M.Sc. Engg. Thesis

PRACTICAL MODELS AND ALGORITHMS FOR THE DNA FRAGMENT ASSEMBLY PROBLEM

By
Jesun Sahariar Firoz
Student No.: 1009052026

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

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

June 2012

The thesis titled "**Practical Models and Algorithms for the DNA Fragment Assembly Problem**," submitted by Jesun Sahariar Firoz, Roll No. 1009052026P, Session October 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 June 16, 2012.

Board of Examiners

1. _____

Dr. M. Sohel Rahman — Chairman (Supervisor)
Associate Professor
Department of Computer Science and Engineering
BUET, Dhaka 1000

2. _____

Dr. Abu Sayed Md. Latiful Hoque — Member (Ex-officio)
Professor & Head
Department of Computer Science and Engineering
BUET, Dhaka 1000

3. _____

Dr. M. Kaykobad — Member
Professor
Department of Computer Science and Engineering
BUET, Dhaka 1000

4. _____

Dr. Mohammed Eunus Ali — Member
Assistant Professor
Department of Computer Science and Engineering
BUET, Dhaka 1000

5. _____

Dr. Md. Shazzad Hosain — Member (External)
Assistant Professor
Department of Electrical Engineering & Computer Science
North South University, Dhaka

Candidate's Declaration

It is hereby declared that this thesis or any part of it has not been submitted elsewhere for the award of any degree or diploma.

Jesun Sahariar Firoz
Candidate

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgments

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

I express my heart-felt gratitude to my supervisor, Dr. M. Sohel Rahman for his constant supervision of this work. He helped me a lot in every aspect of this work and guided me with proper directions whenever I sought one. His patient hearing of my ideas, critical analysis of my observations and detecting flaws (and amending thereby) in my thinking and writing have made this thesis a success.

I would also want to thank the members of my thesis committee for their valuable suggestions. I thank Professor Dr. Abu Sayed Md. Latiful Hoque, Dr. M. Kaykobad, Dr. Mohammed Eunus Ali and specially the external member Dr. Md. Shazzad Hosain.

I also thank one of my friend Tanay Kumar Saha, who was my partner in the early works of my research. He was always there for me when I needed.

In this regard, I remain ever grateful to my beloved mother, who always exists as sources of inspiration behind every success of mine I have ever made.

Abstract

DNA fragment assembly problem is one of the crucial challenges faced by computational biologists where, given a set of DNA fragments, we have to construct a complete DNA sequence from them. As it is an NP-hard problem, accurate DNA sequence is hard to find. Moreover, due to experimental limitations, the fragments considered for assembly are exposed to additional errors while reading the fragments. In such scenarios, meta-heuristic based algorithms can come in handy. In this thesis, we have taken the first ever approach to generate noisy datasets using three realistic error models namely Sanger Sequencing error model, 454 Sequencing error model and Exact error model. Next, we analyze the performance of two swarm intelligence based algorithms namely Artificial Bee Colony (ABC) algorithm and Queen Bee Evolution Based on Genetic Algorithm (QEGA) to solve the fragment assembly problem and report quite promising results. We also propose two hybrid algorithms namely Genetic Algorithm with Simulated Annealing (GA+SA) and Genetic Algorithm with Hill Climbing (GA+HC) for noiseless and noisy datasets. Additionally, we evaluate the performance of Genetic algorithm with noisy datasets. Our main focus is to design meta-heuristic based techniques to efficiently handle DNA fragment assembly problem for noisy and noiseless data.

Chapter 1

Introduction

1.1 DNA Sequencing and Its Application

According to modern molecular biology and genetics, an organism's hereditary information is mainly encoded in Deoxyribonucleic Acid (DNA). The DNA is formed by a sequence of four types of molecules, called *nucleotides or bases*, namely, Adenine (A), Thymine (T), Cytosine (C) and Guanine (G). The process of DNA sequencing provides us with the most basic information of all: the sequence of nucleotides. With this knowledge, for example, we can locate regulatory and gene sequences, make comparisons between homologous genes across species and identify mutations. Moreover, decoding a DNA sequence is vital to understand the function as well as malfunction of living things [3].

The DNA sequence information is vital for medical, agricultural and many other research areas. Inexpensive, time-efficient and accurate DNA sequencing will be a major accomplishment not only for the field of Genomics, but for the entire human civilization because, for the first time, individuals will be able to have their entire DNA sequenced. Utilizing this information, it

is speculated that health care professionals, such as physicians and genetic counselors, will eventually be able to use genomic information to predict what diseases a person may get in the future and attempt to either minimize the impact of that disease or avoid it altogether through the implementation of personalized, preventive medicine. DNA sequencing will allow health care professionals to analyze the entire human genome of an individual and thereby detect all disease-related genetic variants, regardless of the genetic variant's prevalence or frequency. This will enable the rapidly emerging medical fields of Predictive Medicine and Personalized Medicine and will mark a significant leap forward for the clinical genetic revolution. So, DNA sequencing is clearly of great importance for research into the basis of genetic disease because it will enable us to test for genetic markers associated with the disease [3].

Now-a-days, screening of newborn for childhood diseases allows detection of rare disorders that can be prevented or better treated by early detection and intervention. Specific genetic tests are also available to determine when a child's symptoms appear to have a genetic basis. Full DNA sequencing, in addition has the potential to reveal a large amount of information (such as carrier status for autosomal recessive disorders, genetic risk factors for complex adult-onset diseases, and other predictive medical and non-medical information) that is currently not completely understood, may not be clinically useful to the child during childhood, but may become useful for the individual upon reaching adulthood [3].

Accordingly, the Human Genome Project (HGP) was launched in October 1990 with a primary goal of determining the sequence of chemical base pairs which make up DNA, and of identifying and mapping the approximately 20,000-25,000 genes of the human genome from both a physical and functional standpoint [4]. All humans have unique gene sequences. Therefore the data published by the HGP does not represent the exact sequence of every individual's genome. It is the combined "reference genome" of a small number of anonymous donors [4]. The HGP genome is a scaffold for future work in identifying differences among individuals. So automating DNA se-

quencing,for knowing individual's DNA in feasible time, by using computer software is imperative.

1.2 DNA Fragment Assembly Problem in the Context of DNA Sequencing

In DNA sequencing, assembling fragments or "reads" of DNA for the reconstruction of long continuous and least ambiguous contigs (i.e., groups of overlapping fragments of a DNA) is a difficult but an important step. In this context, DNA fragment assembly problem is one of the crucial challenges faced by computational biologists. To illustrate the problem with a metaphor, let us consider a scenario [36]: imagine several copies of a magazine cut into millions of pieces. Each copy is cut in a different way, so a piece from one copy may overlap pieces from another. Assuming that some large number of pieces are just sort of lost, and the remaining pieces are splashed with ink, can we recover the original text? This, essentially, is the problem of fragment assembly in DNA sequencing.

To read the DNA fragments, a process named *shotgun sequencing* [60] is used. In this method, multiple copies of the DNA sequence are first generated through a process called *amplification*. Then these sequences are cut at random points keeping in mind that we can only directly read a sequence of several hundred base pairs (bps) long. With these fragments, we try to reconstruct the overlapping DNA sequence as accurately as possible. So, in DNA Fragment Assembly Problem (FAP), we are given a set of large number of DNA fragments, possibly with errors and we are asked to find the correct sequence of the DNA by finding the permutation of fragments that best represent the original DNA sequence.

DNA FAP finds its motivation from the limitation of current technology, which enables us to read only several hundred of bps of a single DNA at a time. Consequently, we need to read fragments of the DNA but not the

whole sequence at a time. During this process base pairs may be removed, misread or inserted. Even if we disregard the presence of noise, this problem is NP-hard [44]. Note that, given k fragments, there are $2^k * k!$ combinations.

1.3 Our Contribution

In this thesis, we focus on evaluating the performance of various meta-heuristics for solving the DNA fragment assembly problem with noiseless and noisy data. Notably, the recent work of [50] also focused on noisy dataset with an important drawback. The drawback of this technique is that no particular sequencing error model was taken into consideration. To overcome this drawback and for the construction of a realistic read data set, here we use a sequencing simulator MetaSim [59]. In this simulator, the user is able to choose from different (adaptable) error models of current sequencing technologies (e.g. Sanger [47, 48], Rochei's 454 [45] and Illumina (former Solexa) [17]). We have used Sanger, 454 and Exact error models of MetaSim to generate the noisy dataset. Beside this, we have collected noiseless dataset generated with the help of another simulator Genfrag [29].

After generating the datasets with and without errors, we have used ParadisEO [22], a software framework for meta-heuristics, to solve the DNA fragment assembly problem (FAP). ParadisEO is a C++ white-box object-oriented framework dedicated to the reusable design of meta-heuristics. We have implemented genetic algorithm with simulated annealing and genetic algorithm with hill climbing for solving FAP in the ParadisEO framework and compared the relative fitness achieved in each case.

Additionally, we have implemented Artificial Bee Colony (ABC) algorithm and Queen Bee Evolution Based on Genetic Algorithm (QEGA) in C++ to solve the DNA fragment assembly problem as accurately as possible and compared the relative fitness achieved in noisy and noiseless case.

As we have mentioned before, in most of the literature it was assumed

that, the fragments being read are correct. We have eliminated this assumption by incorporating error models in generating artificial fragments to be sequenced [59]. We use either simulated annealing or hill climbing along with genetic algorithm so that worse individuals in the population are taken into consideration. In this way, the chance of completely eliminating a probably misread fragment is minimized and we are bound to find more realistic solutions. We exploit the probabilistic behavior of ABC or QEGA for the same purpose. Also, using forward and backward read technique of MetaSim [59], we have been able to emulate a more realistic input sequence mimicking experimental output. These facts give us a promising insight about the feasibility of using meta-heuristics to solve DNA fragment assembly problem.

In particular, Our contributions are summarized as follows.

- We have taken the first ever approach to generate noisy datasets using realistic error models namely:
  - Sanger Sequencing Error Model
  - 454 Sequencing Error Model
  - Exact Error Model
- We have used MetaSim and Genfrag tools for generating noisy and noiseless data respectively.
- We have tried to solve the DNA Fragment Assembly Problem (DNA FAP) for noisy and noiseless datasets with the following algorithms:
  - Artificial Bee colony Algorithm (ABC_FAP)
  - Queen Bee Evolution Based on Genetic Algorithm (QEGA_FAP)
  - Genetic Algorithm with Simulated Annealing (GA+SA)
  - Genetic Algorithm with Hill Climbing (GA+HC)
  - Genetic Algorithm (for noisy datasets)
- We have implemented GA, GA+SA and GA+HC in ParadisEO framework, which is a template-based meta-heuristic library.

1.4 Organization

The rest of the thesis is organized as follows. Chapter 2 provides background information, problem definition and overview of two sequencing techniques. Chapter 3 gives a general overview of meta-heuristics. Chapter 4 discusses previous works related to DNA FAP. Chapter 5 introduces the three models that have been used to generate noisy datasets. Chapter 6 describes two of our meta-heuristic algorithms namely Artificial Bee Colony (ABC) algorithm and Queen-bee Evaluation based on Genetic Algorithm (QEGA). Chapter 7 presents our hybrid meta-heuristic based algorithms namely genetic algorithm with simulated annealing (GA+SA) and genetic algorithm with hill climbing (GA+HC). In Chapter 8, we report the results obtained by our algorithms and present a comparative study. Chapter 9 concludes the thesis with future research directions.

Chapter 2

Preliminaries

2.1 Introduction

Now-a-days, genomic data analysis using computational approaches is very popular. The primary goal of any genomic project is to determine the complete sequence of the genome (mainly DNA) and its genetic content. Thus, a genome project is accomplished in two steps: the first step is the *genome sequencing* and the second is the *genome annotation* (i.e., the process of identifying the boundaries between genes and other features in raw DNA). The DNA fragment assembly is performed at the very beginning of the process.

In this chapter, we present some background and preliminaries. Most of the definitions and procedures presented in this section follows from[44].

2.2 Definitions

The DNA fragment assembly occurs in the frame of determining the specific function of every gene composing a given DNA chain. Other steps depend on its accuracy. The input of the DNA fragment assembly problem is a set of fragments that are randomly cut from a DNA sequence. The DNA is

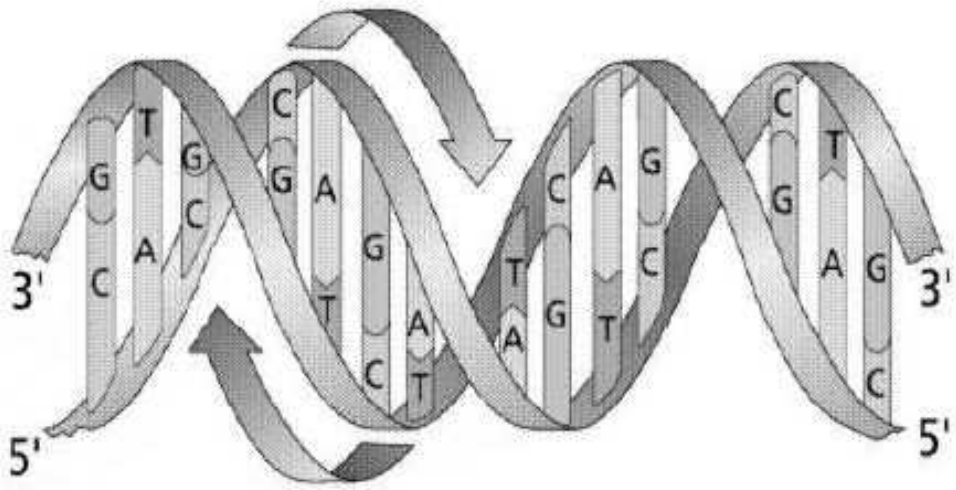


Figure 2.1: Double Stranded DNA [44].

a double helix of two anti-parallel and complementary nucleotide sequences (Figure 2.1). One strand is read from $5'$ to $3'$ and the other from $3'$ to $5'$. There are four kinds of nucleotides in any DNA sequence:Adenine (A), Thymine (T), Guanine (G), and Cytosine (C).

To further understand the problem, we need to know the following basic terminology:

- *Fragment:* A short sequence of DNA with length up to 1000 bps.
- *Shotgun data:* A set of fragments.
- *Prefix:* A substring comprising the first k characters of a fragment f.
- *Suffix:* A substring comprising the last k characters of a fragment f.
- *Overlap:* Common sequence between the suffix of one fragment and the prefix of another fragment.
- *Layout:* An alignment, i.e., placement of a collection of fragments based on the overlap order, i.e., the fragment order in which the fragments must be joined (see Figure 2.2).

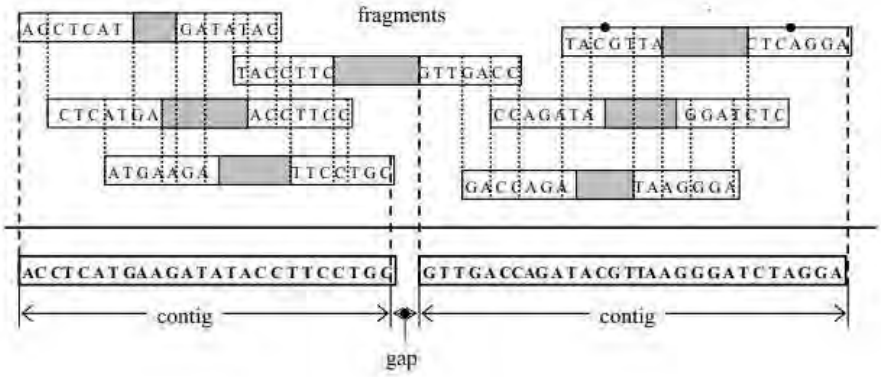


Figure 2.2: Formation of contigs and a final layout (figure borrowed from [41])

- *Contig:* A layout consisting of contiguous overlapping fragments, i.e., a sequence in which the overlap between adjacent fragments is greater than a predefined threshold (see Figure 2.2).

- *Consensus:* A sequence or string derived from the layout by taking the majority vote for each column of the layout.

- *x–mer:* The term x–mer (where x can be virtually any consonant of choice) usually refers to a specific x–tuple of nucleotides that can be used to identify certain regions within DNA. For example, A sequence of dimers = AGAGAGAGAGAGAG and A sequence of trimers = AAGAAGAAGAAG. Here AG and AAG are dimers and trimers respectively.

To measure the quality of a consensus, we can look at the distribution of the coverage. Coverage at a base position is defined as the number of fragments at that position. It is a measure of the redundancy of the fragment data. It denotes the number of fragments, on average, in which a given nucleotide in the target DNA is expected to appear and is computed as follows [60]:

$$Coverage = \frac{\sum_{i=0}^{n} \text{length of the fragment } i}{\text{target sequence length}} \tag{2.1}$$

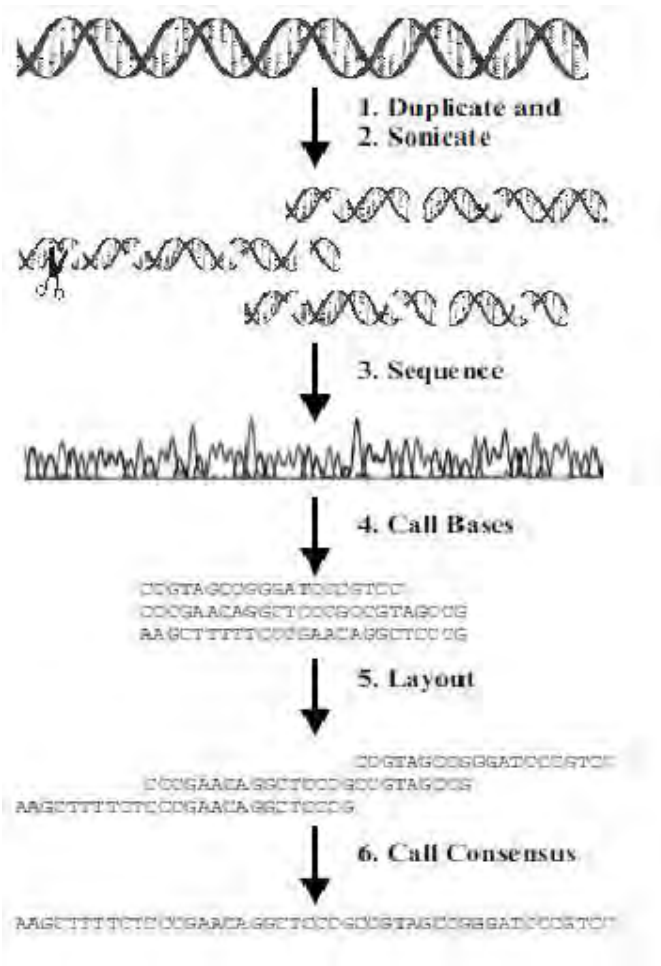


Figure 2.3: Graphical representation of DNA sequencing and assembly (figure borrowed from [44])

Where n denotes no. of fragments. For example, a hypothetical genome with 2,000 base pairs reconstructed from 8 reads with an average fragment length of 500 nucleotides will have coverage of 2.

2.3 DNA Sequencing Process

To determine the function of specific genes, scientists read the sequence of nucleotides comprising a DNA sequence in a process called DNA sequencing (Figure 2.3). At present, strands of DNA that are longer than 600 base pairs cannot routinely be sequenced accurately. Hence, large strands of DNA need

to be broken into small fragments for sequencing. Logically, the whole process of DNA sequencing is divided into two parts: one is the biological part of cloning, fragmenting, and reading, and the other one is the computational part of assembling the fragments.

2.3.1 Biological Part

The basic procedure starts with a large number of copies of DNA whose sequence we need to find out. The genome is then physically cut into a large number of random fragments. Fragments that are too large or too small are then discarded. The length of short fragments are about 2Kbp, and the long ones are about 10Kbp. The fragments are then inserted into the DNA of a bacterial virus (phage), called *vector*. Typically one vector contains one fragment. The fragments are called *inserts* and the set of inserts with similar size, a *library*. Next, a bacterium is infected with a single vector, which generates clones of the vector as well as the insert (the fragment) within it. Then, the base pair at both ends of all the fragments are read with DNA sequencer as shown in Figure 2.4. Only about 500 to 600 bps can be read using present sequencer technology. This read length depends on the passing speed in the capillary of the sequencer. But even if done meticulously, a read length of more than 1000 bp is not possible. The base sequence at each of the ends of fragment read by the sequencer is called a *read*, and the pair of reads from the two ends is called *mate–pairs*.

In summary, multiple exact copies of the original DNA sequence are made. Each copy is then cut into short fragments at random positions (*duplicate and sonicate phases*). Then this biological material is "converted" to computer data (*sequence and call bases phases*). *Base calling* is the process of assigning bases. All these steps take place in the laboratory.

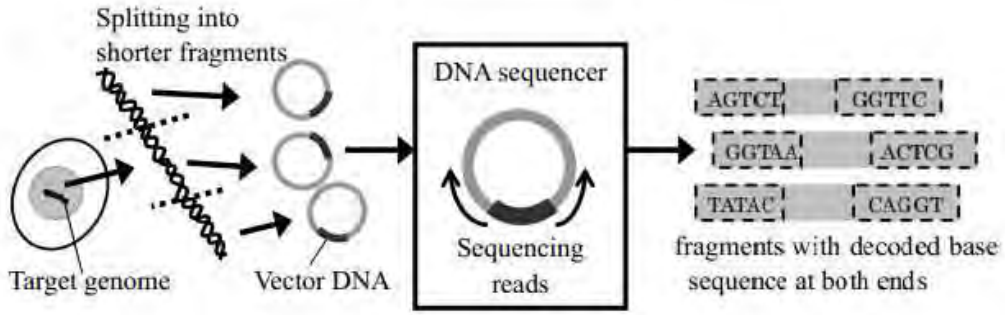


Figure 2.4: Biological part of the DNA sequencing process [41].

2.3.2 Computational Part

After the set of fragments are obtained, traditional assembly approach is followed in the following phases in the given order: *overlap, layout,* and then *consensus*. To ensure that enough fragments overlap, the reading of fragments continues until the coverage is satisfied. In the *overlap phase*, the best or longest match between the suffix of one sequence and the prefix of another are found. *Layout phase* consists of finding the order of fragments based on the computed similarity score. *Consensus phase* consists of deriving the DNA sequence from the layout. In what follows, we give a brief description of each of the three phases.

Overlap Phase

This phase consists in finding the best or longest match between the suffix of one sequence and the prefix of another. In this step, we compare all possible pairs of fragments to determine their similarity. Usually, the dynamic programming algorithm applied to semiglobal alignment is used in this step. The intuition behind finding the pairwise overlap is that fragments with a significant overlap score are very likely to be next to each other in the target sequence.

Layout Phase

Layout phase consists of finding the order of fragments based on the computed similarity score. Several issues complicate this layout phase. These are discussed below:

- **Unknown orientation:** After the original sequence is cut into many fragments, the orientation is lost. The sequence can be read in either $5'$ to $3'$ or $3'$ to $5'$. One does not know which strand should be selected. If one fragment does not have any overlap with another, it is still possible that its reverse complement might have such an overlap.
- **Base call errors:** There are three types of base call errors, namely, substitution, insertion, and deletion errors. They occur due to experimental errors in the electrophoresis procedure. Errors affect the detection of fragment overlaps. The consensus determination requires multiple alignments in high coverage regions.
- **Incomplete coverage:** It happens when the algorithm is not able to assemble a given set of fragments into a single contig.
- **Repeated regions:** Repeats are sequences that appear two or more times in the target DNA. Repeated regions have caused problems in many DNA sequencing projects, and none of the current assembly programs can handle them perfectly [44].
- **Chimeras and contamination:** Chimeras arise when two fragments that are not adjacent or overlapping on the target molecule join together into one fragment. Contamination occurs due to the incomplete purification of the fragment from the vector DNA.

After the order is determined, progressive alignment is applied to combine all the pairwise alignments obtained in the overlap phase.

Consensus Phase

Consensus phase consists of deriving the DNA sequence from the layout. The most common technique used in this phase is to apply the majority rule

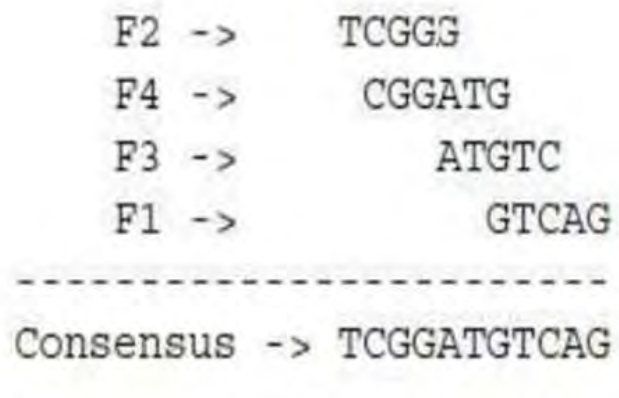


Figure 2.5: Layout and Consensus for the example in Section 2.3.3.

in building the consensus. The majority rule states that, given a specific location of overlapping fragments, we put the base (A, T, C or G) at that specific location with maximum appearance.

2.3.3 An Example

We now give an example borrowed from [44]. Given a set of fragments {F1 = GTCAG, F2 = TCGGA, F3 = ATGTC, F4 = CGGATGg}, assume the four fragments are read from $5'$ to $3'$ direction. First, we need to determine the overlap of each pair of the fragments by the using semiglobal alignment algorithm. Next, we determine the order of the fragments based on the overlap scores, which are calculated in the overlap phase. Suppose we have the following order: F2 F4 F3 F1. Then, the layout and the consensus for this example can be constructed as shown in Figure 2.5. In this example, the resulting order allows to build a sequence having just one contig.

2.4 Sanger Sequencing Process

In 1974, two DNA sequencing methods were invented independently and simultaneously in Cambridge, England by Fred Sanger and in Cambridge, Massachusetts by Walter Gilbert. Gilbert, used a "chemical cleavage protocol", while Sanger, designed a procedure similar to the natural process of

DNA replication. Even though both teams shared the 1980 Nobel Prize, Sanger's method became the standard because of its practicality [5].

Sangers method takes advantage of how cells make copies of DNA [36]. Cells copy a strand of DNA nucleotide by nucleotide in a reaction that adds one base at a time. Sanger realized that he could make copies of DNA fragments of different lengths if he starved the reaction of one of the four bases: a cell can only copy its DNA while it has all of the bases in supply. For a sequence ACGTAAGCTA, starving at T would produce a mixture of the fragments ACG and ACGTAAGC. By running one starvation experiment for each of A, T , G and C and then separating the resulting DNA fragments by length, one can read the DNA sequence. Each of the four starvation experiments produces a ladder of fragments of varying lengths called the *Sanger ladder*. For example The Sanger ladder for T shows the lengths of all subfragments ending at T and therefore reveals the set of positions where T occurs. The DNA sequencing machines measure the lengths of DNA fragments in the Sanger ladder, but even this task is difficult; we cannot measure a single DNA fragment, but must measure billions of identical fragments. The technique to create identical fragments is described in Subsection 2.3.1. After creating the identical fragments, they are read by two machines: *automated gel sequencers* that use electrophoresis and *fluorescent markers*. These machines are used to determine the sequence of the nucleotides in one fragment. The ability of these machines to read consecutive pieces of DNA degrades quickly with the length of the sequence, and today a sequencing machine can read up to $\approx$700 consecutive base pairs of a fragment of DNA, depending on the degree of accuracy desired.

The Sanger approach culminated in the sequencing of a 5386-nucleotide virus in 1977 and a Nobel Prize shortly thereafter. Since then the amount of DNA sequence data has been increasing exponentially, particularly after the launch of the Human Genome Project in 1989. By 2001, it had produced roughly 3 billion nucleotide sequence of the human genome.

Classical Sanger sequencing relies on base-specific chain terminations in

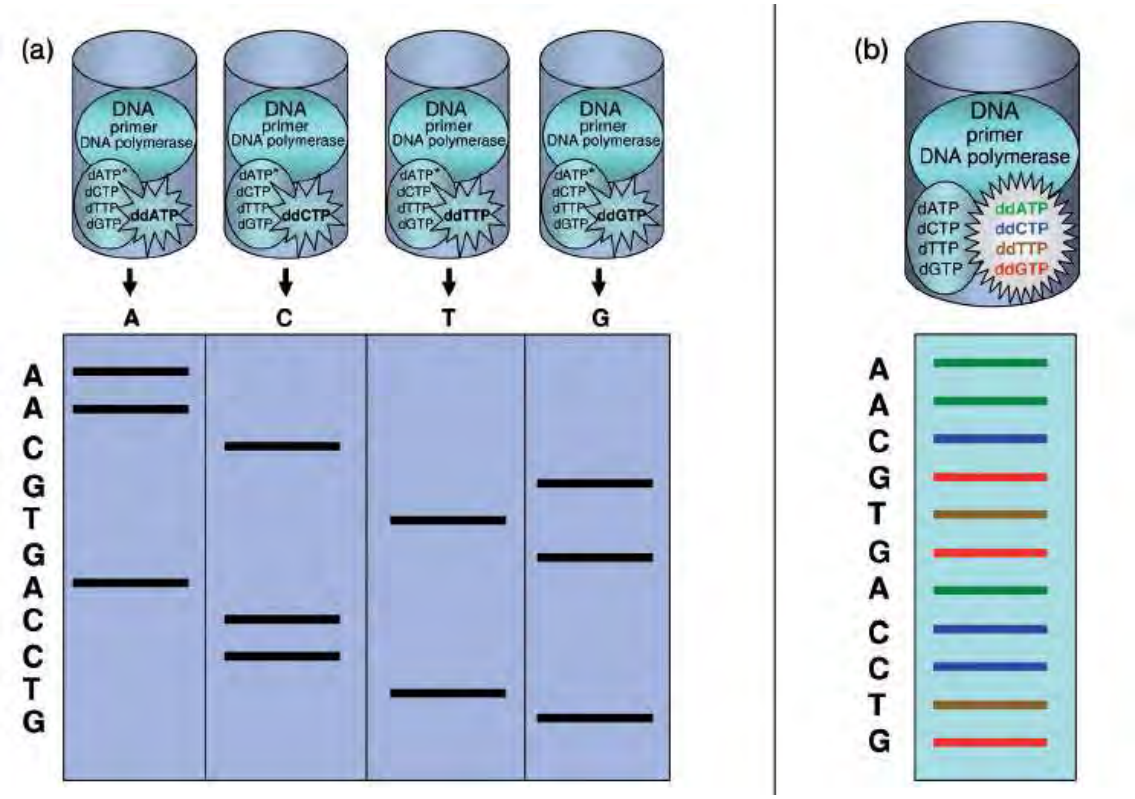


Figure 2.6: Schematic principle of the Sanger sequencing method. (a) Four separate DNA extension reactions are performed, each containing a single-stranded DNA template, primer, DNA polymerase, and all four dNTPs to synthesize new DNA strands. Each reaction is spiked with a corresponding dideoxynucleoside triphosphate (ddATP, ddCTP, ddTTP, or ddGTP). In the presence of dNTPs, one of which is radioactively labeled (in this case, dATP), the newly synthesized DNA strand would extend until the available ddNTP is incorporated, terminating further extension. Radioactive products are then separated through four lanes of a polyacrylamide gel and scored according to their molecular masses. Deduced DNA sequence is shown on the left. (b) In this case, instead of adding radioactive dATP, all four ddNTPs are labeled with different fluorescent dyes. The extension products are then electrophoretically separated in a single glass capillary filled with a polymer. Similar to the previous example, DNA bands move inside the capillary according to their masses. Fluorophores are excited by the laser at the end of the capillary. The DNA sequence can be interpreted by the color that corresponds to a particular nucleotide (Figure borrowed from [35]).

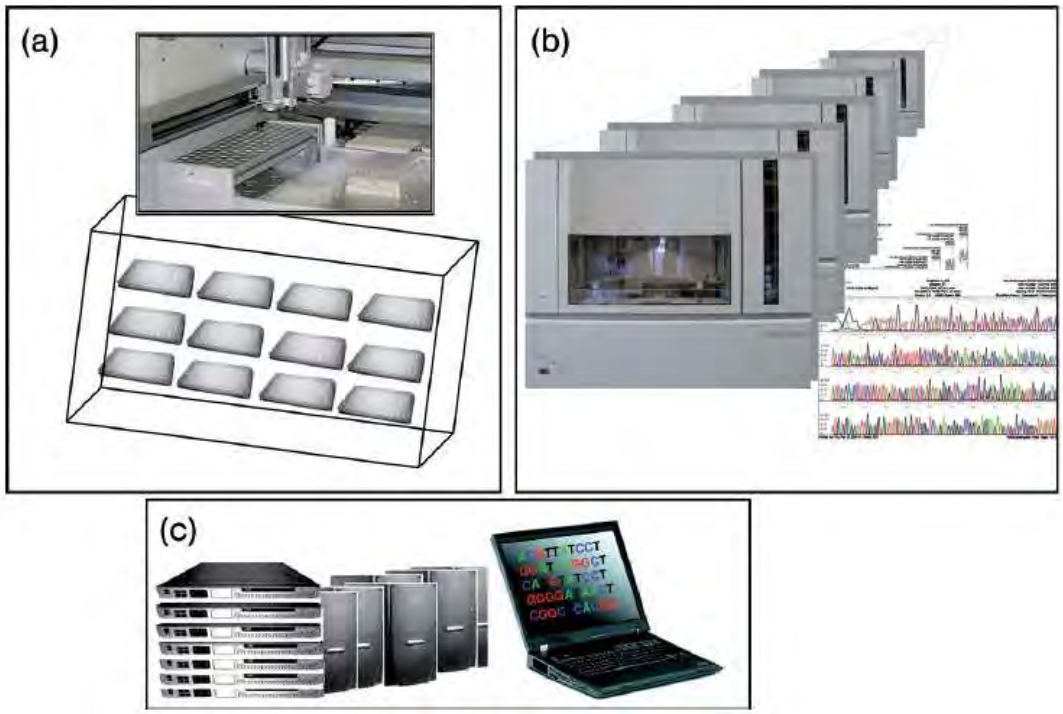


Figure 2.7: Sanger sequencing pipeline. (a) DNA clone preparation usually starts with the isolation of total DNA (e.g., whole genomic DNA from an organism or already fragmented DNA, cDNA, etc.), followed by further fragmentation and cloning into a vector for DNA amplification in bacterial cells. As a result, millions of individual bacterial colonies are produced and individually picked into multiwell plates by liquid-handling robots for isolation of amplified DNA clones. This DNA then goes through a sequencing reaction described in Figure 2.6(b) Processed sequenced DNA undergoes capillary electrophoresis where labeled nucleotides (bases) are collected and scanned by the laser producing raw sequencing traces. (c) Raw sequencing information is converted into computer files showing the final sequence and quality of every scanned base. The resultant information is stored on dedicated servers and also is usually submitted into free public databases, such as the GeneBank and Trace Archive (Figure borrowed from [35]).

four separate reactions (A, G, C and T) corresponding to the four different nucleotides in the DNA makeup (Figure 2.6) [35]. In the presence of all four $2'$ deoxynucleotide triphosphates (dNTPs), a specific $2'$, $3'$-dideoxynucleotide triphosphate (ddNTP) is added to every reaction, for example, ddATP to the "A" reaction and so on. The extension of a newly synthesized DNA strand terminates every time the corresponding ddNTP is incorporated. As the ddNTP is present in minute amounts, the termination happens rarely and stochastically, resulting in a cocktail of extension products where every position of an "N" base would result in a matching product terminated by incorporation of ddNTP at the $3'$ end. The second novel aspect of the method is the use of radioactive phosphorus or sulfur isotopes incorporated into the newly synthesized DNA strand through a labeled precursor (dNTP or the sequencing primer), therefore, making every product detectable by radiography. Finally, as each extension reaction results in a very complex mixture of large radioactive DNA products, probably the most crucial achievement is the development of ways to individually separate and detect these molecules. The innovative use of a polyacrylamide gel (PAG) allowed very precise sizing of termination products by electrophoresis followed by in situ autoradiography. Later, the autoradiography is partially replaced by less hazardous techniques such as silver staining of DNA in PAGs.

But slab PAGs are very slow and laborious and cannot be readily applied to interrogating large genomes. The next two major technological breakthroughs took place in (i) 1986 when a Caltech team (led by Leroy Hood) and ABI developed an automated platform using fluorescent detection of termination products separating four-color-labeled termination reactions in a single PAG tube and in (ii) 1990 when the fluorescent detection was combined with electrophoresis through a miniaturized version of PAGs, namely, capillaries. Capillary electrophoresis (CE), by taking advantage of a physically compact DNA separation device coupled with laser-based fragment detection, eventually became compatible with 96- and 384-well DNA plate format making highly parallel automation a feasible reality. Finally, the combination of dideoxy-based termination chemistry, fluorescent labeling, capillary separation, and computer-driven laser detection of DNA fragments has established

the four elegant cornerstones. on which modern building of high-throughput Sanger sequencing stands today. Figure 2.7 shows the Sanger Sequencing pipeline.

2.5 454 Sequencing Process

Recently, 454 Life Sciences announced a new, highly parallel sequencing system with significantly higher throughput than achievable with previous methods. The name 454 was the code name by which the project was referred to at CuraGen Corporation and the numbers have no special meaning.
Using the current Sanger method of cloning DNA in bacteria, the amplification process currently takes approximately three weeks and also introduces bias in the DNA samples. On the other hand, 454 sequencing takes only eight hours to do the same job. It uses emulsion-based PCR amplification of a large number of DNA fragments and high-throughput parallel pyro-sequencing. A detailed description of emulsion-based PCR amplification and pyro-sequencing is out of scope of this thesis and hence not given here. The system is reportedly able to sequence 25 million bases within four hours. Cloning of the target DNA fragments is not necessary and the method is much cheaper, per base, than other existing methods. Drawbacks of the method are shorter read lengths of about 100 bases, in contrast to 800 bases using Sanger sequencing, and a higher error rate. Further, sequencing of paired-end reads is not yet possible.

In pyro-sequencing, the intensity of emitted light is used to estimate the length of homopolymers, i.e., runs of identical nucleotides in a sequence. During sequencing, the four DNA composing nucleotides are periodically flowed over the inserts (i.e. small pieces of DNA) to be sequenced. A nucleotide complementary to the template strand generates a light signal. The light signal is recorded by the CCD camera (Figure 2.8). The signal strength is proportional to the number of nucleotide incorporated. So, within each flow, the intensity of the signal emitted reflects the number of nucleotides incorporated and thus the length of the homopolymer under consideration.The

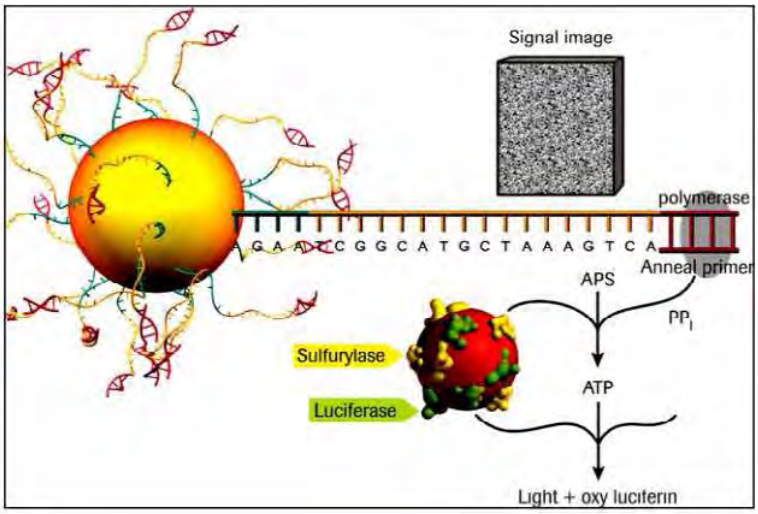


Figure 2.8: 454 sequencing process (Figure borrowed from [1]).

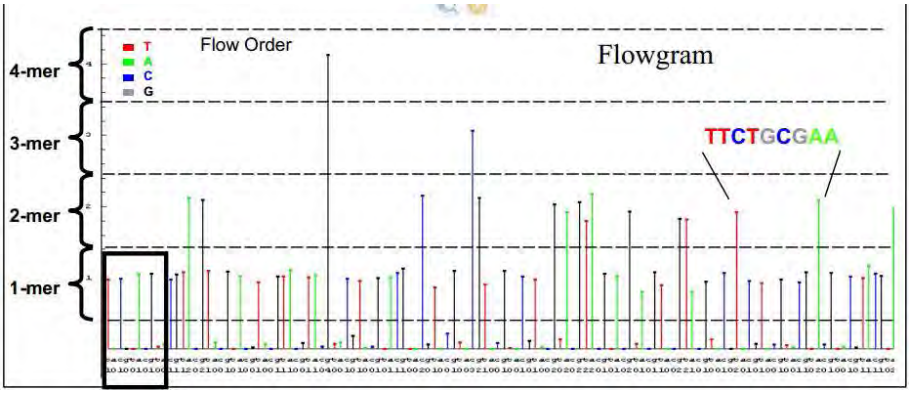


Figure 2.9: Flowgram obtained from 454 sequencing process (Figure borrowed from [1]).

intensity of light generated during the flow of a single nucleotide varies proportionately with the consecutive number of complementary nucleotides on the single-stranded DNA fragment being analyzed. For example, if there are three consecutive As in the single-stranded fragment, the amount of light generated would be three times that of a single A in the fragment (Figure 2.9). The signals created in the sequencing process are then analyzed by the 454 Sequencing Systems software to generate millions of sequenced bases per hour from a single run.

For 454 sequencing [6] process, let r denote the length of a given homopolymer. The emitted light intensity can be modeled by a normal distribution $N(\mu, \sigma)$, with mean $\mu = \sigma$ and standard deviation $\sigma = k * \sqrt{r}$, where k is a fixed proportionality factor ($k \approx 0.15$) [45].

Although basic statistics imply that the standard deviation should grow with the square root of r, for the light intensity emitted during 454-sequencing, it is reported to be $\sigma = k * r$.

2.6 Summary

In this chapter, we have given background information and concepts required to understand the DNA fragment assembly problem. We briefly discussed the DNA sequencing process and introduced the fragment assembly problem in its context. We have discussed the errors that occur during the fragment assembly. Lastly, we have mentioned two sequencing process, namely the Sanger and 454 sequencing. In the next chapter, we give an overview of the basics of different meta-heuristic techniques. Later, we apply various meta-heuristics to solve the DNA FAP.

Chapter 3

Meta-heuristics

3.1 Introduction

In this chapter, we give an introduction to meta-heuristics for solving different problems. We briefly discuss on single-solution and population-based approaches.

3.2 Meta-heuristics

For many optimization problems, computing optimal solutions is intractable. For this reason, we are generally interested in reasonably "good" solutions, which are obtained by heuristic or meta-heuristic algorithms. Meta-heuristics provide acceptable solutions in a reasonable time for solving hard and complex problems [63]. Meta-heuristics 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, meta-heuristics do not define how close are the obtained solutions from the optimal ones.

The word *heuristic* originates from the old Greek word *heuriskein*, which means the art of discovering new strategies (rules) to solve problems. The suffix *meta*, which is also a Greek word, means *upper level methodology*. Meta-

heuristic search methods can be defined as upper level general methodologies (templates) that can be used as guiding strategies in designing underlying heuristics to solve specific optimization problems [63].

Of great importance for the functioning of a meta-heuristic are the concepts called *diversification* and *intensification* [18]. 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. Each meta-heuristic application is characterized by a balance between diversification and intensification. This is important, on one side to quickly identify regions in the search space with high quality solutions, and on the other side not to waste too much time in regions of the search space which are either already explored or which do not provide high quality solutions.

3.3 Classification

There are different ways to classify and describe meta-heuristic algorithms, each of them being the result of a specific viewpoint. Some classifications are described below [63]:

- **Nature inspired versus non-nature inspired:** Many meta-heuristics are inspired by natural processes: evolutionary algorithms and artificial immune systems from biology; ants, bee colonies, and particle swarm optimization from swarm intelligence into different species (social sciences); and simulated annealing from physics.
- **Memory usage versus memoryless methods:** Some meta-heuristic algorithms are *memoryless*; that is, no information extracted dynamically is used during the search. Some representatives of this class are local search, GRASP, and simulated annealing. While other meta-heuristics use a *memory* that contains some information extracted online during the search. Examples include short-term and long-term memories in tabu search.
- **Deterministic versus stochastic:** A *deterministic* meta-heuristic

solves an optimization problem by making deterministic decisions (e.g., local search, tabu search). In *stochastic* meta-heuristics, some random rules are applied during the search (e.g., simulated annealing, evolutionary algorithms). In deterministic algorithms, using the same initial solution will lead to the same final solution, whereas in stochastic meta-heuristics, different final solutions may be obtained from the same initial solution. This characteristic must be taken into account in the performance evaluation of meta-heuristic algorithms.

- **Population-based search versus single-solution based search:** *Single-solution based algorithms* (e.g., local search, simulated annealing) manipulate and transform a single solution during the search while in *population-based algorithms* (e.g., particle swarm, evolutionary algorithms) a whole population of solutions is evolved. These two families have complementary characteristics as discussed below. Single-solution based meta-heuristics are *exploitation* oriented; they have the power to intensify the search in local regions. Population-based meta-heuristics are *exploration* oriented; they allow a better diversification in the whole search space. Generally, algorithms that work on a single solution at any time are referred to as *trajectory methods*. They comprise all meta-heuristics that are based on local search, such as tabu search, iterated local search and variable neighborhood search. They all share the property that the search process describes a trajectory in the search space. Population-based meta-heuristics, on the contrary, either perform search processes which can be described as the evolution of a set of points in the search space (as for example in evolutionary computation), or they perform search processes which can be described as the evolution of a probability distribution over the search space (as for example in ant colony optimization).

- **Iterative versus greedy:** In *iterative* algorithms, we start with a complete solution (or population of solutions) and transform it at each iteration using some search operators. *Greedy* algorithms start from an empty solution, and at each step a decision variable of the problem is assigned until a complete solution is obtained.

3.4 Single-solution Based Algorithms

As stated previously, *Single-solution based algorithms* manipulate and transform a single solution during the search procedure. Examples include simulated annealing and hill climbing. To optimize a candidate solution we need to be able to do four things [43]:

- Provide one or more initial candidate solutions. This is known as the *initialization* procedure.
- Assess the quality of a candidate solution. This is known as the *assessment* procedure.
- Make a copy of a candidate solution.
- Tweak a candidate solution, which produces a randomly slightly different candidate solution. This, plus the copy operation are collectively known as the *modification* procedure.

The single-solution based meta-heuristics algorithms typically provide a *selection* procedure that decides which candidate solutions to retain and which to reject as it wanders through the space of possible solutions to the problem.

3.5 Population-based Methods

Population-based meta-heuristics, (e.g., particle swarm, genetic algorithms, bee base algorithms etc.) at each iteration, deal with a set of solutions rather than a single solution. From this set of solutions the population of the next iteration is produced by the application of certain operators. Most population-based methods steal concepts from biology. One particularly popular set of techniques, collectively known as *Evolutionary Computation (EC)*, borrows liberally from population biology, genetics, and evolution. An algorithm chosen from this collection is known as an *Evolutionary Algorithm (EA)*. Most EAs may be divided into *generational algorithms*, which update

the entire sample once per iteration, and *steady-state algorithms*, which update the sample a few candidate solutions at a time. Common EAs include the *Genetic Algorithm (GA)* and *Evolution Strategies (ES)*; and there are both generational and steady-state versions of each. Some definitions related to population based methods are described below [43]:

- **Individual:** A candidate solution.
- **Child and Parent**: A child is the tweaked copy of a candidate solution (its parent).
- **Population:** Set of candidate solutions.
- **Fitness:** Quality.
- **Fitness landscape:** Quality function.
- **Fitness assessment or evaluation:** Computing the fitness of an individual.
- **Selection:** Picking individuals based on their fitness.
- **Mutation:** Plain tweaking. We will give details about the mutation operator we use in our algorithms in Section 7.5.
- **Recombination or Crossover:** A special tweak which takes two parents, swaps sections of them, and (usually) produces two children. This is mostly related to vector representation. We will give details about the crossover operator we use in our algorithms in Section 6.6.
- **Breeding:** Producing one or more children from a population of parents through an iterated process of selection and tweaking (typically mutation or recombination).
- **Genotype:** An individuals data structure, as used during breeding.

The basic generational evolutionary computation algorithm first constructs an initial population, then iterates through three procedures. First, it **assesses** the fitness of all the individuals in the population. Second, it uses this fitness information to **breed** a new population of children. Third, it

joins the parents and children in some fashion to form a new next-generation population, and the cycle continues. Breed operation usually has two parts: **Selecting** parents from the old population, then Tweaking them (usually *Mutating* or *Recombining* them in some way) to make children. The *Join* operation usually either completely replaces the parents with the children, or includes fit parents along with their children to form the next generation.

3.6 Summary

This chapter has presented a high level introduction to meta-heuristics for solving different problems. We have discussed on single-solution and population-based approaches. In the following chapters, we propose algorithms based on population-based approaches to solve DNA FAP. Before that, in the next chapter, we mention different methods in the literature, previously employed, to solve the FAP.

Chapter 4

Related Works

4.1 Introduction

In this chapter, we give descriptions of various state-of-the-art techniques found in the literature for solving DNA fragment assembly problem. Most of the techniques applied different heuristics. Some assemblers are also developed in the last couple of years to automate the computational part of DNA sequencing process.

4.2 State-of-the-art

Many deterministic and stochastic search techniques have been proposed to solve DNA fragment assembly problems [46]. For instance, [61] and [67] proposed deterministic greedy search algorithm to solve the problem. But a drawback of this type of algorithm is that manual manipulation on the computer-generated result is required to obtain a biologically plausible final result. Others investigated deterministic search algorithms like a branch-and-cut algorithm [31] and an overlap-graph based algorithm [20]. Although mathematical analyses of these two algorithms indicate that the algorithms are capable to solve the assembly problem, neither algorithm has as yet been successfully applied to problems generated using real data. Researchers also investigated stochastic search algorithms such as a simulated annealing algo-

rithm [25, 21] and a genetic algorithm [54] for DNA FAP. Promising results have been reported in [54] where the genetic algorithm has shown to have outperformed a greedy search technique. Additionally, the need for manual intervention is also eliminated in this case. Although a significant improvement over the greedy search result has been achieved, later, it was pointed out that the search efficiency could be further improved if the redundancy in the solution representation could be eliminated from the search algorithm [54].

In [46], approaches based on Ant Colony System (ACS) are proposed to solve FAP. The authors in [46] formulated the DNA fragment assembly problem as a special kind of the Traveling Salesman problem (TSP) that is generally referred to as an *asymmetric TSP*. They used ACS to solve the FAP and showed that the performance of the ACS algorithm and that of the nearest neighbour heuristic rule at solving single-contig problems are approximately the same. In contrast, the ant colony system algorithm outperforms the nearest neighbour heuristic search when multiple-contig problems are considered. However, it was reported that the multiple-contig problems with large number of fragments cannot always be solved using the ant colony system algorithm.

In [44], the authors present several methods - a canonical genetic algorithm, a CHC (Cross generational elitist selection, Heterogeneous recombination and Cataclysmic mutation) method, a scatter search algorithm, and a simulated annealing method, to solve accurately some problem instances that are 77K base pairs long. Since the work of [44], the problem of DNA fragment assembly has been tackled with different other heuristics and meta-heuristics in the literature [13, 27, 50]. We give a brief overview of the techniques proposed in [13, 27, 50] shortly. Nonetheless, the quest for new more accurate and faster techniques still continues.

A lot of tools have been invented to automate DNA sequencing. Among these tools, PHRAP [32], TIGR assembler [62], STROLL [23], CAP3 [33], Celera assembler [51] and EULER [55] may be cited. All of these tools focus on coping with different problems faced during fragment assembly.

PHRAP (PHRagment Assembly Program) [32, 7] is originally designed for and mostly used in the assembly of data from shotgun sequencing, but has been used in EST clustering, genotyping, and to identify sequence polymorphisms. In the assembly process it allows the use of entire reads; not just trimmed high quality parts of the sequences. To improve accuracy in the assembly process of PHRAP, in the presence of repeats, a combination of user supplied information and internal information is used, like clone name, the direction of the read, and the dye chemistry used to generate the read. PHRAP provides extensive information about the assembly to assist in trouble-shooting.

As mentioned earlier, PHRAP purely assembles complete reads, that is, it does not trim the reads prior to assembly, and problematic reads, such as vector contaminated reads, must be dealt with before assembly (e.g., by cross_match, a part of the phrap package). PHRAP uses read quality data (from phred, also part of the phrap package) and uses this to assign "quality" values (Log Likelihood Ratios (LLR)) to matches, it then assembles the matching reads into contigs by using a greedy algorithm based on the LLR values.

The CAP3 (Contig Assembly Program 3) [33, 7] clips $5'$ and $3'$ low quality end regions in reads. The overlap detection is performed by finding chains of ungapped segment identical alignment. The overlaps are processed with a banded Smith-Waterman algorithm and scored, taking quality values into account. Reads are then joined to form contigs in decreasing order of overlap scores, and forward-reverse constraints are used to make corrections to contigs. Finally a multiple sequence alignment of the reads is constructed, and a consensus sequence, along with a quality value for each base, is computed for each contig. Notably, In construction of the multiple alignment and consensus sequence the quality scores are used. CAP3 produces fewer errors than PHRAP.

TIGR Assembler [62] is a tool capable of using quality values and assembles data with greater care to repeat detection and contig-level overlapping.

While assembler programs like PHRAP, CAP3, TIGR and the Celera assembler are based on a heuristic algorithm and "overlap-layout-consensus" paradigm, the Euler assembler [55] uses a very different approach, namely, an *Eulerian Superpath* approach. It uses clone end sequencing by using double barreled (DB) data to perform the assembly. The result is an assembler capable of assembling repeat regions, which is complicated for other assemblers [7]. The Euler assembler creates a virtual *Sequencing By Hybridization (SBH)* problem, by breaking the reads into overlapping n-mers. A special graph called de Bruijn graph is build, in which each edge corresponds to an n-mer from one of the original sequence reads. The source and destination nodes corresponds respectively to the $n-1$ prefix and $n-1$ suffix of the corresponding n-mer. The original DNA sequence is reconstructed, by finding a path that uses all the edges exactly once i.e., an *Eulerian path*.

It is important to note that the classical DNA fragment assemblers stated above use fitness functions that favor solutions having strong overlaps between adjacent fragments in the layouts. But we also need to obtain an order of the fragments that minimizes the number of contigs, with the goal of reaching one single contig, i.e., a complete DNA sequence composed of all the overlapping fragments. PALS [13] (Problem Aware Local Search) is a simple, fast and accurate heuristic solution that is recently proposed with the objective of achieving this goal. In PALS, the number of contigs is used as a high-level criterion to judge the whole quality of the results since it is difficult to capture the dynamics of the problem into other mathematical functions. However, the calculation of the number of contigs is quite time-consuming, and this fact definitely precludes any algorithm to use such calculation. A solution to this problem was introduced in [13] as an utilization of a method that should not need to know the exact number of contigs and thus be computationally light. The key contribution of PALS [13] is to indirectly estimate the number of contigs by measuring the actual number of contigs that are created or destroyed when tentative solutions are manipulated. The authors

in [13] used a variation of a heuristic algorithm for TSP (Traveling Salesman Problem) for the DNA field, which does not only use the overlaps among the fragments, but also takes into account (in an intelligent manner) the number of contigs that has been created or destroyed.

In [27], the authors proposed a new method combining a general purpose meta-heuristic with a local search method specifically designed for this problem, namely, PALS of [13]. They presented a new approach of cellular genetic algorithms (cGA) that regulates the intensity on the search while solving a problem, outperforming the compared cGAs with static populations. This model also has the advantage that it requires almost no additional cost with respect to the canonical cGA, since it basically consists of changing the population shape (which has zero cost if the population is implemented as a list of individuals) in order to keep an appropriate balance between the intensification and diversification performed during the search.

While the sequential genetic algorithm has given good results, it is unable to sequence very large DNA molecules efficiently. As a result, several authors have proposed parallel and distributed versions of genetic algorithm to overcome this difficulty. In [14], the authors have presented two parallel methods,a distributed genetic algorithm and a parallel simulated annealing algorithm,to solve accurately problem instances that are 77K base pairs long. Another work [57] has proposed a new parallel asynchronous cellular genetic algorithm model for multi-core processors. The algorithm has been used for solving the DNA FAP with the aim to find highly accurate solutions in lesser computational effort. Also, several new local search methods have been designed, and their influence on the performance of the algorithm has been analyzed.

Another paper [42] proposed evolutionary based iterative optimization method called Prototype Optimization with Evolved Improvement Steps (POEMS) to solve the DNA fragment assembly problem. POEMS is an iterative algorithm that seeks for the best modification of the current solution

in each iteration. The modifications are evolved by means of an evolutionary algorithm.

All of the above mentioned previous works consider operations on noiseless data. Recently, in [50], performance of various meta-heuristic based algorithms were discussed where a uniform random error model was introduced in the fitness score matrix. Moreover, the authors have used constructive heuristic seeding strategies to generate initial solutions with an aim to exploit promising regions of the search space.

4.3 Summary

This chapter has discussed different techniques found in the literature to solve DNA FAP. In the next chapter, we discuss the error models that have been used to generate noisy datasets.

Chapter 5

Error Models

5.1 Introduction

In this chapter, we discuss about the error models that have been used to generate noisy datasets for experimentation. We first mention the errors introduced during Sanger Sequencing. Next, we explain the errors being generated due to fluctuation of light signal in 454 Sequencing. Lastly, we discuss about the errors due to misread orientation. These three models have been simulated in MetaSim to generate noisy datasets. No previous work take these error models into consideration, during the generation of datasets. We have taken the first ever approach to generate realistic datasets by incorporating error models in the fragmentation process of DNA instances.

5.2 Sanger Sequencing Error Model

In Sanger Sequencing, fragments are read by two machines: *automated gel sequencers* that use electrophoresis and *fluorescent markers*. These machines determine the sequence of the nucleotides in one fragment. The ability of these machines to read consecutive pieces of DNA degrades quickly with the length of the sequence, and today a sequencing machine can read up to ≈700 consecutive base pairs of a fragment of DNA, depending on the degree of accuracy desired.

So, the error profile of Sanger sequencing is modeled as follows [6]:

1. The probability of an error occurring at position i of a read increases linearly with i, and
2. If an error occurs at position i, then with some fixed probabilities, it is either a substitution, a deletion or an insertion.

In this error model, the following parameters concerning different probability values, P must be set because of the two assumptions mentioned above:

1. P (error at i) for positions $i = 1$ and $i = 1000$
2. P (deletion | error), deletion error rate.
3. P (insertion | error), insertion error rate.
4. P (substitution | error) = 1 - P (deletion | error) - P (insertion | error), substitution error rate.

5.3 454 Sequencing Error Model

As discussed previously, In pyro-sequencing, the intensity of emitted light signal is used to estimate the length of homopolymers. For chemical and technical reasons, this signal is subject to fluctuations that lead to sequencing errors. The term *noise flow values* (in literature sometimes referred to as *negative flow values*) is used to refer to the fact that the light signal is weak. An important factor in modeling error in generating fragments using 454 sequencing technique is to properly reflect the behaviour of negative flow.

Light intensities of negative flows follow a lognormal distribution, with mean $\mu = 0.23$, and standard deviation $\sigma = 0.15$. A random variable X is said to be lognormally distributed, if the random variable $\ln X$ is normally

distributed. Let μ and σ be the mean and standard deviation of X and let m and s be the mean and standard deviation of $\ln(X)$. Then,

$$m = \ln(\frac{\mu^2}{\sqrt{\sigma^2 + \mu^2}}) \tag{5.1}$$

and

$$s = \sqrt{\ln((\frac{\sigma}{\mu})^2 + 1)} \tag{5.2}$$

The probability density function of a lognormally distributed random variable is usually specified with mean m and standard deviation s of the underlying normally distributed $\ln(X)$. Based on this, the base calling intensities of negative flows are simulated and the misinterpretation of nullmers can be modeled as homopolymers of length one.

So far, we have discussed how to simulate the light intensities for negative flow. We also need to model the base calling process. In the base calling process, the intersections of the density functions of the normal distributions for different homopolymer lengths r_1 and r_2 are calculated and stored in an intersection matrix M. Then, these values are used to decide which homopolymer length is called.

5.4 Exact Error Models

The Exact Error Model is suitable for sampling reads without modifying any bases, i.e. in contrast to the other provided error models, no substitution, insertion or deletion is applied to the read sequences. But orientations may be changed during reading.

5.5 Summary

In this chapter, we have discussed the three error models that have been used to generate noisy datasets. In the next chapter, we present two of our

algorithms, based on meta-heuristics, for solving DNA Fragment Assembly problem.

Chapter 6

Meta-heuristics for the DNA Fragment Assembly Problem

6.1 Introduction

In this chapter we present an Artificial Bee Colony (ABC) algorithm and a Queen-bee Evaluation based on Genetic Algorithm (QEGA) for solving the DNA FAP. These two algorithms belong to a special class of techniques based on swarm intelligence. We start with a brief review of swarm intelligence techniques in the following subsection. Then we briefly review the ABC algorithm before presenting our approach.

6.2 Swarm Intelligence

Many existing meta-heuristic algorithms in the literature are nature-inspired. The behavior of a single ant, bee, termite and wasp often is too simple, but their collective and social behavior is of paramount significance. The collective and social behavior of living creatures motivated researchers to undertake the study of swarm intelligence (SI) [26]. SI systems are typically made up of a population of simple agents (an entity capable of performing/executing certain operations) interacting locally with one another and with the environ-

ment. Although there is normally no centralized control structure dictating how individual agents should behave, local interactions between such agents often lead to the emergence of global behavior. Many biological creatures such as fish schools and bird flocks clearly display structural order, with the behavior of the organisms so integrated that even though they may change shape and direction, they appear to move as a single coherent entity. The main properties of the collective behavior are homogeneity, locality, collision avoidance, velocity matching and flock centering [26].

A swarm can be viewed as a group of agents cooperating to attain some goal through achieving some purposeful behavior. This collective intelligence seems to emerge from what are often large groups. According to Milonas [49], five basic principles define the SI paradigm. First is the *proximity principle*: the swarm should be able to carry out simple space and time computations. Second is the *quality principle*: the swarm should be able to respond to quality factors in the environment. Third is the principle of *diverse response*: the swarm should not commit its activities along excessively narrow channels. Fourth is the *principle of stability*: the swarm should not change its mode of behavior every time the environment changes. Fifth is the *principle of adaptability*: the swarm must be able to change behavior when it is worth the computational price. The 4^{th} and the 5^{th} principles are the opposite sides of the same coin.

Two fundamental concepts, *self-organization* and *division of labour*, are necessary and sufficient properties to obtain swarm intelligent behaviour. *Self-organization* is one of the fundamental features of any SI system. In general, it refers to the various mechanisms by which pattern, structure and order emerge spontaneously in complex systems. Self-organization results in structures at the global level of a system by means of interactions among its low-level components. These mechanisms establish basic rules for the interactions between the components of the system. The rules ensure that the interactions are executed on the basis of purely local information without any relation to the global pattern. Bonabeau et al. have tried to define self-organization using the following words [19]:

Self-organization is a set of dynamical mechanisms whereby structures appear at the global level of a system from interactions of its lower-level components .

There are four main features that govern the self-organization in insect colonies:

1. **Positive feedback (amplification):** It is a simple behavioural rules of thumb that promotes the creation of convenient structures. Recruitment and reinforcement such as *trail laying and following* in some ant species or *dances* in bees can be cited as the examples of positive feedback.

2. **Negative feedback:** It counterbalances positive feedback and helps to stabilize the collective pattern. In order to avoid the saturation which might occur in terms of available foragers, food source exhaustion, crowding or competition at the food sources, a negative feedback mechanism is needed.

3. **Amplification of fluctuations:** Fluctuations like random walks, errors, random task switching among swarm individuals are vital for creativity and innovation. Randomness is often crucial for emergent structures since it enables the discovery of new solutions.

4. **Multiple interactions:** Self organization requires a minimal density of mutually tolerant individuals, enabling them to make use of the results from their own activities as well as others.

The agents use simple local rules to govern their actions and via the interactions of the entire group, the swarm achieves its objectives.

The second unique feature of SI system is *division of labour*: inside a swarm, there are different tasks, which are performed simultaneously by specialized individuals. Simultaneous task performance by cooperating specialized individuals is believed to be more efficient than the sequential task per-

formance by unspecialized individuals [19]. Division of labour also enables the swarm to respond to changed conditions in the search space.

6.3 Bee Colony Algorithms

The classical example of a swarm is bees swarming around their hive. A family of swarm algorithms, known as Bee Colony algorithms, tries to model the natural behaviour of real honey bees. Honey bees use several mechanisms like waggle dance that makes them a good candidate for developing new intelligent search algorithms.

6.3.1 Behaviour of a Honey Bee Swarm

The minimal model of forage (food) selection that leads to the emergence of collective intelligence of honey bee swarms consists of three essential components: food sources, employed foragers and unemployed foragers. The model defines two leading modes of the behaviour: the recruitment to a nectar source and the abandonment of a source [38]. A brief description of the components are given below.

1. **Food Sources:** The value of a food source depends on many factors, such as, its proximity to the nest, its richness or concentration of its energy, and the ease of extracting this energy. For the sake of simplicity, the "profitability" of a food source can be represented with a single quantity.

2. **Employed Foragers:** They are associated with a particular food source which they are currently exploiting or are "employed" at. They carry with them information about this particular source, its distance and direction from the nest, the profitability of the source and share these information with a certain probability.

3. **Unemployed Foragers:** They are continually at the look out for a food source to exploit. There are two types of unemployed foragers,

namely, scouts and onlookers. Scouts search the environment surrounding the nest for new food sources and onlookers wait in the nest and establish a food source through the information shared by employed foragers. The mean number of scouts of total bees is about $5 - 10\%$.

The exchange of information among bees is the most important occurrence in the formation of the collective knowledge. While examining the entire hive it is possible to distinguish between some parts that commonly exist in all hives. The most important part of the hive with respect to exchanging information is the dancing area. Communication among bees related to the quality of food sources takes place in the *dancing area*. This dance is called a *waggle dance*.

Since information about all the current rich sources are available to an onlooker on the dance floor, probably she can watch numerous dances and decides to employ herself at the most profitable source. Employed foragers share their information with a probability proportional to the profitability of the food source, and the sharing of this information through waggle dancing is longer in duration. So, there is a greater probability of onlookers choosing more profitable sources since more information is circulated about the more profitable sources. Hence, the recruitment is proportional to the profitability of the food source. In order to understand the basic behaviour characteristics of foragers better, let us adopt an example. Assume that there are two discovered food sources: A and B. At the very beginning, a potential forager will start as unemployed forager. That bee will have no knowledge about the food sources around the nest. There are two possible options for such a bee:

1. It can be a scout and starts searching around the nest spontaneously for a food due to some internal motivation or possible external clue.

2. After watching the waggle dances, it can be recruited to search for a food source.

After locating the food source, the bee utilizes its own capability to memorize the location and then immediately starts exploiting it. Hence, the bee

will become an "employed forager". The foraging bee takes a load of nectar from the source and returns to the hive and unloads the nectar to a food store. After unloading the food, the bee has the following three options:

1. It becomes an uncommitted follower after abandoning the food source.
2. It dances and then recruits nest mates before returning to the same food source.
3. It continues to forage at the food source without recruiting other bees.

It is important to note that not all bees start foraging simultaneously. The experiments confirmed that new bees begin foraging at a rate proportional to the difference between the eventual total number of bees and the number of current foragers [38].

In the case of honey bees, the basic properties on which self-organization relies are as follows:

1. *Positive feedback:* As the nectar amount of food sources increases, the number of onlookers visiting them increases, too.
2. *Negative feedback:* The exploitation process of poor food sources is stopped by bees.
3. *Fluctuations:* The scouts carry out a random search process for discovering new food sources.
4. *Multiple interactions:* Bees share their information about food sources with their nest mates on the dance area.

6.4 Overview of Bee Based Algorithms

The Bee Colony meta-heuristic belongs to the class of nature-inspired algorithms which are inspired by various biological and natural processes observed in honey bee swarm. Bee-inspired approaches in this narrower sense can be

roughly classified into three different main types [15]. The first group is inspired by the foraging behaviour of honey bee. The basic idea behind this approach is to create a colony of artificial bees able to efficiently solve hard combinatorial optimization problems. A number of algorithms in this regard appeared during the last decade, namely, Bee System (BS) [53], BeeHive [66], Bee Colony Optimization (BCO) [65], Artificial Bee Colony (ABC) [38], Bee Swarm Optimization (BSO) [28], Virtual Bee Algorithm (VBA) [68], Honey Bee Colony Algorithm (HBCA) [24], Bee Algorithm (BA) [56] etc.

The second group of bee-based algorithm is indirectly inspired by the mating behaviour of honey bees. Each honey bee colony consists of the queen, drones, workers, and broods. The marriage process starts with a dance performed by the queen who then starts a mating flight. During this flight the drones follow the queen and mate with her in the air. In each mating, sperm accumulates there to form the genetic pool of the colony. Each time a queen lays eggs, she retrieves at random a mixture of the sperms accumulated to fertilize the egg. Marriage Bee Optimization (MBO) [12] is the first search algorithm inspired by this behaviour. In the artificial analogue model, the mating flight can be visualized as a set of transitions in a state space where the queen moves between the different states in the space and mate with the drone encountered at each state probabilistically. The probability of mating is high when either the queen is still in the start of her mating flight and therefore her speed is high, or when the fitness of the drone is as good as the queen's one. The algorithm starts with initializing the queen's genotype at random. After that, a heuristic is used to improve the queen's genotype realized by workers. In [64] the marriage of honey bee is analysed as the continuation of the work presented in [12].

The third type of bee-based algorithm is studied in [37] and [16]. In [37], the authors presented a new search algorithm inspired by the queen bee evolution process and have used it to enhance the optimization capability of genetic algorithms. The queen-bee evolution makes it possible for genetic algorithms to quickly attain the global optimum as well as to decrease the probability of premature convergence. The authors of [16] have subsequently

modified the queen bee evolution technique of [37].

6.5 Artificial Bee Colony (ABC) Algorithm for the DNA FAP

In this section, we describe the Artificial Bee Colony (ABC) algorithm for solving the DNA FAP. In what follows, we will use ABC_FAP algorithm to refer to our ABC algorithm to solve DNA FAP. Recall that, in the ABC algorithm [39], the colony of artificial bees contains three groups of bees: employed bees, onlookers and scouts. A bee which waits at the dance area for making decision to choose a food source depending on waggle dance of employed bee, is called an onlooker and a bee going to the food source visited by itself previously is named an employed bee. A bee which performs random search for food is called a scout. In the ABC algorithm (Algorithm 1), first half of the colony consists of employed artificial bees and the second half constitutes the onlookers. For every food source around the hive, there is only one employed bee. An employed bee becomes a scout when its food source is exhausted.

6.5.1 Formulation of the Problem and Objective

For the DNA FAP, given a set of fragments, our target is to find the permutation of fragments that minimize the number of contigs and maximize the fitness. The solution achieving these two objectives best represents the DNA sequence instance.

6.5.2 Initialization

In the ABC_FAP algorithm, each of the food sources represents a permutation of fragments of a DNA sequence. A food source represents a possible solution and the nectar amount of a food source corresponds to the quality

Algorithm 1 Generic ABC Algorithm

1: Initialize potential food sources for employed bees.
2: **while** Requirements are not met **do**
3: Each employed bee goes to a food source in her memory and determines a neighbour source, then evaluates its nectar amount and dances in the hive
4: Each onlooker watches the dance of employed bees and chooses one of their sources depending on the dances, and then goes to that source. After choosing a neighbour around that, she evaluates its nectar amount.
5: Abandoned food sources are determined and are replaced with the new food sources discovered by scouts.
6: The best food source found so far is registered
7: **end while**

(fitness) of the associated solution. Initially we generate the food sources (i.e., permutations) randomly. We don't assume that the initial seeds are taken from the best of some previous executions of some other algorithm. This is more realistic assumption compared to [50] where they use seeding strategies to find good solutions beforehand to exploit promising regions, for both noisy and noiseless data. Our algorithm is robust in the sense that, although we do not make any such preprocessing, as will be reported in a later section, our algorithm converges to almost same fitness compared to that of [50] in reasonable time.

6.5.3 Iteration Using PALS

After the initialization, the food sources are repeatedly searched by employed bees, onlooker bees and scout bees. An employed or onlooker bee produces a modification of the solution for finding a new food source and tests the nectar amount (fitness value) of the new solution. For the modification, we have used problem aware local search (PALS) [13] on the permutation μ under

Algorithm 2 PALS [13]

1: $s \leftarrow GenerateInitialSolution()$
2: **while** there are changes **do**
3: $L \leftarrow \emptyset$
4: **for** $i = 0$ to N **do**
5: **for** $j = 1$ to N **do**
6: $\Delta_c, \Delta_f \leftarrow$ CalculateDelta(s, i, j)
7: **if** $\Delta_c \geq 0$ **then**
8: $L \leftarrow L \cup < i, j, \Delta_f, \Delta_c >$
9: **end if**
10: **end for**
11: **end for**
12: **if** $L <> \emptyset$ **then**
13: $< i, j, \Delta_f, \Delta_c > \leftarrow$ Select(L)
14: Applymovement(s, i, j)
15: **end if**
16: **end while**
17: **return** s

Algorithm 3 CalculateDelta Function

1: $\Delta_c \leftarrow 0$
2: $\Delta_f \leftarrow 0$
3: $\Delta_f = w_{s[i-1]s[j]} + w_{s[i]s[j+1]}$
4: $\Delta_f = \Delta_f - w_{s[i-1]s[i]} - w_{s[j]s[j+1]}$
5: **if** $w_{s[i-1]s[i]} > cutoff$ **then**
6: $\quad \Delta_c = \Delta_c + 1$
7: **end if**
8: **if** $w_{s[j]s[j+1]} > cutoff$ **then**
9: $\quad \Delta_c = \Delta_c + 1$
10: **end if**
11: **if** $w_{s[i-1]s[j]} > cutoff$ **then**
12: $\quad \Delta_c = \Delta_c - 1$
13: **end if**
14: **if** $w_{s[i]s[j+1]} > cutoff$ **then**
15: $\quad \Delta_c = \Delta_c - 1$
16: **end if**
17: return Δ_f, Δ_c

consideration (Algorithm 2). We have done so because, besides considering fitness value, we also take the number of contigs of a solution into consideration (Algorithm 3).

The calculation of the number of contigs is quite a time-consuming operation. A solution to this problem is the utilization of the method which should not need to know the exact number of contigs and thus be computationally light. PALS [13] indirectly estimate the number of contigs by measuring the number of contigs that are created or destroyed when tentative solutions are manipulated.

PALS (Algorithm 2) works on a single solution (an integer permutation encoding a sequence of fragment numbers, where consecutive fragments overlap) which is generated using the *GenerateInitialSolution* method (see *Algorithm 2*), and it is iteratively modified by the application of *movements* in a structured manner. A *movement* is a perturbation (*ApplyMovement* method of Algorithm 2) that, given a solution s, and two positions i and j, reverses the subpermutation between the positions i and j.

The key step in PALS is the calculation of the variation in the overlap (Δ_f) and in the number of contigs (Δ_c) among the current solution and the resulting solution after applying a movement (see Algorithm 3). This calculation is computationally light since we do not calculate either the fitness function or the number of contigs, but instead we estimate the *variation* of these values. To do this, we only need to analyze the affected fragments by the tentative movement (i, j, $i-1$ and $j+1$), removing the overlap score of the affected fragments of the current solution and adding the one of the modified solution to Δ_f (equations of lines 3–4 of Algorithm 3) and checking whether some current contig is broken (first two *if* statements of Algorithm 3) or two contigs are merged (last two *if* statements of Algorithm 3) by the movement operator.

In each iteration, PALS makes these calculations for all possible movements, storing the candidate movements in a list L. It only considers can-

didates to be applied the movements which reduce the number of contigs ($\Delta_c \leq 0$). Once it has completed the previous calculations, the method selects a movement of the list L and applies it. We select the best movement, i.e., we choose the movement having the lowest Δ_c (thus the movement maintains or reduces the number of contigs). In case that several movements have the same Δ_c, the applied movement will be the one with a higher Δ_f value (it increases the overlap among the fragments).

The algorithm stops when no more candidate movements are generated. The cutoff value and its significance in Algorithm 3 will be discussed in Chapter 8.

6.5.4 Fitness Calculation

Let us denote the set of fragments by an array f, so that the i^{th} fragment in the set is denoted by $f[i]$. Also let us denote the current solution under consideration by μ. In all cases, we have used a fitness function (calculation of the amount of nectar) that sums the overlap score for adjacent fragments ($f[i]$ and $f[i+1]$) in a given solution. Let us denote the overlap score by $w(f[i], f[i+1])$ and fitness function of μ by F_μ. So, we have,

$$F_\mu = \sum_{i=0}^{n-2} w(f[i], f[i+1]), \tag{6.1}$$

where n is the number of fragments in the solution.

6.5.5 Probability Calculation for Food Source

If the nectar amount of the new modified source is higher than that of the previous one, the bee memorizes the new solution and forgets the old one. Otherwise it keeps the previous food source. Onlooker bees choose the food source for modification probabilistically, depending on the nectar amount of the food source. The probability, p_μ is calculated as follows:

$$p_\mu = \frac{F_\mu}{global_{max}}, \tag{6.2}$$

where $global_{max}$ denotes the maximum fitness value found so far among all food sources. If the nectar amount of a food source increases, the probability with which that food source is chosen by an onlooker increases, too. Recall that, the dance of employed bees carrying higher nectar has higher probability of being selected by onlookers for exploitation.

In Summary, ABC_FAP algorithm, in conjunction with PALS, tries to select the best fitness-valued solution with the goal of achieving minimum number of contig.

6.6 Queen-bee Evaluation Based on Genetic Algorithm (QEGA) for DNA FAP

Conventional genetic algorithm sometimes become unsuccessful to find a globally optimal solution within limited number of evolutions. To overcome this disadvantage, we have employed a queen-bee evolution based on genetic algorithm (QEGA) ([37], [58]) for solving the DNA FAP. In what follows, we will use QEGA_FAP algorithm to refer to our QEGA algorithm to solve DNA FAP.

The population of QEGA consists of several permutations of fragments of a DNA sequence. To determine the fitness value of each individual in the population, we have used Equation 6.1 as the fitness calculating function in Steps 2 and 16 of QEGA_FAP (Algorithm 4).

There are two major differences between conventional genetic algorithm (CGA) and QEGA_FAP. Firstly, the parents $p(t)$ in CGA are composed of the k individuals selected by a selection algorithm such as tournament selection. On the other hand, parents $p(t)$ in QEGA_FAP consist of the $\frac{k}{2}$ copies of a queen-bee $I_q(t-1)$, where $q = arg\ max\ \{F_\mu, 1 \leq \mu \leq k\}$ and $\frac{k}{2}$ copies of

Algorithm 4 QEGA Algorithm

Input: time t, population size k, populations P, normal mutation rate σ, normal mutation probability p_m, strong mutation probability p'_m, a queen-bee I_q, selected bees I_m

Output: best fitness solution

```
1: Initialize P(t)
2: Evaluate P(t)
3: while Condition not met do
4:    t ← t + 1
5:    Select P(t) from P(t − 1)
6:    P(t) = {I_q(t − 1), I_m(t − 1)
7:    Recombine P(t)
8:    Crossover
9:    for i = 0 to k do
10:      if i ≤ (σ ∗ k) then
11:         Mutate with p_m
12:      else
13:         Mutate with p'_m
14:      end if
15:   end for
16:   Evaluate P(t)
17: end while
```

bees $I_m(t-1)$ selected by a selection algorithm, where $1 \leq m \leq \frac{k}{2}$. Secondly, all individuals in conventional genetic algorithm are mutated with a small mutation probability p_m, while in QEGA_FAP only a part of the individuals are mutated with normal mutation probability p_m and the others are mutated with strong mutation probability p'_m. The ratio between p_m and p'_m is denoted by σ in QEGA. Generally, p_m is less than 0.1 and p'_m is greater than p_m.

So, In QEGA_FAP, the fittest individual in a generation, crossbreeds with the bees selected as parents by means of a selection algorithm. This feature of queen-bee evolution reinforces the exploitation of genetic algorithms. That is, fitness of the offsprings mainly rely on the crossover operation and the fittest individual. Consequently, it also increases the probability of premature convergence. Nonetheless, the second feature of QEGA_FAP helps genetic algorithms search new space, i.e. it increases the exploration of genetic algorithms through strong mutation. These two features enable genetic algorithms to evolve quickly and simultaneously maintain good solutions. In other words, the queen-bee evolution makes it possible for genetic algorithms to quickly approach the global optimum as well as decreasing the probability of premature convergence.

Ordered two-point crossover (Figure 6.1) is used as crossover operator in Step 8 of Algorithm 4. In this scheme, given two parents, two random crossover points are selected partitioning them into a left, middle and right portion. Then ordered crossover is carried out in the following way: Child 1 inherits its left and right section from Parent 1, and its middle section is determined by the fragments in the middle section of Parent 1 in the order in which the fragments appear in Parent 2. A similar process is applied to determine Child 2. Notably, the Ordered crossover operator is a pure recombination operator [63]. For strong mutation, we have used problem aware local search (PALS) [13]. For normal mutation, we have done random mutation on the offspring.

Parent 1 : 4 2 | 1 3 | 6 5 Child 1 : 4 2 | 3 1 | 6 5
Parent 2 : 2 3 | 1 4 | 5 6 Child 2 : 2 3 | 4 1 | 5 6

Figure 6.1: Example of ordered crossover [2].

6.7 Summary

Swarm intelligence based meta-heuristic techniques have been found instrumental to solve different real-life problems. In this chapter we discuss Artificial Bee Colony (ABC) algorithm and Queen-bee Evaluation based on Genetic Algorithm (QEGA) for solving the DNA FAP. In the next chapter we present two hybrid meta-heuristics based on genetic algorithms to solve the problem.

Chapter 7

Hybrid Meta-heuristics for the DNA Fragment Assembly Problem

7.1 Introduction

Recently, hybrid meta-heuristics have gained much interest of the researchers. Hybrid meta-heuristic can be broadly defined as integration of a meta-heuristic related concept with some other techniques (possibly another meta-heuristic) [18]. We may distinguish between two categories: the first consists in designing a solver including components from a meta-heuristic into another one, while the second combines meta-heuristics with other techniques typical of fields such as operations research and artificial intelligence. A prominent representant of the first category is the use of trajectory methods into population based techniques or the use of a specific local search method into a more general trajectory method such as Iterated Local Search (ILS). The second category includes hybrids resulting from the combination of meta-heuristics with constraint programming (CP), integer programming (IP), tree-based search methods, data mining techniques, etc.

We have used genetic algorithm with Hill climbing (GA+HC) and genetic algorithm with Simulated annealing (GA+SA) for solving the DNA fragment assembly problem. Before we proceed with our hybrid algorithms, we give an overview of Genetic algorithm, Simulated annealing and Hill climbing individually. Then we state our hybrid algorithms.

7.2 Genetic algorithm (GA)

In a genetic algorithm (Algorithm 5), a population of strings (called chromosomes or the genotype of the genome), which encode candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem, evolves toward better solutions [43]. Traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible. The evolution usually starts from a population of randomly generated individuals and happens in generations. In each generation, the fitness of every individual in the population is evaluated, multiple individuals are stochastically selected from the current population (based on their fitness), and modified (recombined and possibly randomly mutated) to form a new population. The modification step is used to generate a second generation population of solutions from those selected through genetic operators: crossover (also called recombination), and/or mutation. The new population is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached.

7.3 Hill Climbing

This technique (Algorithm 6) is related to gradient ascent, but it doesn't require us to know the strength of the gradient or even its direction: we just iteratively test new candidate solutions in the region of the current candidate, and adopt the new ones if they are better. This enables us to climb up

Algorithm 5 The Genetic Algorithm

```
1: popsize←desired population size
2: P ← {}
3: for popsize times do
4:    P ← P ∪ {new random individual}
5: end for
6: Best← □
7: while Best is not the ideal solution or we have more time do
8:    for each individual P_i ∈ P do
9:       AssessFitness(P_i)
10:      if Best=□ or Fitness (P_i) > Fitness (Best) then
11:         Best← P_i
12:      end if
13:   end for
14:   Q ← {}
15:   for popsize/2 times do
16:      Parent P_a ←SelectWithReplacement(P)
17:      Parent P_b ←SelectWithReplacement(P)
18:      Children C_a, C_b ← Crossover(Copy(P_a),Copy(P_b))
19:      Q ← Q ∪ {Mutate(C_a),Mutate(C_b)}
20:   end for
21:   P ← Q
22: end while
23: return Best
```

the hill until we reach a local optima [43].

Algorithm 6 The Hill climbing Algorithm

1: $S \leftarrow$Some initial candidate solution
2: **while** (S is not the ideal solution or we have more time) **do**
3: $R \leftarrow$Tweak(Copy(S))
4: **if** (Quality (R) $>$ Quality(S)) **then**
5: $S \leftarrow R$
6: **end if**
7: **end while**
8: **return** S.

The Tweak operation in Step 3 of Algorithm 6 is the main factor in achieving either exploitation or exploration. If we tweak a small amount, then Hill-Climbing will march right up a local hill and be unable to make the jump to the next hill because the bound is too small for it to jump that far. Once it is on the top of a hill, everywhere it jumps will be worse than where it is presently, so it stays put. Further, the rate at which it climbs the hill will be bounded by its small size. On the other hand, if the tweaking is large, then Hill-Climbing will bounce around a lot. Importantly, when it is near the top of a hill, it will have a difficult time converging to the peak, as most of its moves will be so large as to overshoot the peak. Thus small sizes of the bound move slowly and get caught in local optima; and large sizes on the bound bounce around too frenetically and cannot converge rapidly to finesse the very top of peaks. Optimization algorithms which make largely local improvements are exploiting the local gradient, and algorithms which mostly wander about randomly are thought to explore the space.

7.4 Simulated Annealing

Simulated Annealing (Algorithm 7) varies from Hill-Climbing in its decision of when to replace S, the original candidate solution, with R, its newly tweaked child [43]. Specifically, if R is better than S, we always replace S

with R as usual. But if R is worse than S, we may still replace S with R with a certain probability $P(t, R, S)$ as follows.

$$P(t, R, S) = \exp^{\frac{Quality(R)-Quality(S)}{t}} \tag{7.1}$$

That is, the algorithm sometimes goes down hills. If R is not much worse than S, we still select R with a reasonable probability. Here, we have a tunable parameter t. If t is close to 0, the probability is close to 0. If t is high, the probability is close to 1. The idea is to initially set t to a high number, which causes the algorithm to move to every newly-created solution regardless of how good it is, i.e., we are doing a random walk in the space. Then, t decreases slowly, eventually to 0, at which point the algorithm is doing nothing more than a plain Hill-Climbing.

Algorithm 7 The Simulated Annealing Algorithm

```
1: t ←temperature, initially a high number
2: S ←Some initial candidate solution
3: Best← S
4: while (Best is not the ideal solution or we have more time) do
5:    R ←Tweak(Copy(S))
6:    if (Quality (R) > Quality(S) or if a random number chosen from 0 to
      1 < exp^((Quality(R)−Quality(S))/t)) then
7:       S ← R
8:    end if
9:    Decrease t
10:   if (Quality (S) > Quality(Best)) then
11:      Best ← S
12:   end if
13: end while
14: return Best
```

Figure 7.1: Example of swap mutation.

7.5 Hybrid Algorithms for the DNA FAP

Let us now discuss genetic algorithm with Hill climbing (GA+HC) and genetic algorthm with Simulated annealing (GA+SA) that we have designed for solving the DNA fragment assembly problem. These two hybrid algorithms are implemented in ParadisEO [22] software framework. We have also implemented pure Genetic algorithm (GA) for noisy instances with the objective of comparing the performance of these three algorithms.

We have implemented our hybrid algorithms by incorporating both hill-climbing algorithm and simulated annealing algorithm separately inside the Genetic algorithm, after applying the mutation operator in Step 19 of Algorithm 5. We have used *Shift neighborhood technique* [40] to generate neighbors. In this technique, a shift operator s_{ij} is applied to a permutation $\pi = \{\pi_1, \ldots \pi_n\}$ such that it moves the element in Position i of π to Position j. Furthermore if $i < j$ $(i > j)$, the elements in position $i + 1, \ldots, j$ $(j, \ldots, i-1)$ are shifted to one position to the left (right). The corresponding neighbourhood is called the *shift neighbourhood*. Here we have used Ordered crossover as crossover operator, similar to QEGA_FAP. For each case we have used tournament selection of size 2 and pure generational replacement. *Tournament selection* (Algorithm 8) acts as the primary selection technique of an individual from current population. In *pure generational replacement*, no member of one population is allowed to pass to the next population. In mutation step, we used a basic technique called *swap mutation*. In swap mutation, two positions are selected at random and their contents are being exchanged (See Figure 7.1).

The motivation of using simulated annealing and hill climbing inside a

Algorithm 8 Tournament selection Algorithm

```
P ←population
t ← tournament size, t ≥ 1
Best ← Individual picked at random from P with replacement.
for i = 2 to t do
    Next ← Individual picked at random from P with replacement.
    if Fitness(Next) > Fitness (Best) then
        Best ← Next
    end if
end for
return Best
```

genetic algorithm is to compensate for the elitist selection [43]. We anticipate that a global algorithm equipped with a local exploration inside can be beneficial with respect to accidental errors introduced in the fragments during fragmentation phase. Taking into consideration that various insertions, deletions and substitutions of base pairs took place during fragment generation, we can't discard their effects and analyze our algorithm's efficiency in an utopian environment. To the best of our knowledge, none of the algorithms in the literature takes this important factor into account. Our main goal was to get a clear picture of and useful insight on how meta-heuristics perform on noisy data to solve the FAP. Additionally, we evaluate the performance of our algorithms for noiseless cases as well.

7.6 Summary

In this chapter, we have presented Genetic Algorithm with Hill Climbing (GA+HC) and Genetic Algorithm with Simulated Annealing (GA+SA) for solving the DNA FAP. In the next chapter, we present the experimental results obtained by Artificial Bee Colony (ABC) algorithm, Queen-bee Evaluation based on Genetic Algorithm (QEGA), Genetic Algorithm with Hill climbing (GA+HC) and Genetic Algorithm with Simulated annealing (GA+SA) for solving the DNA FAP.

Chapter 8

Experimental Results

8.1 Introduction

In this chapter, we present our experimental setup and the results obtained by our algorithms. The experiments were conducted in Ubuntu 11.04 running on an Intel core i3 Processor with 2GB RAM. We have designed two frameworks for evaluating fitness of the resulting solution. One framework solely used consecutive fragment overlapping as the metric for fitness assessment. The other framework used as fitness criteria: the number of final contig achieved in conjunction with fragment overlap. Both of the approaches are prevalent in the literature [46, 44]. The framework that uses only fragment overlapping as fitness assessment criteria does not use contig calculation because if we want to do so, we need exhaustive search for getting the best fragment suitable for each of the iteration. This is not a feasible solution if we take time into consideration. Moreover, contig calculation only ensure a continuous sequence in terms of the threshold or cutoff defined previously. For noisy instances, specially for longer datasets, ensuring single contig may not be pragmatic because of the errors. If one wants a continuous sequence, he can easily calculate the final contig number of the best fitness candidate and lower the cutoff value according to his observation. On the other hand, the framework that use both fragment overlapping and contig number as the fitness measure, can do so in reasonable time because of the use of Problem Aware Local Search (PALS) inside them. PALS is designed such that it

only take into consideration the increment or decrement of total number of contigs after applying a movement to current solution. In particular, instead of exploring the whole search space, PALS enables us to only consider the difference in the number of contigs, thus minimizing the exploration time.

8.2 Experimental Setup

8.2.1 Datasets

We have used GenFrag [29] and MetaSim [59] for generating noiseless and noisy artificial fragments respectively.

Target Problem Instances

We collected the DNA sequences from NCBI [8, 9, 10, 11]. We have chosen four sequences from the NCBI website as used in [50]: a human MHC class II region DNA with fibronectin type II repeats HUMMHCFIB, with accession number X60189; a human apolopoprotein HUMAPOBF, with accession number M15421; the complete genome of bacterio-phage lambda, with accession number J02459; a sequence of Neurospora crassa BAC, with accession number BX842596 (GI38524243). Besides we have selected other sequences from the NCBI web site similar to [50] and they correspond to a human microbion bacterium ATCC 49176 with accession numbers from ACIN02000001 to ACIN02000026. Particularly, we have used the longer sequences from this genome. We give a summary on the different features of the datasets in Table 8.1.

Noiseless input data generation

We have used GenFrag [29] for generating noiseless artificial fragments. GenFrag takes a known DNA sequence and uses it as a parent strand from which random fragments are generated according to the criteria supplied by the user (mean fragment length and coverage of parent sequence).

Table 8.1: Information of datasets. Accession numbers are used as the name of the instances

Instances	Coverage	Mean fragment length	Number of fragments	Original sequence length (in bps)
acin1	26	182	307	2170
acin2	3	1002	451	147200
acin3	3	1001	601	200741
acin5	2	1003	751	329958
acin7	2	1003	901	426840
acin9	7	1003	1049	156305
x60189_4	4	395	39	3835
x60189_5	5	386	48	
x60189_6	6	387	68	
m15421_5	5	398	127	10089
m15421_7	7	383	177	
j02459_7	7	700	352	48502
BX842596_4	4	708	442	77292
BX842596_7	5	703	773	

Noisy input data generation

We have used MetaSim [59] for noisy input data generation. We first discuss the motivation behind using Metasim as follows. We have consulted different DNA sequence simulators like FASIM [34], GenFrag [29] and Celsim [52]. In [34], the authors have showed various problems associated with Celsim and GenFrag version 1.0 and 2.1. Mainly, these programs are designed on the assumption that fragments are equally distributed on genome. That is, clone sampling is uniformly carried out during fragment generation. Thus, these programs do not sufficiently reflect actual conditions of WGSS (Whole Genome Shotgun Sequencing). For FASIM, current release of the software is only available upon request. On the other hand, MetaSim is a freely available sequencing simulator for genomic and metagenomics. It can be utilized to simulate fragments of real read experiments by incorporating errors which occur at the Layout phase, i.e., unknown orientation, base call errors, incomplete coverage, repeated regions, chimeras and contamination. Here we have used three error models, namely, 454, Sanger and exact error models considering configuration of all parameters. These configurations are presented in Table 8.2.

8.2.2 Score Matrix Calculation

As the exact orientation of the generated fragments are not predictable, to tackle the problem of unknown orientation, we have checked fragment overlapping in both forward and backward orientation during calculation of the score matrix. For example, let us assume that we want to calculate overlap between Fragment 1 and Fragment 2. We first calculate normal overlap between these two segments (suffix-prefix) and then compare the obtained value with the overlap value calculated from the reverse direction of Fragment 2 (suffix-suffix). We take the best value from the above two options. The score matrix (for both noisy and noiseless instances) is a matrix that is populated with the overlap value of each pair of fragments, calculated in the way stated above. The matrix is symmetric with respect to the diagonal. Each of the row (column) of the matrix designate one fragment and the overlap score of this fragment with each of the rest of the fragments. We use two different

Table 8.2: Configuration of MetaSim for different error models and no. of generations used by the algorithms

Parameter Considered	Error Model	Instances								
		acin1	acin2	acin3	acin5	acin7	acin9	x60189_6	m15421_7	j02459_7
Number of reads or Mate pair	454, Sanger, Exact	307	451	601	751	901	1049	68	177	352
DNA clone size distribution type [a]	-do-	Normal								
Mean	-do-	190	1020	1001	1003	1003	1003	387	387	700
Second Parameter	-do-	10	30	100						
Mate pair Probability	-do-	0								
Expected read length	454	190	1020	1007	1007	1005	1005	388	388	700
Mate pair read length	-do-	20								
Number of flow cycles	-do-	75	400	395	395	394	394	152	152	275
Mean negative flow cycles[b]	-do-	0.23								
Std. deviation for negative flow cycle[c]	-do-	0.15								
Signal std. deviation multiplier	-do-	0.15								
Read length distribution type	Sanger	Normal								
Error rate at read start	-do-	0.01								
Error rate at end of read	-do-	0.02								
Insertion error rate	-do-	0.2								
Deletion error rate	-do-	0.2								
Insertion	454	1330	10460	13651	17146	20882	23872	604	1571	5694
	Sanger	146	1433	1643	2147	2480	2860	71	157	666
Deletion	454	369	2758	3687	4438	5396	6373	125	388	1469
	Sanger	162	1370	1679	2131	2467	2965	70	165	621
Substitutions	Sanger	532	4095	5094	6409	7642	8969	216	537	1988
No. of base pair processed	454	57857	458964	594676	738514	889923	1034189	23719	63688	233127
	Sanger	56414	452401	567768	710617	852423	997708	23145	61276	223168
	Exact	55915	461288	596036	755338	903193	1050247	23719	69562	243163
No. of base pair generated	454	58818	466666	604640	751222	905409	1051688	24198	64871	237352
	Sanger	56398	452464	567732	710633	852436	997603	23146	61268	223213
	Exact	55915	461288	596036	755338	903193	1050247	24198	69562	243163

[a] A Clone in MetaSim is a DNA fragment that is randomly extracted from the source genome sequence for read/mate-pair sampling.

[b] A negative flow is a flow of nucleotides in which the sequence to synthesize is not elongated. Light intensities of negative flows follow a lognormal distribution. The default value ($\mu = 0.23$) is taken from [30].

[c] The default value (0.15) is taken from [30].

score matrices for each of the instances, based on noisy or noiseless datasets.

8.2.3 ABC Control Parameters

ABC algorithm has a few control parameters. We set Maximum number of cycles (MCN), i.e., maximum number of generation to 4000 and the colony size, i.e., population size to 256. The percentage of onlooker bees was set to 50% of the colony, the employed bees were 50% of the colony and the number of scout bees was selected as one. Notably, an increase in the number of scouts encourages the exploration whereas that in onlookers of a food source increases the exploitation.

8.2.4 GA+SA and GA+HC Implementation Details in ParadisEO

We have used templates of the paradisEO [22] software framework for implementing our Genetic algorithm with Simulated Annealing (GA+SA) and Genetic algorithm with Hill climbing (GA+HC) algorithms. ParadisEO is based on EO (Evolving Objects), a template-based ANSI-C++ compliant evolutionary computation library. EO mainly contains *functors*, that are objects which have a method called operator(). Such objects are used as if they were functions, but the big differences are:

- functors are functions with private data.
- Users can have different functors objects of the same class, i.e. users can use at the same time the same functionality with different parameters.
- Users can have a hierarchy of functors objects, which means that users have a hierarchy of functions with defaults behaviors and specialized sub-functions.

For conciseness, we very briefly outline the implementation details of GA+SA and GA+HC in ParadisEO here. In ParadisEO, representation of integer fragments for FAP has been done as follows. We use *eoInt* template to

represent vector of integer fragments and *eoInitPerc* template for generating the permutation of fragments consisting each individual. Ordered crossover and swap mutation with probability 0.3 and 0.7 respectively have been used for genetic algorithm. For this, we use *eoSwapMutation* and *eoOrderXover*. These values have been obtained from some preliminary results. To define our fitness function based on overlapping, we have to define a class inherited from the *eoEvalFunc<EOT>*. As mentioned previously, EO uses a functor style: the fitness function is computed in the method *operator()(EOT& _sol)*. We compute the fitness value in this method, and put the fitness value in the solution at the end by using its method *fitness*. For Simulated Annealing, we use *moSimpleCoolingSchedule* for defining probability to accept an inferior solution.

8.3 Results Obtained for Noiseless Data

Now, we present the results obtained by our algorithms : ABC_FAP, QEGA_FAP, GA, GA+HC and GA+SA for solving different noiseless DNA instances. There are two criteria found in the literature to assess the fitness value of each individual. One of the most common criteria is based on assessing the fitness value by considering the sum of the overlap score of each individuals by Equation 6.1. We measure the quality of the solution by considering this fitness criteria for all of the algorithms mentioned above. The second criteria for fitness assessment is to also consider the number of contigs in an individual. With the help of PALS, in QEGA_FAP and ABC_FAP algorithms, we also take into consideration the number of contigs in the final sequence obtained. The significance of contig calculation is to ensure that the solution best represents a continuous assembled sequence.

8.3.1 Results Obtained by ABC_FAP and QEGA_FAP for Noiseless Instances (Fitness Criteria: Overlap and No. of Contigs)

For noiseless instances, we set a cutoff value, i.e., required overlap between two adjacent fragments, to thirty [13] in *CalculateDelta* function of PALS (see Subsection 6.5.3) used in QEGA_FAP and ABC_FAP algorithms. This value provides one filter for spurious overlaps introduced by experimental errors [13]. Table 8.3 shows the results obtained for noiseless data by QEGA_FAP and ABC_FAP algorithms as well as by simulated annealing (SA) ([50]), Problem aware local search (PALS) ([50]) and pure Genetic algorithm (GA) ([50]). In this case we have found that our implementation of ABC_FAP and QEGA_FAP perform very competitively with the best algorithm in the literature which is based on simulated annealing (SA) for noiseless instances [50]. Similar behavior is observed in comparison to PALS [13], a defacto standard to compare genome assemblers, in all cases. These experiments validate our algorithms' good performance. Moreover, QEGA_FAP algorithm performs better than ABC algorithm for most of the instances. Notably, QEGA_FAP produces better results for longer instances. In terms of number of contigs, QEGA_FAP does a better job at contig reduction than the ABC_FAP algorithm. But both of the algorithms perform worse than SA and PALS in achieving reduced contig number, especially in case of larger instances like the *acinx* (where x= 1,2,3,5,7,9) instances. This is because, for larger instances, the cutoff value of thirty is much higher to ensure minimum overlap. As fragments are generated from a single DNA sequence randomly during dataset creation process, there is no guarantee that the consecutive fragments are overlapped by a threshold value of thirty. This issue become prominent when fragmenting larger instances as the coverage value is fixed at the inception of fragmenting process and the probability of ensuring threshold value of thirty decreases with the longer instances.

Table 8.3: Best final contig number and fitness for noiseless data

Instances	Best Fitness					Best Contig				
	ABC	QEGA	SA	PALS	GA	ABC	QEGA	SA	PALS	GA
x60189_ 4	11478	11476	11478	11204	11478	1	1	1	1	1
x60189_ 5	14016	14027	14027	12898	13502	1	1	1	1	1
x60189_ 6	18339	18266	18301	16992	17688	1	1	1	1	1
x60189_ 7	21184	21208	21271	20424	20884	1	1	1	1	1
m15421_ 5	38423	38578	38583	36540	37714	3	1	1	1	1
m15421_ 6	47515	47882	48048	45773	46949	2	1	1	1	2
m15421_ 7	54607	55020	55048	51454	52695	2	1	1	1	2
j02459_ 7	114774	116222	116257	108744	111103	3	1	1	1	1
bx842596_ 4	225431	227252	226538	211792	220558	13	8	1	1	6
bx842596_ 7	440187	443600	436739	412374	416414	12	3	1	1	3
acin1	45403	47115	46955	44656	45565	8	4	1	1	5
acin2	142539	144133	144705	138423	143444	237	233	1	1	236
acin3	155413	156138	156630	151928	154947	362	358	1	1	358
acin5	144339	144541	146607	143835	145332	556	552	1	1	552
acin7	155182	155322	157984	155089	155873	726	722	1	1	722
acin9	321397	322768	324559	310235	313203	552	552	1	1	552

8.3.2 Results Obtained by GA+SA and GA+HC for Noiseless Instances (Fitness Criteria: Overlap)

We also executed Genetic algorithm with Simulated annealing (GA+SA) and Genetic algorithm with Hill Climbing (GA+HC) for noiseless cases and compared the results with PALS in Table 8.4. As mentioned earlier, PALS is the defacto standard to compare DNA FAP algorithms. We have executed 100 iterations of GA+SA for each instance and 30000 iterations of GA+HC for each instance. The results obtained in 30000 iterations of GA+HC is comparable to the results obtained by 100 iterations of GA+SA. In GA+HC, hillclimbing is used as a mutation operator. In GA+SA, a cooling scheme is employed as an operator to generate large local candidates within fewer generations so that we can achieve equal results as the previous one in smaller steps. We present the best fitness obtained in each case. We have executed our algorithms for longer instances like *acinx*, as these instances are more challenging to assemble. As the results show, genetic algorithm with hill climbing (GA+HC) outperforms genetic algorithm with simulated annealing (GA+SA). GA+HC also performs better than PALS [13] for all instances except *acin9*.

We also try to demonstrate the behaviour of the fitness improvement over increment of number of iterations of GA+HC and GA+SA. Table 8.5 and 8.6 presents the results obtained by executing different iterations of GA+HC and GA+SA for *acin2* instance respectively. We have conducted experiments on other instances as well but only chose to present the results for *acin2* for conciseness. Similar results are obtained for other instances.By looking at Tables 8.5 and 8.6 in conjunction with Figures 8.1 and 8.2, we can see that the fitness value improves significantly with the increment of number of iterations. This suggest that, to get higher accuracy, we need to increase the number of iterations of GA+HC or GA+SA, if time permits.

Table 8.4: Best fitness obtained by GA+HC, GA+SA and PALS for noiseless data

Instances	Best fitness obtained		
	GA+SA	GA+HC	PALS
acin1	44747	46259	44656
acin2	138464	144544	138423
acin3	151202	154761	151928
acin5	144194	146031	143835
acin7	156480	156480	155089
acin9	273011	273011	310235

Table 8.5: Fitnesses obtained for acin2 by GA+HC for noiseless data

No of iterations	Fitness obtained
10000	131047
15000	133807
20000	137525
25000	144541
30000	144544
45000	144553

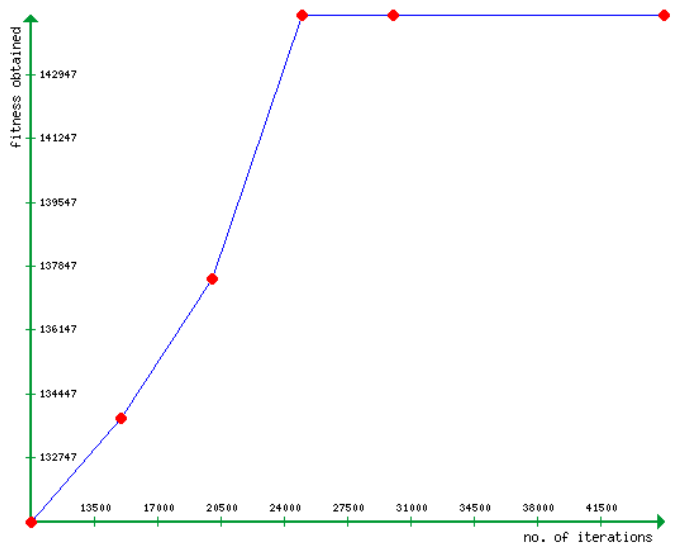


Figure 8.1: Relation between no. of iterations and fitness for acin2 by GA+HC for noiseless data.

Table 8.6: Fitnesses obtained for acin2 by GA+SA for noiseless data

No of iterations	Fitness obtained
40	129407
50	132733
60	134477
70	135870
80	136889
90	137922
100	138464
110	138912
120	139911

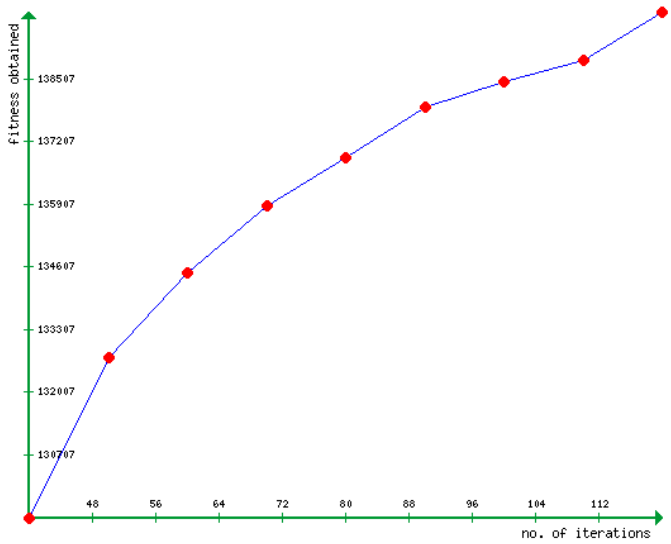


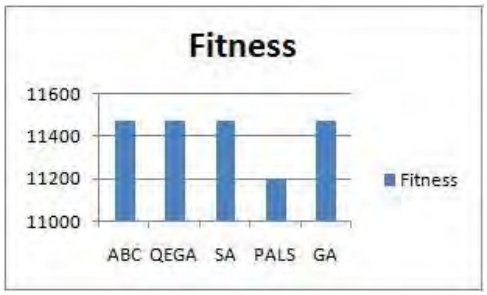
Figure 8.2: Relation between no. of iterations and fitness for acin2 by GA+SA for noiseless data.

8.4 Barchart Representation of the Results Obtained for Noiseless Instances

Figures 8.3, 8.4 and 8.5 represent the fitness values obtained by different algorithms for noiseless datasets, with the help of barcharts. It is useful to compare visually which algorithm perform better than others.

8.5 Results Obtained For Noisy Data

Next, we execute ABC_FAP, QEGA_FAP, GA, GA+SA, GA+HC for noisy instances and present the results.

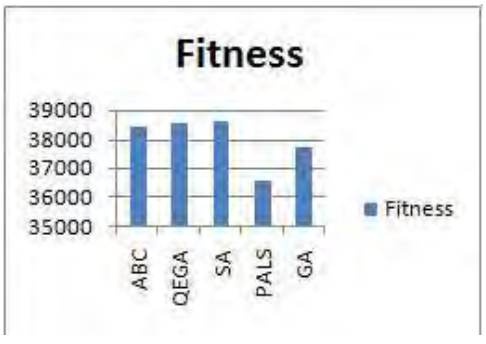


(a) x60189_4

(b) x60189_5

(c) x60189_6

(d) x60189_7



(e) m15421_5

(f) m15421_6

Figure 8.3: Barcharts showing best fitness obtained by the algorithms for noiseless data

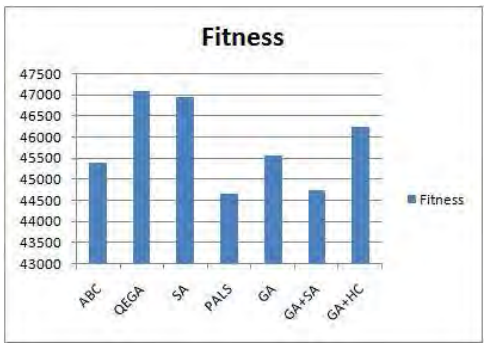
(a) m15421_7

(b) j02459_7

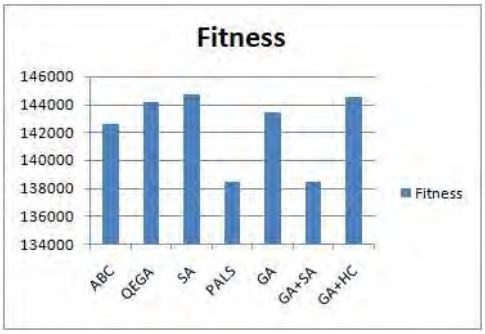
(c) bx842596_4

(d) bx842596_7

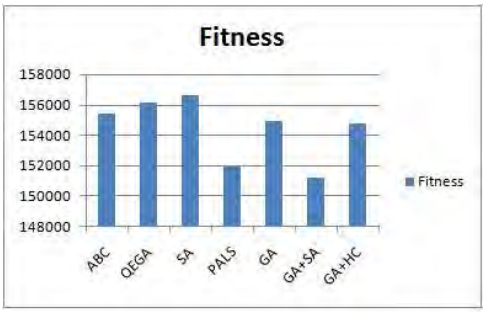
Figure 8.4: Barcharts showing best fitness obtained by the algorithms for noiseless data (cont.)



(a) acin1

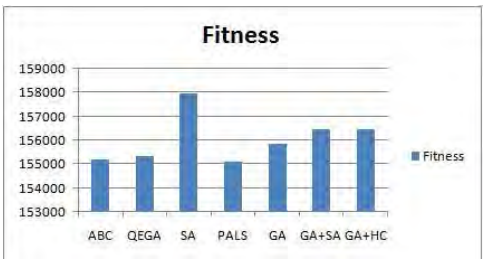


(b) acin2

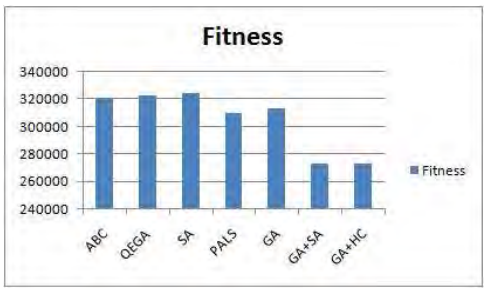


(c) acin3

(d) acin5



(e) acin7



(f) acin9

Figure 8.5: Barcharts showing best fitness obtained by the algorithms for noiseless data (cont.)

8.5.1 Results Obtained by ABC_FAP and QEGA_FAP for Noisy Instances (Fitness Criteria: Overlap and No. of Contigs)

As mentioned before, a cutoff value, i.e., required overlap between two adjacent fragments, is set in the *CalculateDelta* function of PALS (see subsection 6.5.3), used in QEGA_FAP and ABC_FAP algorithms. It provides one filter for spurious overlaps introduced by experimental errors [13]. It is to be noted that, in noisy instances, a lot of insertions, deletions or substitutions can occur. For instance, if we refer back to Table 8.2, there are 5694 insertions, 1469 deletions and 1988 substitutions on *j02459_7* instance in 454 error model. So, empirically we set the cutoff value for noisy instances to lower values than thirty, which was used as the cutoff value in noiseless cases. Nonetheless, we also ensure overlapping adjacent fragments. The results obtained by our ABC_FAP and QEGA_FAP algorithms are presented in Table 8.7. Previously, only [50] have introduced random error but there was no indication of how much error was being introduced. As opposed to them, we have clearly mentioned the parameters of our noisy dataset generation in Table 8.2. Instead of introducing random error in score matrix [50], we generated fragments with three error models (454, Sanger and Exact). These models are standard and pragmatic for DNA fragment generation. We execute our algorithms on these instances. We also take into consideration that DNA FAP is an off-line problem. So, instead of emphasizing on execution time of the algorithms, we concern ourself with assessment of the solution quality, based on number of contigs and fitness value.

As can be seen from Table 8.7, for most of the instances (except for those with multiple final contigs), QEGA_FAP outperforms ABC_FAP algorithm because of the elitist selection method imposed by QEGA_FAP. Our implementation of QEGA_FAP breeds fittest parent (i.e. queen bee) with other parents. The strong mutation of Algorithm 4 employs PALS to get the fittest mutants from the offspring. So we perform exploitation in the vicinity of the best solution of the current cycle. On the other hand, although ABC_FAP algorithm performs analogous action of mutation, but it does not

do so with the fittest ones. Exploitation can sometime leads to local minima and QEGA_FAP countermeasures this by random mutation. All the fitness values for QEGA_FAP and ABC_FAP algorithm shown in Table 8.7 are obtained for a single contig i.e. one continuous sequence with some exceptions. These exceptions can not reach single contig value and the final contig value for each of these instances are shown in brackets just beside their fitness value (obtained by QEGA_FAP and ABC_FAP) in Table 8.7. For QEGA_FAP and ABC_FAP, all other instances reach single contig, hence not mentioned in the Table for clarity (It is to be noted that GA, GA+SA and GA+HC, in the next Section 8.5.2, use only overlap as fitness evaluation criteria, so no value mentioned for contig number in the corresponding columns of GA, GA+SA and GA+HC in Table 8.7 does not mean single contig obtained by GA, GA+SA and GA+HC).

Effect of Cutoff Value on The Number of Contigs

As mentioned previously, to ensure overlapping between adjacent fragments, we need to define a cutoff value for each instances. But as errors are introduced, higher cutoff values cannot assure that we meet this criteria. So, empirically we need to setup a suitable cutoff value. We show the trend of decreased contig number with lower cutoff value in Figure 8.8. In this experiment, we used ABC_FAP algorithm to assemble the m15421_7 instance in 454 error model.

Fitness comparison for noisy and noiseless data using QEGA_FAP

Figures 8.6 and 8.7 show the progressive improvement of fitness values for the instance j02459_7 for noiseless and noisy data by applying QEGA_FAP. As can be seen from the figures, final fitness values for the noisy cases is significantly smaller than that of noiseless cases. Figure 8.6 also shows that QEGA_FAP is able to overcome any local minima as is evident from the transition of lower fitness value at almost 1100-th iteration to higher fitness value at almost 1200-th iteration.

Table 8.7: Best fitness obtained by the algorithms for noisy data

Instances	Error Model	Best Fitness				
		ABC	QEGA	GA	GA+SA	GA+HC
acin1	454	4323	4631	162	3738	4611
	Sanger	7758	8512	178	6744	8394
	Exact	17354	160007(4)	214	14157	17914
acin2	454	2271	2418	260	1778	2208
	Sanger	2981	3142	264	2413	2924
	Exact	20570	20884	267	19140	20677
acin3	454	2960	3170	326	1972	2657
	Sanger	3281	3661	344	2199	3095
	Exact	26611	10605(237)	374	12865	13694
acin5	454	3759	3994	405	2220	3113
	Sanger	4223	4533	430	2384	3540
	Exact	14694	16182	431	7069	15001
acin7	454	3759(2)	4792	490	2528	3487
	Sanger	3964	5177	516	2651	3817
	Exact	15052(4)	18864	522	8212	16029
acin9	454	4671	6315	608	2838	3828
	Sanger	5204	7881	623	2982	4735
	Exact	31914(2)	48154(107)	625	11816	38262
x60189_6	454	353	363	41	358	362
	Sanger	609	623	36	622	624
	Exact	2292	2303	36	2289	2302
m15421_7	454	978	1029	88	939	1026
	Sanger	1429	1467	97	1380	1455
	Exact	7028	7181	93	6840	7170
j02459_7	454	1670	1776	183	1429	1673
	Sanger	2466	2563	188	2095	2443
	Exact	13353	13607	179	12659	13526

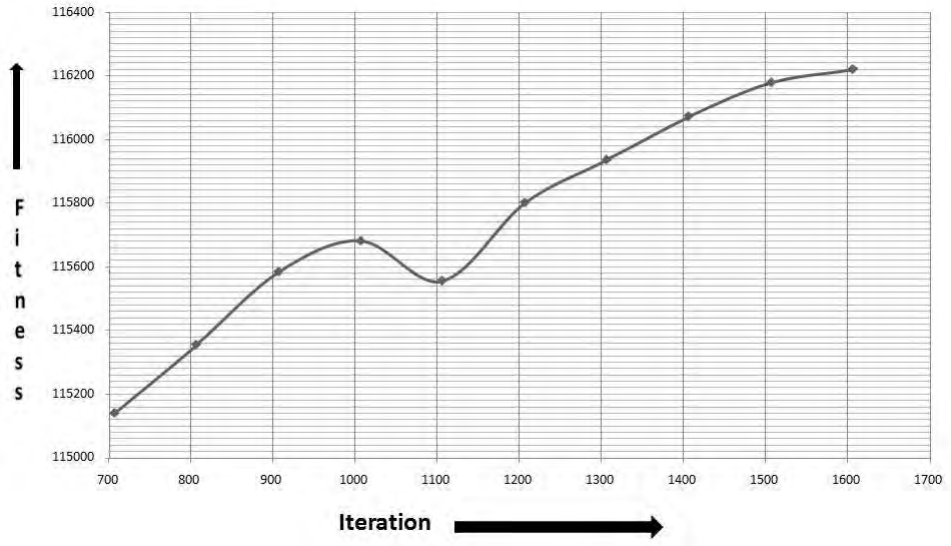


Figure 8.6: Iteration vs Fitness graph for j02459_7 noiseless data using QEGA_FAP

8.5.2 Results Obtained by GA, GA+SA and GA+HC for Noisy Instances (Fitness Criteria: Overlap)

The results obtained by our Genetic Algorithms (GA), Genetic Algorithm with Simulated Annealing (GA+SA) and Genetic Algorithm with Hill Climbing (GA+HC) are also shown in Table 8.7. As can be seen from Table 8.7, GA+HC performs better than GA and GA+SA. GA+SA probabilistically choose individuals with lower fitness. Although it is useful to overcome local maxima, with significant error introduced in instances, sometimes choosing a less fit individual can lead the result to astray. On the other hand, GA+HC constantly chooses the best fittest individual. Our observation is that, for this reason GA+HC performs better than GA+SA and GA. It is also to be noted that, GA+HC performs comparatively with QEGA, in terms of sum of overlap fitness criteria. Evidently, for noisy instances, elitism is better.

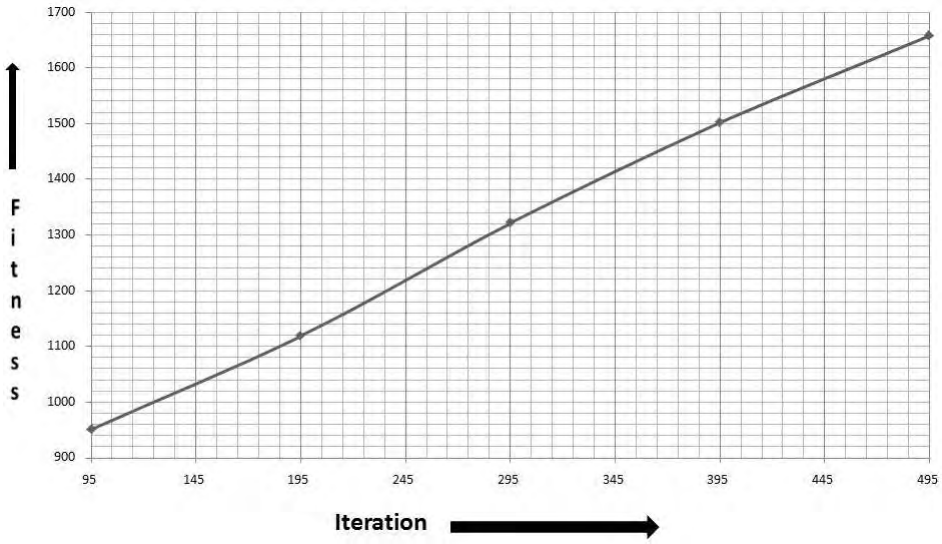


Figure 8.7: Iteration vs Fitness graph for j02459_7 noisy data(454 error model) using QEGA_FAP

8.6 Barchart Representation of the Results Obtained for Noisy Instances

Figures 8.9, 8.10, 8.11, 8.12, 8.13 and 8.14 represent the fitness values obtained by different algorithms for noisy datasets, with the help of barcharts.

8.7 Overall Performance of the Algorithms for Noisy and Noiseless Datasets

If we compare Tables 8.3, 8.4 and 8.7, we observe that the best fitness values obtained by different instances for noisy cases are significantly less than the best fitness obtained by them in case of noiseless cases. Among ABC_FAP, QEGA_FAP, GA, GA+HC and GA+SA, QEGA_FAP performs better for both noisy and noiseless cases. The number of contigs obtained in case of some noisy instances, by ABC_FAP and QEGA_FAP are much lower than noiseless instances, which might seem counterintuitive. But note that, we set lower cutoff values in case of noisy instances than noiseless instances. In

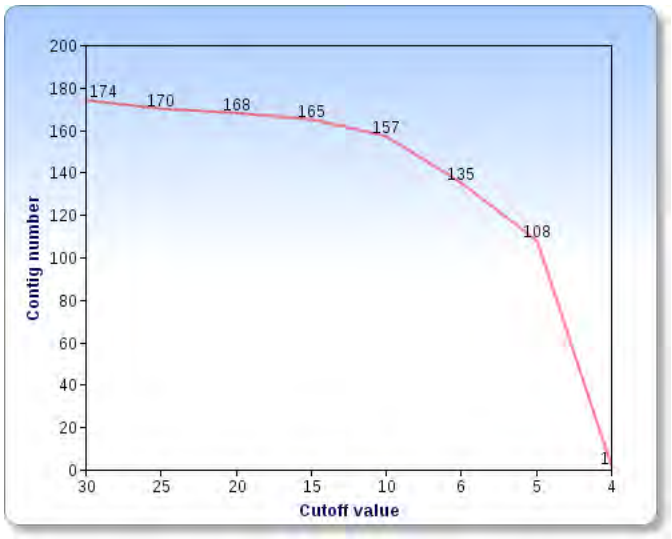
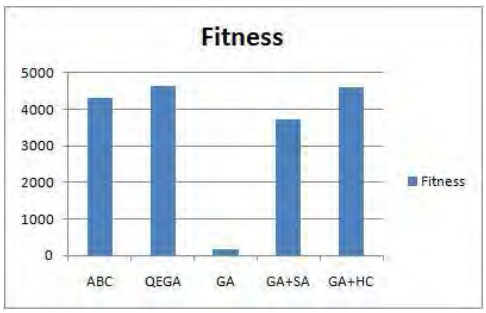


Figure 8.8: Relation between Contig number and Cutoff value for m15421_7 instance using ABC_FAP algorithm in 454 error model

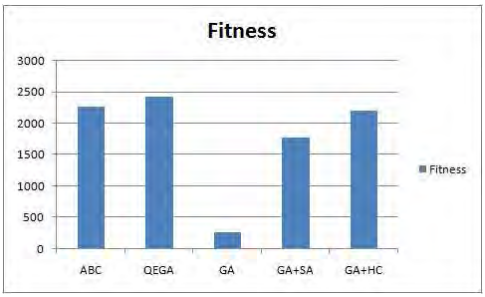
particular, for comparison with SA and PALS for noiseless cases, we set the cutoff value to a fixed value of thirty. This is not the case for noisy instances. We reported the best fitness obtained by ABC_FAP and QEGA_FAP in case of noisy datasets and empirically set the cutoff values to meet our fitness criteria, for each individual instance. This is justified by the fact that different amount of errors were being introduced for instances of various lengths, as evident from Table 8.2. We executed GA+HC and GA+SA with the hope of getting better solutions by occasionally considering bad solutions. Then we employed ABC_FAP with the objective of using collective intelligence to develop better solutions. We observe that ABC_FAP performs competitively with GA+HC. Lastly, we implemented QEGA_FAP to exploit elitism. This algorithm performed best in terms of final overlapping fitness value.

8.8 Statistical Analysis by One Way ANOVA

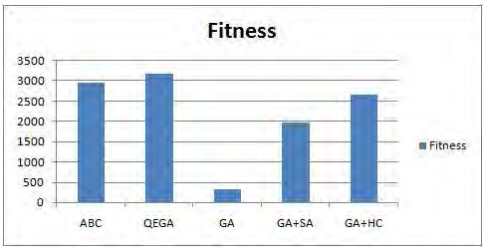
ANalysis Of VAriance, or ANOVA can be used to test for significant differences among sample means when the independent (predictor) variable is a set of discrete categories, and the dependent variable is continuous, ordinal, or dichotomous. For example, it can be employed to test the null hypothesis



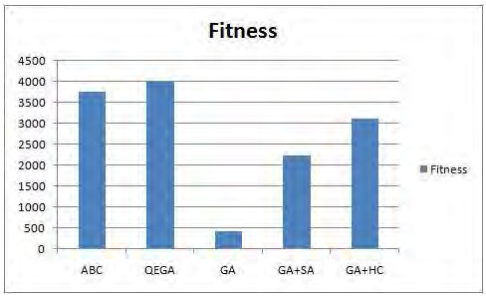
(a) acin1



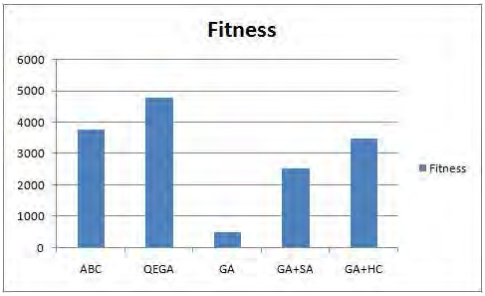
(b) acin2



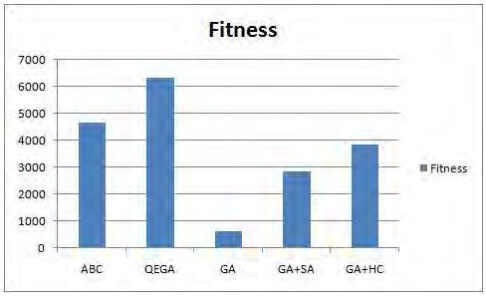
(c) acin3



(d) acin5

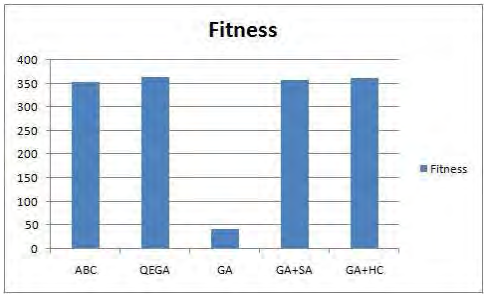


(e) acin7

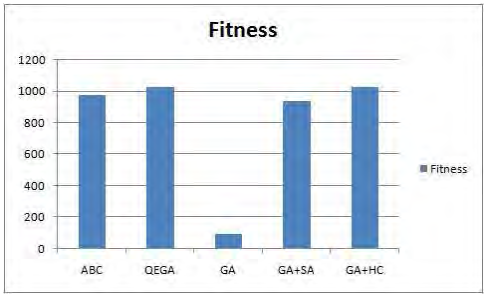


(f) acin9

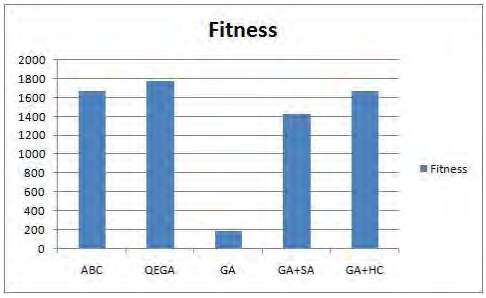
Figure 8.9: Barcharts showing best fitness obtained by the algorithms for noisy data (454 Sequencing Error Model)



(a) x60189_6

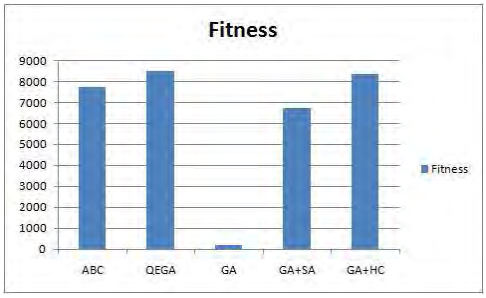


(b) m15421_7

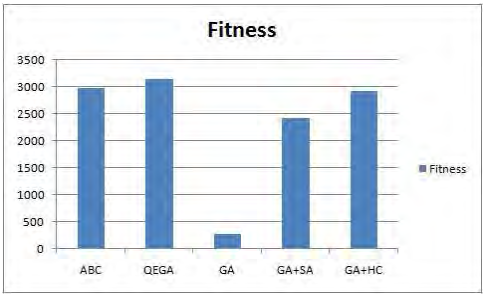


(c) j02459_7

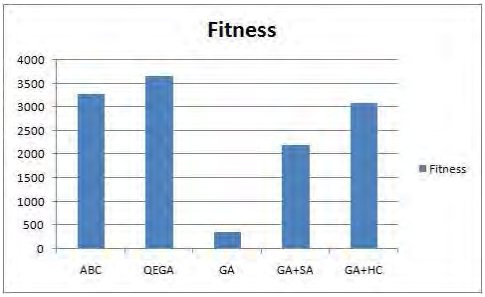
Figure 8.10: Barcharts showing best fitness obtained by the algorithms for noisy data (454 Sequencing Error Model)(cont.)



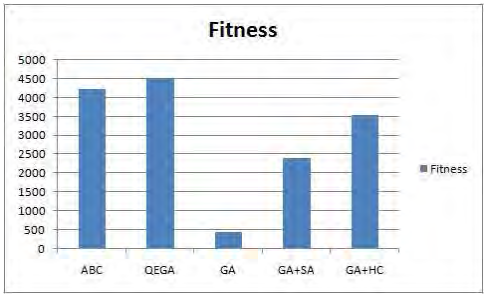
(a) acin1



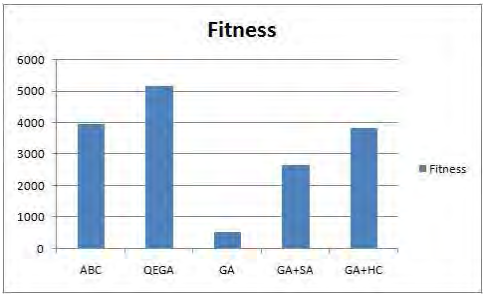
(b) acin2



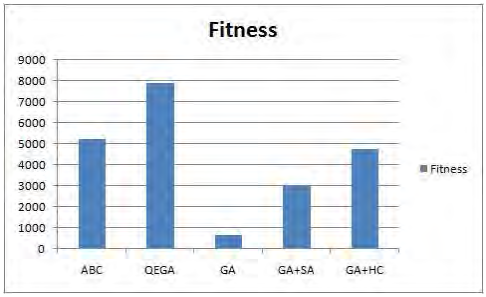
(c) acin3



(d) acin5

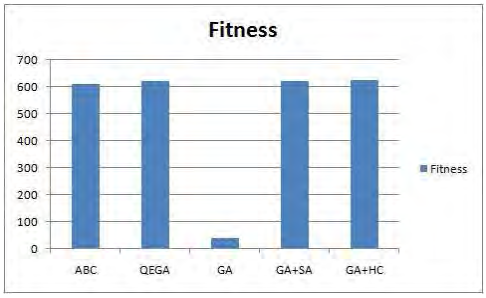


(e) acin7

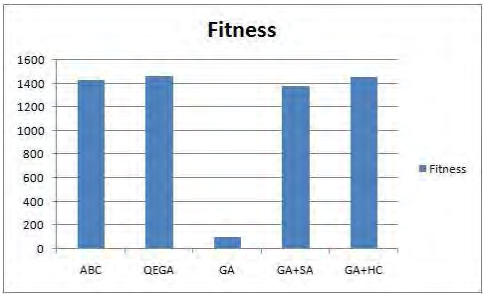


(f) acin9

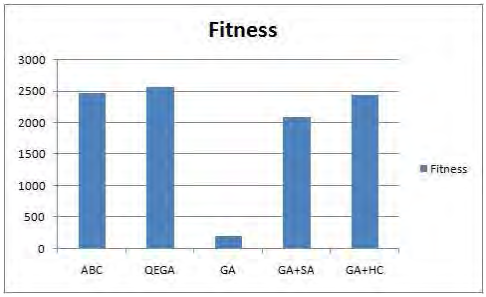
Figure 8.11: Barcharts showing best fitness obtained by the algorithms for noisy data (Sanger Sequencing Error Model)



(a) x60189_6

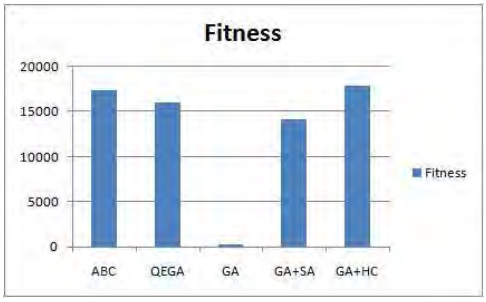


(b) m15421_7

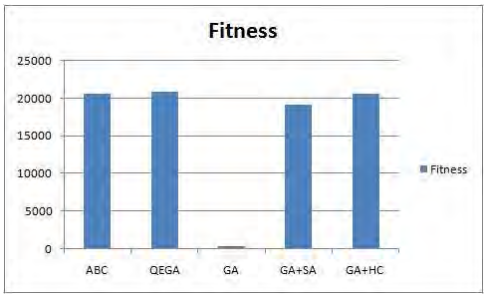


(c) j02459_7

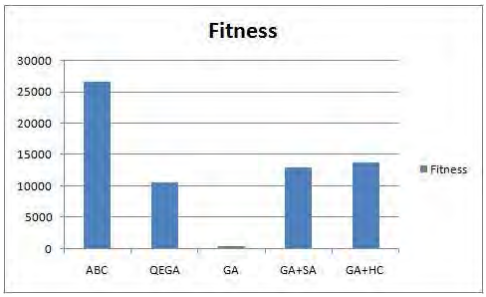
Figure 8.12: Barcharts showing best fitness obtained by the algorithms for noisy data (Sanger Sequencing Error Model)(cont.)



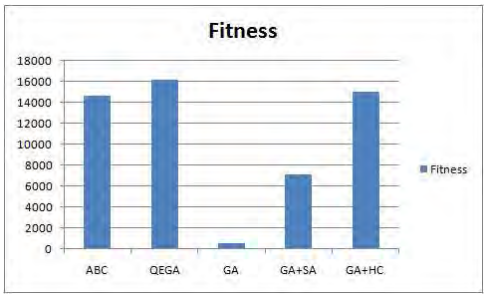
(a) acin1



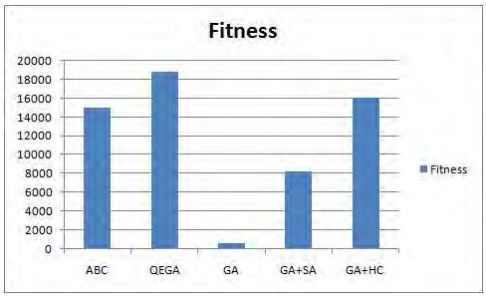
(b) acin2



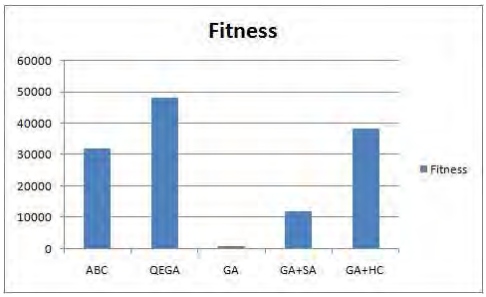
(c) acin3



(d) acin5

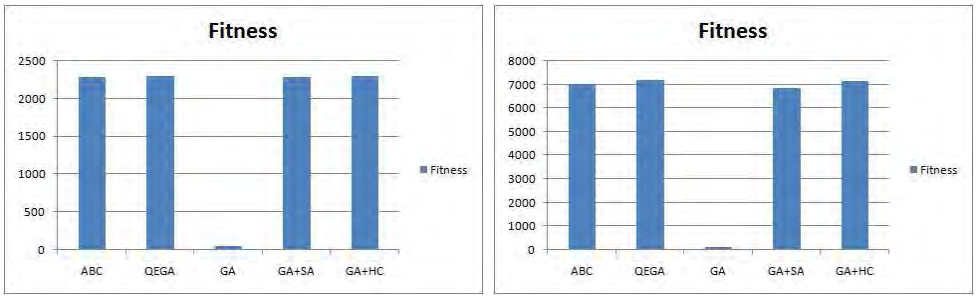


(e) acin7

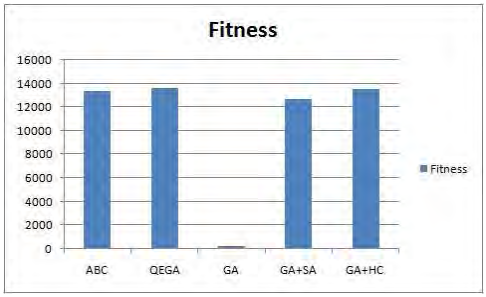


(f) acin9

Figure 8.13: Barcharts showing best fitness obtained by the algorithms for noisy data (Exact Sequencing Error Model)



(a) x60189_6 (b) m15421_7



(c) j02459_7

Figure 8.14: Barcharts showing best fitness obtained by the algorithms for noisy data (Exact Sequencing Error Model)(cont.)

Anova: Single Factor

SUMMARY

Groups	Count	Sum	Average	Variance
ABC	5	17984	3596.8	967710.2
QEGA	5	20528	4105.6	2223322
GA+HC	5	10489	2097.8	2931525
GA+SA	5	12546	2509.2	631233.2
GA	5	7689	1537.8	3180082

ANOVA

Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	22528052	4	5632013	2.834752	0.051751	2.866081
Within Groups	39735490	20	1986775			
Total	62263542	24				

Figure 8.15: ANOVA table (showing p-value)

that plumbers, electricians and carpenters all have roughly the same average income. Note that the t-test would have been used if the null hypothesis had concerned only two groups. In ANOVA test, we will be concerned with situations in which three or more sample means are compared with each other to test for statistically significant differences among those means and, in turn, among the means for their populations. The one-way ANOVA is used with one categorical independent variable and one continuous variable. The independent variable can consist of any number of groups (levels).

In our analysis, the algorithms act as independent variables and fitness of different DNA instances act as the continuous variable. The most interesting parameter returned by ANOVA is called p-value that indicates if data is statistically different (p-value is less than the confidence level) or if, on the contrary, we can not ensure that the data is different (p-value is equal to or greater than the confidence level). We use a confidence level of 5% and this test shows that there exists a significant difference among the algorithms (the p-value is always less than 0.05) (Figure 8.15).

8.9 Summary

In this chapter, we have presented the results obtained by Artificial Bee Colony (ABC_FAP) algorithm, Queen-bee Evaluation based on Genetic Algorithm (QEGA_FAP), Genetic Algorithm with Hill climbing (GA+HC) and Genetic Algorithm with Simulated annealing (GA+SA) for solving the DNA FAP. We executed our algorithms on both noisy and noiseless dataset. For noiseless instances, we compare our results with those obtained by the standard algorithms found in the literature. In the case of noisy dataset, we introduce three error models for generating the dataset and compare our algorithms. As has been mentioned before, in the recent and only previous work of [50] on noisy instances, in order to obtain the noisy data, the authors simulate the noise by changing the score for each instance randomly. In other words, they choose and change some values inside each score matrix (overlapping information) in a uniform way. But this is not the realistic scenario. In practice, the Sanger Sequencing and 454 Sequencing technologies are used for DNA sequencing. During DNA fragment generation phase, each of the Sanger Sequencing and 454 Sequencing technology, depending upon their sequencing technique, encounter errors introduced as insertions, deletions and substitutions. In our work, these error models are simulated using MetaSim to generate pragmatic datasets with noise. We, therefore, execute our algorithms on these realistic datasets and observed that the fitnesses achieved for noisy data are significantly lower than noiseless cases. This observation is vital as it points out the weakness of the so far reported fitness values obtained for noiseless data.

Chapter 9

Conclusion and Future Works

In this thesis, we have made an effort to solve the DNA Fragment Assembly Problem with five algorithms namely Artificial Bee Colony (ABC_FAP) algorithm, Queen Bee Evolution Based on Genetic Algorithm (QEGA_FAP), Genetic algorithm, Genetic Algorithm with Simulated Annealing (GA+SA) and Genetic Algorithm with Hill Climbing (GA+HC) for noiseless and noisy datasets. In one hybrid algorithm (GA+HC) hillclimbing is used as a mutation operator. In the other hybrid algorithm (GA+SA), a cooling scheme is employed as an operator to generate large local candidates within fewer generations so that we can achieve equal results as the previous one in smaller steps. In comparison of the five algorithms, QEGA_FAP outperforms all other algorithms for noisy cases. We have recorded the results obtained by our algorithm, taking into consideration various standard error models normally used by DNA sequencing experiments. Previous works on DNA FAP report their results for noiseless cases mostly. As has been mentioned before, in the recent and only previous work of [50] on noisy instances, in order to obtain the noisy data, the authors simulate the noise by changing the score for each instance randomly. In other words, they choose and change some values inside each score matrix (overlapping information) in a uniform way. But this is not the realistic scenario. In practice, the Sanger Sequencing and 454 Sequencing technologies are used for DNA sequencing. During DNA fragment generation phase, each of the Sanger Sequencing and 454 Sequencing technology, depending upon their sequencing technique, encounter errors

introduced as insertions, deletions and substitutions. In our work, three error models are simulated using MetaSim to generate pragmatic datasets with noise. We, therefore, execute our algorithms on these realistic datasets and observed that, although our algorithms perform very well for noiseless data just like others, for noisy instances their obtained fitness value decrease significantly for different error models. It is expected that, for noisy data, the fitness value would decrease. We have pointed out the fact that errors in DNA fragments significantly impact the performance of standard algorithms. Future works should take this into account.

In future, we should try to design algorithms to achieve near-optimal fitness for noisy instances, as we obtain for noiseless datasets. We can also explore the influence of repeated fragments on the sequencing process. Additionally, we can implement the parallel versions of our algorithms, as population based algorithms lend themselves suitable to be emulated in parallel environments. In particular, we can use OpenMP for shared memory programming and MPI as message passing interface for implementing parallel versions.

Bibliography

[1] http://www.454.com/downloads/news-events/how-genome-sequencing-is-done_FINAL.pdf. [Online; accessed 20-November-2011].

[2] http://www.scribd.com/doc/53306810/35/ORDERED-CROSSOVER. [Online; accessed 16-January-2012].

[3] http://en.wikipedia.org/wiki/Genome_sequencing#Sequencing_versus_analysis. [Online; accessed 16-April-2012].

[4] http://en.wikipedia.org/wiki/Human_Genome_Project. [Online; accessed 16-April-2012].

[5] http://www.bio.davidson.edu/courses/molbio/molstudents/spring2003/obenrader/sanger_method_page.htm. [Online; accessed 16-April-2012].

[6] http://ab.inf.uni-tuebingen.de/teaching/ss07/albi2/script/readsim_2July2007.pdf. [Online; accessed 16-November-2011].

[7] http://genome.ku.dk/resources/assembly/methods.html. [Online; accessed 16-April-2012].

[8] http://www.ncbi.nlm.nih.gov/nuccore/178817?report=fasta(M15421.1). [Online; accessed 20-November-2011].

[9] http://www.ncbi.nlm.nih.gov/nuccore/34645?report=fasta(X60189.1. [Online; accessed 20-November-2011].

[10] http://www.ncbi.nlm.nih.gov/nuccore/215104?report=fasta(J02459.1. [Online; accessed 20-November-2011].

[11] `http://0-www.ncbi.nlm.nih.gov.ilsprod.lib.neu.edu/Traces/wgs/?hide\_master\\*=\&val=ACIN02000001\#9(ACIN\_ALL)`. [Online; accessed 20-November-2011].

[12] Hussein A. Abbass. Mbo: Marriage in honey bees optimization a haplometrosis polygynous swarming approach. In *Proceedings of the Congress on Evolutionary Computation*, 2001.

[13] E. Alba and G. Luque. A new local search algorithm for the DNA fragment assembly problem. In *Evolutionary Computation in Combinatorial Optimization, EvoCOP'07*, Lecture Notes in Computer Science 4446, pages 1–12, Valencia, Spain, April 2007. Springer.

[14] Enrique Alba, Gabriel Luque, and Sami Khuri. Assembling DNA fragments with parallel algorithms. In *Congress on Evolutionary Computation*, pages 57–64, 2005.

[15] M. Adel Alimi Amira Hamdi, Nicolas Monmarch and Mohamed Slimane. Bee-based algorithms: a review. In *International Conference on Metaheuristics and Nature Inspired Computing, META10.*

[16] M.F. Azeem and A.M. Saad. Modified queen bee evolution based genetic algorithm for tuning of scaling factors of fuzzy knowledge base controller. *IEEE INDICON 2004 Proceedings of the India Annual Conference*, pages 299–303.

[17] DR Bentley. Whole-genome re-sequencing. *Current Opinion in Genetics and Development*, 16(6):545 – 552, 2006.

[18] Christian Blum, María J. Blesa Aguilera, Andrea Roli, and Michael Sampels, editors. *Hybrid Metaheuristics, An Emerging Approach to Optimization*, volume 114 of *Studies in Computational Intelligence.* Springer, 2008.

[19] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm intelligence: from natural to artificial systems.* Oxford University Press, Inc., New York, NY, USA, 1999.

[20] Marília D. V. Braga and Joao Meidanis. An algorithm that builds a set of strings given its overlap graph. In *Proceedings of the 5th Latin American Symposium on Theoretical Informatics*, LATIN '02, pages 52–63, London, UK, 2002. Springer-Verlag.

[21] C. Burks, M. Engle, S. Forrest, R. I. Parsons, C. Soderlund, and P. Stolorz. Stochastic optimization tools for genomic sequence assembly. In *M. D. Adams, C. Fields and J. C. Venter (Eds.), Automated DNA Sequencing and Analysis Techniques.* UK Academic Press.

[22] S. Cahon, N. Melab, and Talbi E.G. Paradiseo: A framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics*, 10:357–380, May 2004.

[23] Ting Chen and Steven S. Skiena. Trie-based data structures for sequence assembly. In *The Eighth Symposium on Combinatorial Pattern Matching*, pages 206–223. Springer-Verlag, 1997.

[24] Chin Soon Chong, Appa Iyer Sivakumar, Malcolm Yoke Hean Low, and Kheng Leng Gay. A bee colony optimization algorithm to job shop scheduling. In *Proceedings of the 38th conference on Winter simulation*, WSC '06, pages 1954–1961. Winter Simulation Conference, 2006.

[25] G. Churchill, C. Burks, M. Eggert, M. Engle, and M. Waterman. Assembling DNA sequence fragments by shuffling and simulated annealing. Technical report, Los Alamos National Lab, Los Alamos, NM.

[26] Swagatam Das, Ajith Abraham, and Amit Konar. Swarm intelligence algorithms in bioinformatics. *Computational Intelligence in Bioinformatics*, 147(2008):113–147, 2008.

[27] Bernabé Dorronsoro, Enrique Alba, Gabriel Luque, and Pascal Bouvry. A self-adaptive cellular memetic algorithm for the DNA fragment assembly problem. In *IEEE Congress on Evolutionary Computation*, pages 2651–2658, 2008.

[28] Habiba Drias, Souhila Sadeg, and Safa Yahi. Cooperative bees swarm for solving the maximum weighted satisfiability problem. In *IWANN*, pages 318–325, 2005.

[29] M. L. Engle and C. Burks. Artificially generated data sets for testing DNA sequence assembly algorithms. *Genomics*, 16(1):286–8, 1993.

[30] M. Margulies et al. Genome sequencing in microfabricated high-density picolitre reactors. *Nature*, 437(7057):376–80, 2005.

[31] C. E. Ferreira, C. C. de Souza, and Y. Wakabayashi. Rearrangement of DNA fragments: a branch-and-cut algorithm. *Discrete Appl. Math.*, 116:161–177, February 2002.

[32] P. Green. Phrap. `http://www.phrap.org/`. [Online; accessed 20-November-2011].

[33] Xiaoqiu Huang and Anup Madan. Cap3: A DNA sequence assembly program. *Genome Research*, 9:868–877, 1999.

[34] Cheol-Goo Hur, Sunny Kim, Chang Hoon Kim, Sung Ho Yoon, Yong-Ho In, Cheolmin Kim, and Hwan Gue Cho. Fasim: Fragments assembly simulation using biased-sampling model and assembly simulation for microbial genome shotgun sequencing. *Journal of Microbiology and Biotechnology*, 16(5):683–688.

[35] Michal Janitz. Next-generation genome sequencing: Towards personalized medicine. pages 1–282, 2008.

[36] Neil C. Jones and Pavel A. Pevzner. *An Introduction to Bioinformatics Algorithms*. MIT Press, 2004.

[37] Sung Hoon Jung. Queen-bee evolution for genetic algorithms. *Electronics Letters*, 39(6):575–576, 2003.

[38] D. Karaboga. An idea based on Honey Bee Swarm for Numerical Optimization. Technical Report TR06, Erciyes University, October 2005.

[39] Dervis Karaboga and Bahriye Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *J. of Global Optimization*, 39:459–471, November 2007.

[40] James P. Kelly. *Meta-Heuristics: Theory and Applications*. Kluwer Academic Publishers, Norwell, MA, USA, 1996.

[41] Satoko Kikuchi and Goutam Chakraborty. Heuristically tuned ga to solve genome fragment assembly problem. In *IEEE Congress on Evolutionary Computation*, Vancouver, BC, Canada, 2006.

[42] Jiří Kubalik, Petr Buryan, and Libor Wagner. Solving the DNA fragment assembly problem efficiently using iterative optimization with evolved hypermutations. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, GECCO '10, pages 213–214, New York, NY, USA, 2010. ACM.

[43] Sean Luke. *Essentials of Metaheuristics*. Lulu, 2009. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[44] G. Luque and E. Alba. Metaheuristics for the DNA Fragment Assembly Problem. *International Journal of Computational Intelligence Research*, 1(1-2):98–108, 2005.

[45] M Margulies, M Egholm, W E Altman, S Attiya, J S Bader, L A Bemben, J Berka, M S Braverman, Yi-Ju Chen, Z Chen, and et al. Genome sequencing in microfabricated high-density picolitre reactors. *Nature*, 437(7057):376–380, 2005.

[46] P. Meksangsouy and N. Chaiyaratana. DNA fragment assembly using an ant colony system algorithm. In *Proceedings of Congress on Evolutionary Computation*, volume 3, pages 1756–1763, 2003.

[47] Deirdre Meldrum. Automation for genomics, part one: Preparation for sequencing. *Genome Research*, 10(8):1081–1092, 2000.

[48] Deirdre Meldrum. Automation for genomics, part two: Sequencers, microarrays, and future trends. *Genome Research*, 10(9):1288–1303, 2000.

[49] M. M. Millonas. Swarms, Phase Transitions, and Collective Intelligence. In *C. Langton (Ed.), Artificial Life*, volume 3, pages 417–445, June 1993.

[50] Gabriela Minetti and Enrique Alba. Metaheuristic assemblers of DNA strands: Noiseless and noisy cases. In *IEEE Congress on Evolutionary Computation'10*, pages 1–8, 2010.

[51] Eugene W. Myers. Toward simplifying and accurately formulating fragment assembly. *Journal of Computational Biology*, 2:275–290, 1995.

[52] Gene Myers. A dataset generator for whole genome shotgun sequencing. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 202–210. AAAI Press, 1999.

[53] D. Teodorovic P. Lucic. Bee system: modelling combinatorial optimization transportation engineering problems by swarm intelligence. *In: Preprints of the TRISTAN IV Triennial Symposium on Transportation Analysis, Sao Miguel, Azores Islands, Portugal*, pages 441–445.

[54] Rebecca J. Parsons, Stephanie Forrest, and Christian Burks. Genetic algorithms, operators, and DNA fragment assembly. *Mach. Learn.*, 21:11–33, October 1995.

[55] Pavel A. Pevzner. Computational molecular biology - an algorithmic approach. pages 1–314, 2000.

[56] D. T. Pham, A. Ghanbarzadeh, E. Koc, S. Otri, S. Rahim, and M. Zaidi. The Bees Algorithm, A Novel Tool for Complex Optimisation Problems. In *Proceedings of the 2nd International Virtual Conference on Intelligent Production Machines and Systems (IPROMS 2006)*, pages 454–459. Elsevier, 2006.

[57] Frédéric Pinel, Bernabé Dorronsoro, and Pascal Bouvry. A new parallel asynchronous cellular genetic algorithm for de novo genomic sequencing. In *Proceedings of the 2009 International Conference of Soft Computing and Pattern Recognition*, SOCPAR '09, pages 178–183, Washington, DC, USA, 2009. IEEE Computer Society.

[58] L.D. Qin, Q.Y. Jiang, Z.Y. Zou, and Y.J Cao. A queen-bee evolution based on genetic algorithm for economic power dispatch. In *39th International Universities Power Engineering Conference*, volume 1, pages 453–456, September 2004.

[59] Daniel C Richter, Felix Ott, Alexander F Auch, Ramona Schmid, and Daniel H Huson. Metasim:a sequencing simulator for genomics and metagenomics. *PLoS ONE*, 3(10):e3373+, 2008.

[60] João Carlos Setubal and João Meidanis. Fragment assembly of DNA. *Introduction to Computational Molecular Biology*, pages 105–139, 1997.

[61] R Staden. A new computer method for the storage and manipulation of dna gel reading data. *Nucleic acids research*, 8:3673–3694, 1980.

[62] G.G. Sutton, White O., Adams M.D., and Kerlavage A.R. Tigr assembler: A new tool for assembling large shotgun sequencing projects. *Genome Science and Tech*, pages 9–19, 1995.

[63] El Ghazali Talbi. Metaheuristics from design to implementation. pages 1–593, 2009.

[64] Jason Teo and Hussein A. Abbass. A true annealing approach to the marriage in honey-bees optimization algorithm. *International Journal of Computational Intelligence and Applications*, 3(2):199–211, 2003.

[65] D. Teodorovic and M. Dell'Orco. Bee Colony Optimization – A Cooperative Learning Approach to Complex Transportation Problems. In *10th EWGT Meeting and 16th Mini-EURO Conference*, 2005.

[66] Horst F. Wedde, Muddassar Farooq, and Yue Zhang. BeeHive: An Efficient Fault-Tolerant Routing Algorithm Inspired by Honey Bee Behavior. pages 83–94. 2004.

[67] Xiaoqiu and Huang. A contig assembly program based on sensitive detection of fragment overlaps. *Genomics*, 14(1):18 – 25, 1992.

[68] Xin-She Yang. Engineering Optimizations via Nature-Inspired Virtual Bee Algorithms. pages 317–323. 2005.