf_3
Greedy	1	mean	8.6	-84.3	-285
		STD	3.3731	53.97	213.9
		C.I.	2.41297	38.61	153
		P Value	2.1E-05	8E-04	0.002
	2	mean	8.9	-123	-317
		STD	3.03498	13.62	201.7
		C.I.	2.17109	9.746	144.3
		P Value	6.7E-06	4E-10	8E-04
Matching	1	mean	9.4	-79.6	-186
		STD	3.09839	51.11	163.4
		C.I.	2.21645	36.56	116.9
		P Value	5E-06	8E-04	0.006
	2	mean	9.7	-118	-217
		STD	3.40098	16.41	91.75
		C.I.	2.43291	11.74	65.64
		P Value	8.4E-06	3E-09	4E-05
Forward	1	mean	9.3	-83.4	-37.8
		STD	2.94581	51.95	155.4
		C.I.	2.1073	37.17	111.2
		P Value	3.6E-06	7E-04	0.461
	2	mean	9.6	-122	-68.9
		STD	3.94968	17.68	134.1
		C.I.	2.82543	12.64	95.93
		P Value	3E-05	4E-09	0.139
Backward	1	mean	9.3	-83.4	-37.8
		STD	2.94581	51.95	155.4
		C.I.	2.1073	37.17	111.2
		P Value	3.6E-06	7E-04	0.461
	2	mean	9.6	-122	-68.9
		STD	3.94968	17.68	134.1
		C.I.	2.82543	12.64	95.93
		P Value	3E-05	4E-09	0.139

TABLE 5.25: Comparison of Non-dominated Sorting Genetic Algorithm with local search techniques for Equine Arteritis virus (Also see Figure 5.23)

Heuristic	Mutation type		Δf_1	Δf_2	Δf_3
HC	1	mean	2.5	-79	-85.9
		STD	3.808	70.67	116
		C.I.	2.724	50.56	82.96
		P Value	0.068	0.006	0.044
	2	mean	3.7	-104	-77.5
		STD	4.057	102.2	96.04
		C.I.	2.902	73.09	68.71
		P Value	0.018	0.01	0.031
SAHC	1	mean	-0.1	-73.6	-67.8
		STD	5.99	66.86	104.6
		C.I.	4.285	47.83	74.81
		P Value	0.959	0.007	0.071
	2	mean	1.1	-87.1	-67.3
		STD	4.677	96.88	54.72
		C.I.	3.346	69.31	39.15
		P Value	0.476	0.019	0.004
SAHCwR	1	mean	0.8	-20.9	-4.9
		STD	5.16	74.21	131.7
		C.I.	3.691	53.09	94.24
		P Value	0.636	0.396	0.909
	2	mean	1.4	-55.5	-26.2
		STD	2.797	66	55.62
		C.I.	2.001	47.21	39.79
		P Value	0.148	0.026	0.171

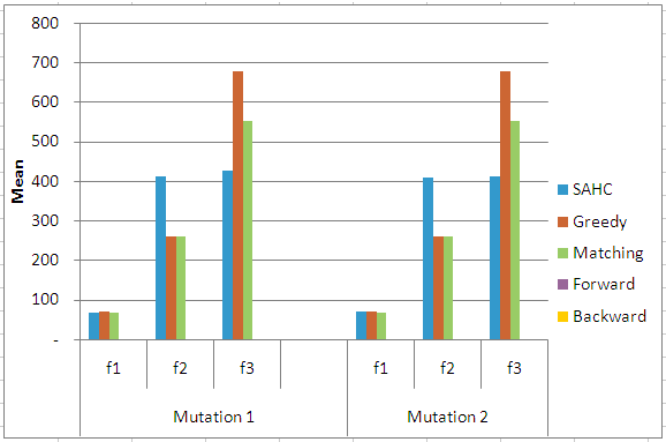

FIGURE 5.25: Comparison of Steepest Ascent Hill Climbing (SAHC) Algorithm with existing heuristics for Measles Virus (Also see Table 5.27)

TABLE 5.26: Comparison of Hill Climbing (HC) Algorithm with existing heuristics for Measles Virus (Also see Figure 5.24)

Heuristic	Mutation type		Δf_1	Δf_2	Δf_3
Greedy	1	mean	-1.8	178.2	-478
		STD	4.442	34.87	315.406
		C.I.	3.178	24.95	225.628
		P Value	0.232	6E-08	0.00098
	2	mean	-0.1	151.8	-520.7
		STD	4.771	69.67	218.773
		C.I.	3.413	49.84	156.501
		P Value	0.949	7E-05	3.6E-05
Matching	1	mean	-0.3	179.7	-228.4
		STD	5.187	39.99	255.76
		C.I.	3.71	28.61	182.96
		P Value	0.859	2E-07	0.01992
	2	mean	1.4	153.3	-271.1
		STD	4.835	73.59	145.717
		C.I.	3.459	52.64	104.24
		P Value	0.384	1E-04	0.00023

TABLE 5.27: Comparison of Steepest Ascent Hill Climbing (SAHC) Algorithm with existing heuristics for Measles Virus (Also see Figure 5.25)

Heuristic	Mutation type		Δf_1	Δf_2	Δf_3
Greedy	1	mean	-1.6	152	-498
		STD	5.038	69.59	198.5
		C.I.	3.604	49.78	142
		P Value	0.341	7E-05	2E-05
	2	mean	-0.7	149.4	-526
		STD	5.697	66.89	213.8
		C.I.	4.075	47.85	152.9
		P Value	0.707	6E-05	3E-05
Matching	1	mean	-0.1	153.5	-248
		STD	5.322	72.34	203.8
		C.I.	3.807	51.75	145.8
		P Value	0.954	9E-05	0.004
	2	mean	0.8	150.9	-277
		STD	5.514	74.35	243.5
		C.I.	3.944	53.19	174.2
		P Value	0.657	1E-04	0.006

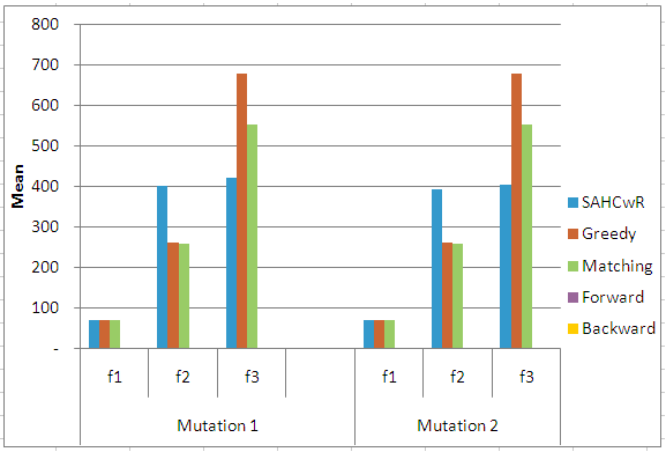

FIGURE 5.26: Comparison of Steepest Ascent Hill Climbing with Replacement (SAHCwR) Algorithm with existing heuristics for Measles Virus (Also see Table 5.28)

TABLE 5.28: Comparison of Steepest Ascent Hill Climbing with Replacement (SAHCwR) Algorithm with existing heuristics for Measles Virus (Also see Figure 5.26)

Heuristic	Mutation type		Δf_1	Δf_2	Δf_3
Greedy	1	mean	-1.3	141.3	-513.2
		STD	3.02	62	286.591
		C.I.	2.161	44.35	205.015
		P Value	0.207	5E-05	0.00031
	2	mean	-1	132.9	-543.6
		STD	3.972	39.26	264.11
		C.I.	2.841	28.09	188.933
		P Value	0.446	2E-06	0.00011
Matching	1	mean	0.2	142.8	-263.6
		STD	2.616	66.07	216.977
		C.I.	1.872	47.26	155.216
		P Value	0.814	8E-05	0.00396
	2	mean	0.5	134.4	-294
		STD	4.601	40.65	261.546
		C.I.	3.291	29.08	187.099
		P Value	0.739	2E-06	0.00617

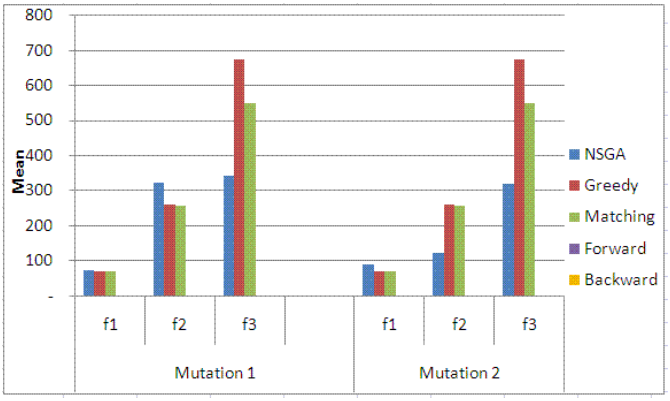

FIGURE 5.27: Comparison of Non-dominated Sorting Genetic Algorithm with existing heuristics for Measles virus (Also see Table 5.29)

TABLE 5.29: Comparison of Non-dominated Sorting Genetic Algorithm with existing heuristics for Measles virus (Also see Figure 5.27)

Heuristic	Mutation type		Δf_1	Δf_2	Δf_3
Greedy	1	mean	3.4	61.6	-665
		STD	8.98394	142.8	165.9
		C.I.	6.42672	102.2	118.7
		P Value	0.26197	0.206	5E-07
	2	mean	18.6	-139	-714
		STD	4.74225	26.22	250.1
		C.I.	3.3924	18.76	178.9
		P Value	5.8E-07	4E-08	8E-06
Matching	1	mean	4.9	63.1	-416
		STD	10.0383	147.3	170.1
		C.I.	7.18094	105.4	121.7
		P Value	0.15708	0.209	3E-05
	2	mean	20.1	-138	-465
		STD	3.92853	33.64	146.1
		C.I.	2.8103	24.06	104.5
		P Value	5.8E-08	4E-07	3E-06

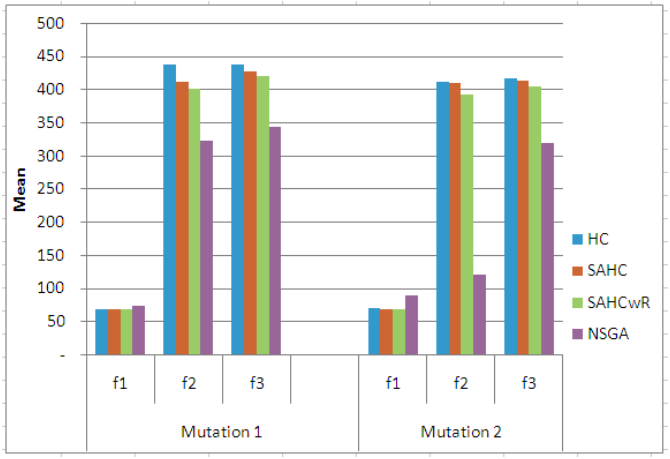

FIGURE 5.28: Comparison of Non-dominated Sorting Genetic Algorithm with local search techniques for Measles virus (Also see Table 5.30)

TABLE 5.30: Comparison of Non-dominated Sorting Genetic Algorithm with local search techniques for Measles virus (Also see Figure 5.28)

Heuristic	Mutation type		Δf_1	Δf_2	Δf_3
HC	1	mean	5.2	-117	-187
		STD	9.727	150.7	252.9
		C.I.	6.959	107.8	180.9
		P Value	0.125	0.037	0.044
	2	mean	18.7	-291	-193
		STD	5.165	66.95	207.8
		C.I.	3.695	47.89	148.7
		P Value	1E-06	2E-07	0.016
SAHC	1	mean	5	-90.4	-168
		STD	9.707	158.3	163.7
		C.I.	6.944	113.3	117.1
		P Value	0.138	0.104	0.01
	2	mean	19.3	-289	-188
		STD	5.314	56.17	229.2
		C.I.	3.801	40.18	164
		P Value	1E-06	6E-08	0.029
SAHCwR	1	mean	4.7	-79.7	-152
		STD	8.769	173.3	206.9
		C.I.	6.273	124	148
		P Value	0.124	0.18	0.045
	2	mean	19.6	-272	-171
		STD	6.637	53.6	229.4
		C.I.	4.748	38.35	164.1
		P Value	6E-06	6E-08	0.043

5.3 Summary

This chapter mainly deals with our experimental techniques and the output of simulation results. These are not only shown in tabular format but also a discussion have been done revealing the reasons behind their behavior.

Chapter 6

Conclusion

This thesis deals with a topic which falls under the subfield of bioinformatics, namely synthetic biology. The main idea of this study is to find a genome sequence which aids the biologists to obtain maximum amount of flexibility and independence to conduct experiments with the sequence in question. Another aim is to give opportunity for investigation of vaccine invention. To this end, for making a genome sequence uniquely prone to enzymes, we proposed to apply metaheuristics process. We conclude the thesis presenting our findings from our research and future directions.

6.1 Findings

This is the first attempt taken for applying high level search techniques to solve optimal placement of unique restriction sites in synthetic genomes. Our aim is to provide a better genome sequence that the existing tool, $PRESTO$ can provide. From our study we have found that local search techniques performs better in terms of enzyme insertion and maintaining uniform gap between consecutive unique sites. In case of local search, as we increase the extent of exploration the improvement increases, i.e., Steepest Ascent Hill climbing with Replacement is better than Steepest Ascent Hill Climbing which performs better than basic Hill Climbing. In each case, we see that our 2nd breeding operator (Mutation 2) leads to reach at a more qualitative candidate solution. Here, the reason is again the 'injection of higher exploration'.

In case of variations of genetic algorithms, the results are not satisfactory even though they explores the search space in higher order. Since *URSPP* is a discrete optimization problem and discrete optimization does not require too much exploration genetic algorithms can not provide expected results even with the 'injection of exploitation'. Here we have experimented with real viral sequence and to the degeneracy property of codon make a genome sequence more fault-tolerant for point mutations. Hence, the use of crossover instead of mutation as breeding operator here might affect the results. Finally, we see that Non-Dominating Sorting Genetic Algorithm offers best results among all algorithms.

6.2 Future Direction

As discussed in previous section, impact of multi-objective tournament selections can not be perceived properly due to the crossover effect. The *Evolutionary Strategies ($\lambda + \mu$) algorithms*[1] use only mutations as their breeding operator. So, this family of algorithms can be applied here to investigate which tournament selection among *Multiobjective Lexicographic Tournament Selection (mlts)*, *Multiobjective Majority Tournament Selection (mmts)* and *Multiobjective Ratio Tournament Selection (mrts)* works better. Additionally, we have seen NSGA returns a comparatively larger front. Defining an appropriate notion to select a good candidate solution among all candidates of same rank might be a possible future work. Another future research issue can be the analysis of the influence of using *Strength Pareto Evolutionary Algorithms*[2] in *URSPP*.

Over the last few decades synthetic biology and computer science have made great strides. Biologists are now in a position of being faced with biological information of such volume that it is impossible to analyze the information by pen and paper. The next step must be to create computational tools to use this information effectively. Hopefully this thesis will serve as a step in that direction. We believe that collaboration of biologists and computer scientists will do wonders for mankind in this century.

[1] Here, in each iteration, λ number of individuals comprise the population and only μ fittest individuals participate to produce next generation through mutation.

[2] Strength of an individual is how many other individuals it can pareto dominate. and SPEA algorithms use notions related to strength to find a better quality candidate.

Bibliography

[1] T. S. Gardner, C. R. Cantor, and J. J. Collins. Construction of a genetic toggle switch in escherichia coli. *Nature*, 403(6767):339–342, 2000.

[2] Y. Cai Y, M. W. Lux, L. Adam, and J. Peccoud. Modeling structure-function relationships in synthetic dna sequences using attribute grammars. *PLoS Comput Biol*, 5(10), 2009.

[3] P. Montes, H. Memelli, C. B. Ward, J. Kim, J. S. B. Mitchell, and S. Skiena. Optimizing restriction site placement for synthetic genomes. In *CPM*, pages 323–337, 2010.

[4] S. A. Benner and A. M. Sismour. Synthetic biology. *Nat Rev Genet.*, 6(7):533–543, 2005.

[5] D. Endy. Foundations for engineering biology. *Nature*, 438(7067):449–453, 2005.

[6] E. Andrianantoandro, S. Basu, D. K. Karig, and R. Weiss. Synthetic biology: new engineering rules for an emerging discipline. *Mol Syst Biol*, pages 2–28, 2006.

[7] M. Heinemann and S. Panke. Synthetic biology–putting engineering into biology. *Bioinformatics*, 22(22):2790–2799, 2006.

[8] `http://ec.europa.eu/research/fp6/index_en.cfm?p=8_nest3`.

[9] `http://commcgi.cc.stonybrook.edu/am2/publish/Medical_Center_Health_Care_4/SBU_Team_Designs_Customized_Wimpy_Polioviruses_A_Method_That_Could_Be_A_New_Path_To_Vaccines.shtml`.

[10] T. Mens and T. Tourwé. A survey of software refactoring. *IEEE Trans. Software Eng.*, 30(2):126–139, 2004.

[11] N. Kuldell and N. Lerner. Genome refactoring. *Morgan and Claypool Publishers.*, ISBN-10: 1598299476:126–139, 2009.

[12] L. Chan, S. Kosuri, and D. Endy. Refactoring bacteriophage t7. *Mol. Syst. Biol.*, 1, 2005.

[13] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & OR*, 13(5):533–549, 1986.

[14] C. R. Reeves. *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell Scientific Publishing, Oxford, England, 1993.

[15] I. H. Osman and G. Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63:513–623, 1996.

[16] S. Vo, S. Martello, I. H. Osman, and C. Roucairol. *Meta-Heuristics - Advances and Trends in Local Search Paradigms for Optimization.* Kluwer Academic Publishers, Dordrecht, The Nether-lands, 1999.

[17] T. Stützle. *Local search algorithms for combinatorial problems - analysis, improvements, and new applications*, volume 220 of *DISKI*. Infix, 1999. ISBN 978-3-89601-220-3.

[18] F. Glover and M. Laguna. *TABU search*. Kluwer, 1999. ISBN 978-0-7923-9965-0.

[19] A. E. Eiben and C. A. Schippers. On evolutionary exploration and exploitation. *Fundam. Inform.*, 35(1-4):35–50, 1998.

[20] C. Blum, M. J. B. Aguilera, A. Roli, and M. Sampels, editors. *Hybrid Metaheuristics, An Emerging Approach to Optimization*, volume 114 of *Studies in Computational Intelligence*. Springer, 2008. ISBN 978-3-540-78294-0.

[21] C. A. C. Coello, D. A. V. Veldhuizen, and G. B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, 2002. ISBN ISBN-10: 0306467623.

[22] D. A. V. Veldhuizen and B. G. Lamont. Multiobjective evolutionary algorithms: Analyzing the state-of-the-art. *Evolutionary Computation*, 8(2):125–147, 2000.

[23] S. Luke. Essentials of metaheuristics. *Genetic Programming and Evolvable Machines*, 12(3):333–334, 2011.

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

[25] K. D. Jong. *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, 1975.

[26] D. Whitley and J. Kauth. *GENITOR: A Different Genetic Algorithm*. Colorado State University Press, 1988.

[27] T. Bäck, U. Hammel, and H. P. Schwefel. Evolutionary computation: comments on the history and current state. *IEEE Trans. Evolutionary Computation*, 1(1): 3–17, 1997.

[28] E. Zitzler, K. Deb, L. Thiele, C. A. C. Coello, and D. Corne, editors. *Evolutionary Multi-Criterion Optimization, First International Conference, EMO 2001, Zurich, Switzerland, March 7-9, 2001, Proceedings*, volume 1993 of *Lecture Notes in Computer Science*, 2001. Springer. ISBN 3-540-41745-1.

[29] K. Deb. *Multi-objective optimization using evolutionary algorithms.* Wiley, Chichester, UK, 2001.

[30] A. Brindle. *Genetic Algorithms for Function Optimization*. PhD thesis, University of Alberta, 1981.

[31] N. Srinivas and K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

[32] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[33] `http://www.blueheronbio.com/`.

[34] `http://www.origene.com/about/corporate/`.

[35] `http://www.geneart.com`.

[36] P. Evans and C. Liu. Sitefind: A software tool for introducing a restriction site as a marker for successful site-directed mutagenesis. *BMC Mol. Biol.*, 6(22), 2005.

[37] I. Anand, S. Kosuri, and D. Endy. Genejax: A prototype cad tool in support of genome refactoring. *Morgan and Claypool Publishers.*, 2006.

[38] S. Skiena. Designing better phages. *Bioinformatics*, 17:S253–S261, 2001.

[39] A. V. Aho and M. J. Corasick. Efficient string matching: An aid to bibliographic search. *Commun. ACM*, 18(6):333–340, 1975.

[40] `http://en.wikipedia.org/wiki/Genetic_code`.

[41] S. Luke and L. Panait. Lexicographic parsimony pressure. In *Proc. of GECCO*, pages 148–157, 2002.

[42] `http://www.ncbi.nlm.nih.gov/`. [Online; accessed 16-January-2012].

[43] R. Roberts, T. Vincze, J. Posfai, and D. Macelis. Rebase-a database for dna restriction and modification: enzymes, genes and genomes. *Nucl. Acids Res.*, 38: D234–D236, 2010.

[44] M. O'Mahony. *Sensory Evaluation of Food: Statistical Methods and Procedures*. CRC Press, 1986. ISBN 0-8247-7337-3.

[45] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1992. ISBN 0-521-43108-5.

[46] `http://www.algorithm.cs.sunysb.edu/presto/`.

[47] M. Sharmin, M., and M. S. Rahman. Local search techniques for placing unique restriction sites in synthetic genomes. In *Proc of BICOB*, 2012.

[48] N. C. Jones and P. Pevzner. *Introduction to Bioinformatics Algorithms*. The MIT Press, 2004.

[49] M. Gerstein. *Bioinformatics Introduction*. University of Yale Press, 1999.

[50] `http://www.metaheuristics.net/`.

[51] E. Talbi. *Metaheuristics - From Design to Implementation*. Wiley, 2009. ISBN 978-0-470-27858-1.

[52] J. R. Coleman, D. Papamichail, S. Skiena, B. Futcher, E. Wimmer, and S. Mueller. Virus attenuation by genome-scale changes in codon pair bias. *Science*, 320 (5884):1784–1787, 2008.

[53] S. M. Richardson, S. J. Wheelan, R. M. Yarrington, and J. D. Boeke. Genedesign: Rapid, automated design of multikilobase synthetic genes. *Genome Res.*, 16(4): 550–556, 2006.

[54] D. L. Stein, C. Mungall, S. Shu, M. Caudy, M. Mangone, A. Day, E. Nickerson, E. J. Stajich, W. T. Harris, A. Arva, and S. Lewis. The generic genome browser: A building block for a model organism system database. *Genome Res.*, 12:1599–1610, 2002.

[55] E. Wimmer, S. Mueller, T. Tumpey, and J. Taubenberger. Synthetic viruses: a new opportunity to understand and prevent viral disease. *Nature Biotech.*, 27(12), 2009.

[56] M. Dayhoff. Computer analysis of protein evolution. *Sci Am*, 221(1):86–95, 1969.

[57] R. D. Fleischmann, O. White M. D. Adams, R. A. Clayton, E. F. Kirkness, A. R. Kerlavage, C. J. Bult, J. F. Tomb, B. A. Dougherty, J. M. Merrick, and et al. Whole-genome random sequencing and assembly of haemophilus influenzae rd. *Science*, 269(5223):496–512, 1995.

[58] R. F. Doolittle, M. W. Hunkapiller, L. E. Hood, S. G. Devare, K. C. Robbins, S. A. Aaronson, and H. N. Antoniades. Simian sarcoma virus onc gene, v-sis, is derived from the gene (or genes) encoding a platelet-derived growth factor. *Science*, 221 (4607):275–277, 1983.

[59] M. D. Waterfield, G. T. Scrace, N. Whittle, P. Stroobant, A. Johnsson, A. Wasteson, B. Westermark, C. H. Heldin, J. S. Huang, and T. F. Deuel. Platelet-derived growth factor is structurally related to the putative transforming protein p28sis of simian sarcoma virus. *Nature*, 304(5921):35–39, 1983.

[60] P. Hogeweg and D. B. Searls. The roots of bioinformatics in theoretical biology. *PLoS Computational Biology*, 7(3):90–96, 2011.

[61] P. Hogeweg. Simulating the growth of cellular forms. *Simulation*, 31(3):90–96, 1978.