وزارة التعليم العالي و البحث العلمي

Badji Mokhtar -Annaba- University
Année**: 2015**

جامعة باجي مختار- عنابة-

Faculty of Engineering Science
Department of Computer Science

T H E S I S

A thesis submitted in partial fulfillment
of the requirement for the degree of Doctor of Science

Development of New Bio-inspired Optimisation Approaches for Knowledge Discovery in Biological Data

Field : Computer science

Prepared By

Youcef GHERAIBIA

Thesis committee members:

Committee President	Pr. Nadir Farah	University of Annaba
Supervisor	Dr. Mazouzi Smaine	University of Skikda
Co- Supervisor	Pr. Abdelouahab Moussaoui	University of Setif
Advisor	Pr. Mouhamed Tarek Khadir	University of Annaba
Advisor	Pr. Hamid Siridi	University of Guelma
Advisor	Pr. Halima BAHI-ABIDAT	University of Annaba

Acknowledgements

This thesis would not have been possible without the help and support of the kind people around me. I would like to extend my thanks to all individuals who stood by me throughout the years of my study, but I wish to particularly mention a number of people who have had a profound impact on my work.

Above all, I am heartily thankful to my supervisors, **Pr**. **Abdelouahab MOUSSAOUI and Dr. Smaine MAZOUZI**, whose encouragement, guidance and support from start to end enabled me to develop my research skills and understanding of the subject. His wealth of knowledge, sincere advice made this research fruitful as well as enjoyable.

I offer many thanks and much gratitude to **Pr. Nadir FARAH** from the university of Annaba, **Pr. Mohamed Tarek Khadir** from the university of Annaba, **Pr. Hamid SIRIDI** from the university of Guelma and **Pr. Halima BAHI-ABIDAT** from the university of Annaba for evaluating and discussing this work.

I would also like to thank all my friends, for encouragement and for making the average work day more fun and interesting. Last but by no means least; I owe my deepest gratitude to my mother, my brothers and my sister for their unconditional support and patience from the beginning to the end of my study.

Abstract

In order to improve the NP-hard problem solvers with new meta-heuristics methods, many optimisation approaches based on natural phenomena such as animal ecology, biology, and other physical systems have been proposed. Recent work suggests that these methods can be further improved in order to obtain more precise search patterns by finding the optimal solution. Our work is in the field of bio-inspiration and combinatorial optimisation for bioinformatics problem. The developments of technologies in digital technologies have led in recent years, extremely large volumes of data, which may conceal useful information for organizations that produced them. This data can be in different form and from heterogeneous sources such as biological data. This constant has spawned a field of exploration: the extraction of knowledge from data, also known as KDD (Knowledge discovery for Databases). In this thesis, we developed a new approaches drawing on natural phenomena to solve hard problems based on biological data. We developed a new metaheuristics algorithm based on the penguins behaviors named PeSOA penguins search optimisation algorithm. We applied these new developed algorithms to a set of hard problems of Bioinformatics; biological sequences matching; biological data compression and DNA fragments Assembly.

Key words: Bioinformatics, Optimisation, Bio-inspired algorithm, DNA fragments assembly, Biological data compression, Biological sequences alignment.

ملخص

من أجل تحسين الحلول للمشاكل الصعبة ، تم اقتراح العديد من المناهج على أساس الظواهر الطبيعية مثل علم البيئة الحيوانية، وعلم الأحياء، والنظم المادية الأخرى .الأعمال الأخيرة تشير إلى أن هذه الأساليب يمكن زيادة تحسينها من أجل الحصول على أنماط بحث أكثر دقة من خلال إيجاد الحل الأمثل . عملنا في هذا المجال الحيوي الهام والتحسين التوافقي. وقد أدت زيادة التطورات في التقنيات الرقمية في السنوات الأخيرة، كميات كبيرة جدا من البيانات، والتي قد تخفي معلومات مفيدة للمؤسسات التي أنتجتها. وقد أفرزت هذه الثوابت في مجال التنقيب عن: استخراج المعرفة من البيانات، المعروف أيضا باسم استخراج البيانات أو استخراج البيانات المفيدة. وهذا هو السبب في أننا نقترح في هذه الأطروحة إلى تطوير التقنيات الارشادية التى تتركز على الظواهر الطبيعية لحل مشاكل معقدة متنوعة مثل تلك البيولوجيا الجزيئية، على سبيل المثال (التسلسل، المحاذاة، والتماثل، الخ).

كلمات مفتاحية:

البيومعلوماتية. التحسين، جمع قطع الدى ان اى، ضغط البيانات البيولوجية، تصفيف السلاسل البيولوجية

Résume

Afin d'améliorer les solutions aux problèmes NP-complet avec des nouvelles méthodes, beaucoup d'approches d'optimisation basées sur les phénomènes naturelles tel que l'écologie animale, la biologie, et d'autre systèmes physiques ont été proposés. Des méthodes récentes suggèrent que ces travaux peuvent être encore améliorés afin d'obtenir des modèles de recherche encore plus précis en recherchant la solution optimale. Les développements accrus des technologies numériques ont engendré depuis quelques années, des volumes de données extrêmement importants, qui peuvent receler des informations utiles pour les organismes qui les ont produites. Ce constant a donné naissance à un nouveau champ d'exploration: l'extraction de connaissances à partir des données. Dans ce travail nous avons développé un nouvel algorithme inspire à partir de phénomène naturel de chasse collaborative de pingouins et d'utiliser un algorithme très connu celui de l'algorithme génétique pour résoudre des problèmes complexe dans la bioinformatique. Nous avons appliqué ces méthodes sur trois problèmes de la bio-informatique, l'alignement de séquences biologiques, l'assemblage de fragments d'ADN et la compression des données biologiques. Les approches proposées dans cette thèse ont été évaluées et validées sur des données biologiques.

Mots clé : Bio-informatique, Optimisation, algorithme bio-inspiré, Assemblage de fragments d'DNA, Compression des données biologique, Alignement de séquences biologique.

Contents

List of Figures

List of Tables

List of Abbreviations

ASIL	Automotive Safety Integrity Level
GA	Genetic Algorithm
ACO	Ant Colony Optimisation
PSO	Particle Swarm Optimisation algorithm
TS	Tabu Search
MSA	Multiple Sequences Alignments
CS	Cuckoo search
FA	Firefly Algorithm
DE	Differential Evolution
DNA	Deoxyribonucleic acid
SAGA	Sequence Alignment by Genetic Algorithm
OLC	Overlap, layout, consensus
DFA	The DNA Fragment Assembly
SVM	support Vector Machine
HMM	Hidden Markov Model
FastHC	Fast Hill Climbing
VFastHC	Very Fast Hill Climbing
PHAGA	Parallel Hierarchical Adaptive GA
VNS	variable neighbourhood search
PESOA	Penguins Search Optimisation Algorithm
QuicklyOC	Quickly Overlap Complexity
Pe-SeeD	PeSOA for Seed
PHII	Pattern Hunter II
IR	Information Retrieval
CPU	Central Processing Unit
ACO-Seed	Ant Colony Optimisation for Seed
CCA	Cost Considering Algorithm
OCCA	Optimised cost considering algorithm
NCBI	The National Centre for Biotechnology Information
PeDFA	PESOA for DNA fragment assembly
TSP	Traveling Salesmen Problem
EMBL	European Molecular Biology Laboratory
QEF	Quantity of eaten fish

Part I

Introduction

Chapter 1

Introduction

1.1 Field of Research

The sizes of biological data banks are augmented with an exponential rate. For example, as of October 2015, the GenBank repository contains more than 188,372,017 sequences (The statistics are freely available at the repository website). In general the sizes of biological data banks are doubled each 15 months, those data are collected from computational analysis and scientific experiments. Biological data banks contain heterogeneous data such as DNA sequences and proteins structures (secondary, tertiary and quaternary). Extract useful knowledge from those data, analysing and searching on this very huge amount of data is become a hard tasks for the scientists.

In 1980s the scientists start working on DNA annotation process (genome annotation) consist as finding all coding region in a given genome (reference needed). This process aims to find the genetic materials hidden in the DNA sequences. This problem appears to be related to the quantity and the form of the data, after that more and more technologies are being integrated into every level of biological process simulation, understanding, data searching, and make biological data representation easier. Such domain of using computer science, mathematics and other technologies in biology is called bioinformatics (Hogeweg 2011). Bioinformatics has become the helpdesk for big part of biological studies. The definition of bioinformatics as submitted to the Oxford English Dictionary is:

(Molecular) bio informatics: bioinformatics is conceptualising biology in terms of molecules

(in the sense of Physical chemistry) and applying informatics techniques (derived from disciplines such as applied maths, computer science and statistics) to understand and organise the information associated with these molecules, on a large scale. In short, bioinformatics is a management information system for molecular biology and has many practical applications (Luscombe et al., 2001).

Bioinformatics since its inception has taken big intention by developing new algorithms in the field. These algorithms aim to handle the challenging problems of bioinformatics (Pevzner 2001), such as redundancy and multiplicity of data, biological database structures, biological data integration, gene transcription and regulation. Biological sequence structural studies and finding homologies between biological sequences. The study of the difficulty of a given problem is known as computational complexity theory (Rudish 2004). The aim of the computational complexity is the classification of the existent problems into classes and each class contain a set of similar problems. Most of problems in bioinformatics are considered as hard problems, means finding an efficient solution in reasonable time with restricted number of resources is a very difficult task. These kinds of problems are known as combinatorial problems. Solving combinatorial problems becomes a hot topic in all fields of big data analytics.
There are three major aims of bioinformatics (Luscombe et al., 2001):

1. In order to save the coherence of existent biological data with the newly founded one, the first aim of bioinformatics is to find an efficient representation of the huge amount of biological data in a way that simplifies data access, data searching and the updating process of data (add new entries to databases), all those problems are hard tasks looking to the quality and the heterogeneity of different sources of biological data.

2. Developing new tools and softwares to help scientist to analyse this amount of data. These tools are more than information retrieval applications (IR) because they require a preliminary understanding of these biological data. Understanding biological data aims to find links between those data and which is the first step to understand and

analyse the whole biological process.

3. The third objective of bioinformatics is the exploitation of the proposed tools with the well organised data to interpret and visualise the results in a corrected biological context.

As described earlier, most of bioinformatics problems are combinatorial problems, there are two kinds of solvers for these problems, exact and approximate algorithms. The exact algorithms are the first algorithms used to solve combinatorial problems. These kinds of algorithm are deterministic algorithms aims to find an exact optimal solution or all possible optimal solutions. The time and space requirement of such algorithms open a new research field which is the approximation methods.
The approximation methods are non-deterministic algorithms based on the optimisation of an objective function. The aim of this process is to produce a solution near to the optimal solution produced by exact methods, but in a reasonable time compared to the time required in exact methods. The complexity of the problem is the main criteria to choose which method is appropriate to the given problem. In the recent years several exact and approximate methods are proposed to handle with the bioinformatics problems such as multiple sequence alignment; structure pre diction and gene finding (reference needed) (the third chapter describe well the bioinformatics problems).

1.2 Research Question

This thesis aims at answering the following research questions:

1. What metaheuristics are currently well used to solve combinatorial optimisation problems in bioinformatics?

2. How can the problem characteristics, solution encoding schema and the used operators help the metaheuristics to perform well when solving combinatorial optimisation problem?

3. What is the popularity and capability of current nature inspired metaheuristics algorithms to solve with the combinatorial optimisation problems in bioinformatics?

The following objectives are formed to address our research question:

1. Discussing the existing nature inspired metaheuristic algorithms for combinatorial problems in bioinformatics by identifying the two main properties diversification and intensification.

2. Analysing biological data based problems, whether problem linked to organising biological data or combinatorial problems in bioinformatics.

3. Proposing new nature inspired based approaches to solve combinatorial problems or to properly represent the biological data. In this thesis, two kinds of problems have been attacked; **an efficient representation of biological data** and **new metaheuristics approaches for solving combinatorial problems** in bioinformatics.

1.3 Thesis Motivations

The main motivations for the research presented in this thesis were:

- To improve the existing metaheuristics algorithms by incorporating new mechanisms and searching strategies when solving combinatorial problems in bioinformatics.
- To show and demonstrating the powerful and the robustness of the new nature inspired metaheuristic approaches for solving hard problems.
- The effectiveness of the proposed approaches by applied it several bioinformatics problems such as DNA fragment assembly problems, Biological data compression and biological sequences alignments.

1.4 Thesis Structure

The thesis is divided in three major parts. First a general introduction of the thesis, after that the state-of-the-art of metaheuristics algorithms for optimisation in bioinformaticsis

discussed and the third part deals with the proposed approaches.

- Chapter 01: In this chapter we described the research filed of the thesis to introduce the problems of biological data. After that the motivation, the research question, and an overview of the thesis have been carried out.
- Chapter 02: In this chapter we described the general optimisation problems, nature, categories and general classification. We investigate a general overview about formal Modelling of an optimisation problem.
- Chapter 03: In this chapter we described the use of metaheuristics in bioinformatics and computational biology problems. The general combinatorial optimisation problem in bioinformatics is presented and the studied problems in this thesis are described in detail with the well-known approaches in the literature.
- Chapter 04: In this chapter we investigate the first studied problem for this thesis which is the similarity searching by handling the problem of finding multiple spaced seed. An experiment and comparison study with the well-known approaches is made to evaluate the efficiency of the proposed approach.
- Chapter 05: In this chapter we discuss the second contribution of the thesis which is a methodology for the biological data compression based on genetic algorithm. An efficient representation of biological data helps any algorithm to well manipulate the data in order to find the optimal solution in reasonable time.
- Chapter 06: This chapter describes the last contribution of the thesis which is the use of the penguin search optimisation algorithm for DNA fragments assembly. To study the DNA sequences we need to cut it in small fragments after that reconstructing the original DNA fragments is considered as combinatorial problem. We compared our results with the well-known methods in the literature.

1.5 Note on Publication

Some materials of this thesis have been published in a set of scientific journals and conferences.

1.5.1 Publications arising from and included in the thesis

1. Youcef Gheraibia, Abdelouahab Moussaoui, Youcef Djenouri, Sohag Kabir, Peng-Yeng Yin, Smaine Mazouzi. Penguin Search Optimisation Algorithm for Finding Optimal Spaced Seeds. International Journal of Software Science and Computational Intelligence (IJSSCI) Vol 7,Issue 2 (2015).

2. Youcef Gheraibia, Abdelouahab Moussaoui, Sohag Kabir, Smaine Mazouzi. PeDFA: Penguins Search Optimisation Algorithm for DNA Fragment Assembly. International Journal of Applied Metaheuristic Computing. (In Press)

3. Youcef Gheraibia, Sohag Kabir, Abdelouahab Moussaoui, Smaine Mazouzi. Penguin Search Optimisation Algorithm for Finding Optimal Spaced Seeds. International Journal of Information and Communication Technology.

4. Gheraibia Y, Moussaoui A: Penguins Search Optimisation Algorithm (PeSOA). Recent Trends in Applied Artificial Intelligence, 2013 pp 222-231.

1.5.2 Other publications

1. Youcef Gheraibia, Abdelouahab Moussaoui, Luis S. Azevedo, David Parker (UK), Yiannis Papadopoulos, Martin Walker. Can Aquatic Flightless Birds Allocate Automotive Safety Requirements? 2015 IEEE Seventh International Conference on Intelligent Computing and Information Systems. Cairo, Egypt, 2015.

2. Youcef Gheraibia, David J Parker, Yiannis I Papadopoulos (2015), Automatic Decomposition and Allocation of ASILs using PeSOA. 7th Annual Departmental Conference for Postgraduate Research. Department of Computer Science, University of Hull, Hull, UK, February 2015.

3. Sohag Kabir, Tanzima Azad, Martin Walker and Youcef Gheraibia (2015), Reliability Analysis of Automated Pond Oxygen Management System. 18th International Conference on Computer and Information Technology (ICCIT'15). Military Institute of Science and Technology, 21-23 (Mon-Wed) December 2015. Dhaka, Bangladesh.

Part II

State of the art

Chapter 2

Optimisation and Metaheuristics

2.1 Introduction

The optimisation problem is a kind of problems where an exact solution can't be found in reasonable time (Kann 1992). Optimisation is everywhere, such as aeronautics, biology, and economy. Optimisation has been widely used in Bioinformatics to solve combinatorial problems with biological data. The nature of manipulated data has been used to classify the optimisation problems in two categories (discrete and continuous optimisation problems). The nature of the objective and the feasibility of the solution can also classify optimisation problems in different categories. In the current chapter, we present the general model of an optimisation problem, by describing the categories of different optimisation problems. After that we describe the optimality searching characteristics and the well-known metaheuristics for the optimality searching. We described an algorithm from each metaheuristic category. Ant Colony Optimisation algorithm, Particle Swarm Optimisation for population based algorithms, neighbourhood searching and discontinuous algorithms. Genetic algorithm from evolutionary algorithms, and Tabu search for trajectory based algorithms and memory usage algorithms.

2.2 Combinatorial Optimisation

Computational complexity theory is a subfield of theory of computation that classifies the practical difficulty of solving problems (Hazewinkel 2001). The time consuming of the solving methods is the main criteria to classify those methods. Combinatorial optimisation consists of finding the optimal object from a set of finite objects. Combinatorial optimisation methods can be divided mainly into two categories: Exact and approximate methods. Exact methods are the methods that provide the exact solution or all possible exact solutions for the problem. There is a large number of exact solvers for combinatorial problem such as, Dynamic programming (Bellman 1958) and Branch and Bound (Land et a., 1960). These methods are time consuming and hard to be used in a complex problems. The second category of the combinatorial problem solvers is the approximate methods (Vazirani 2003), aims to find approximate solutions, near the best solution or the best solution means that it doesn't guarantee to find the best solution. Approximate algorithm is often used for problems that don't have an algorithm in polynomial time, this class of algorithm called NP-hard (non-deterministic polynomial-time hard) (Daniel et al., 1994). The approximation ratio is the ratio between the result obtained by the algorithm and the optimal profit. Further detailed information about this specific class will be synthesised extensively in the next section.

2.3 Optimisation

ISO 26262 defines the functional safety standard for the passenger vehicle industry, one of the main problems for vehicle designer is to find **the optimal** allocation and decomposition of the Automotive Safety Integrity Levels (ASILs) to the different components of the system (Papadopoulos et al., 2010). This allocation must satisfy some requirements called Custsets (CS) and ensure the minimum cost of the total allocation. The airline industry faces one of the largest scheduling problems of the industrial transport related services. The objective is to find **the efficient** planning and scheduling for the different aircrafts and staffs (Farah et al., 2011). Optimisation is everywhere, peoples always try to find the

optimal, the efficient, the best, the ideal, the perfect, the decisive, the powerful, or the productive solution to their problems. The optimisation is one of the most used tools in decision aid science and in the analysis of real systems (Belegundu et al., 2011).

What we need (martials) in order to use the optimisation methods to solve a given problem?

The main thread is to define the objective(s) of the problem, means to set a quantitative measure to compute the robustness of a given solution of the problem. This objective is based on all influenceable parameters and their allowed values of the systems which will be considered as unknowns in the optimisation process. The optimisation problem which can be seen as the maximisation or minimisation of a function subject to a set of constraints on its variables (Nocedal et al., 2006). The mathematical modelling of an optimisation problem is as follows:

Let $\boldsymbol{x}$ be a set of parameters (vector of variables), f is a function of $\boldsymbol{x}$, and $\boldsymbol{C}$ is a vector of constraints on the variables $\boldsymbol{x}$. The optimisation problem can then be stated as follows:

$$\begin{aligned} \text{Objective: To find } x = & \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \\ \text{Which Minimise/Maximise} & \quad \boldsymbol{f(x)} \\ \text{Subject to:} & \quad C_i(x) \end{aligned}$$

The optimisation problems can be classified in numerous ways. First, discrete optimisation problems which refer to the class of optimisation problems which the solution we investigate is one of a number of objects in a finite set (Lee 2004). The discrete variables can take only a finite number of values such as the qualitative and categorical variables, like when we want to find the best instructions that improve the performance of machines. For some problems, the variables are restricted to be integers, this kind of problems are known as integer programming problems ($x \in \mathbb{Z}$). An example of integer programming problem is the ASIL allocation problem, it defines a set of safety levels to ensure the required total safety of the system. It would not make sense to advise the company to allocate

the third and half safety level to such components. On the other hand there is continuous optimisation problems when the solution we investigate is from an unaccountably infinite set, usually real numbers ($x \in \mathbb{R}$). If the variable $\boldsymbol{x}$ can contain any value within some range is called continuous (Boyd et al., 2004).

In discrete and continuous optimisation problems the $\boldsymbol{x}$ variable vector can take value in specific domain. In some optimisation problems some values of the $\boldsymbol{x}$ variable's domain are not allowed. Optimisation problems can be divided in two classes. First, unconstrained optimisation problems with unrestricted domain for the $\boldsymbol{x}$ variable's, means you have to optimize the objective function without having to care about the variables values. Second, constrained optimisation problems with restricted domain for the variables $\boldsymbol{x}$, such as peoples maximizing benefits subject to a budget constraint. The solution within the allowed rang, are called feasible solutions (Verfaillie et al., 1996).

The aim of optimisation is to find the optimal solution in a given set of feasible solutions. This optimal can be local optimum or global optimum, if the optimum is the best solution among all the feasible solutions, this solution is called global optimum solution. The global optimum is better according to the objective function than for all $\boldsymbol{x}$ in some open interval containing this solution. In some applications, it's difficult to identify the global best solution, an optimal solution, called local optimum is found in reasonable time, this solution is better than its entire neighbourhood. This local optimal is better according to the objective function than all $\boldsymbol{x}$ in the allowed domain. In a given problem there can be many local optimum that are not global optimum (Horst et al., 2000).

The function $\boldsymbol{f}$ is used to decide that this solution is better than other solutions, $\boldsymbol{f}$ is called objective function and it serves as the criteria to determine the quality of the solution. Optimisation problems can contain only one objective function with a set of constraints, called single objective optimisation problems. A single objective function with several constraints may not adequately represent an optimisation problem, in this case we might need to represent the problem with several objective functions called multi-objectives optimisation problem, called also multi-criteria optimisation, or multi-attribute optimisation (Hwang et al., 1979). Optimisation problem can be without any objective function, (for example, design of integrated circuit layouts), the objective is to find a set of variables that satisfies

the model constraints. Designer doesn't need to optimise anything, this kind of problems known as feasibility problems.

2.4 Optimality Searching Algorithms

Searching for the optimal solution among all possible solutions of an optimisation problem is like finding solution to the maze game. The only guarantee is that at least one solution exists, but the time required to achieve this solution is up to the used strategy to find it. Suppose that we search for the path without any guidance, the search process can be in infinite time hence of the purely random search. A huge number of methods have been proposed to improve the optimal solution finding by developing new searching strategies. The optimisation method can be classified into two categories based on the relation between the problem and the method to solve this problem. The heuristics are a problem dependent methods they take in consideration the advantage of the problem to converge quickly to optimal solution, the heuristics are specific methods adapted to a given problem (Jon 1983). The second category sets is the metaheuristics which are a problem-independent methods, they are general methods to solve all optimisation problem and they do not take advantage of any specificity of the problem (Blum et al., 2003). Large numbers of metaheuristics have been proposed to solve combinatorial problems. These methods can be classified in numerous ways, the classification are not absolute, means the same propriety can be used by several algorithms in different ways. Many studies have been carried out on the metaheuristics classification. Author in (Blum et al., 2003) proposes a large classification based on number of solutions per iteration, the use of the search history, the kind the objective function and the source of inspiration, these characteristics are organised as follows:

- **Trajectory methods Vs. Discontinuous methods:** The trajectory methods aim to find the optimal solution of an optimisation problem by exploring the possible solution from the neighbourhood of this feasible solution only. The discontinuous methods explore the solutions in a trajectory form and also can jump to solution in other region of the solution space (Betts 1998).

- **Population-based Vs. Single-point search:** the population based optimisation methods use a set of feasible solution at each iteration, and the single-point based optimisation methods used one feasible solution and start exploring the solution space from this solution (Parpinelli et al., 2011).

- **Memory usage Vs. Memoryless methods:** the difference between the methods that uses memory and the methods that don't use memory is the use of the historical information (previous visited solutions) to control further amelioration of the objective function but exploring new non visited solutions (Rego et al., 2006).

- **Nature-inspired Vs. Non-nature inspiration:** Some optimisation algorithms are based on natural and biological phenomena. This kind of algorithm imitates the behavior of biological process or swarm intelligence based methods (Yang et al., 2010).

These characteristics identify the search strategy and the structure of metaheuristic algorithms. We can summarise these characteristics in two main categories: the number of solution manipulated by the algorithm and the use of the search history. In the following we describe some important meta-heuristic techniques that have been widely used for solving optimisation problems.

2.4.1 Genetic Algorithm

Genetic Algorithm (GA) is a bio-inspired metaheuristic algorithm developed by (Goldberg et al., 1988). GA is a stochastic optimisation algorithm imitates the natural evolution process of genomes. The GA is based on three operations, selection, crossover and mutation. These operations are applied iteratively to improve the quality of the solutions (see Figure 2.1). GA starts by generating a population of random feasible solutions (individuals), the initialisation can be made in different ways, the basic GA uses random initialisation of the start population, each solution is considered as an individual on the population.
GA optimisation process is based on a set of operators, allow the algorithm to improve the intilai population in ordre o achieve the optimal solution.The optimisation process of GA is

as follow: two solutions are selected among the individuals of the initial population, by one of the well-known selection techniques. This two selected solutions will be considered as two fathers, this two, later will be crossed to generate two other new solutions considered as (Children), this new solutions (Children) can be mutate according to a given mutation probability. The quality of each solution is computed with the fitness function, this function controls the evolution of the GA population by the deletion of bad and insertion of good solutions in the population. These processes are repeated until the termination criteria is achieved which can be the number of generation or if the population is stabilised. A simple description is presented in the next sections.

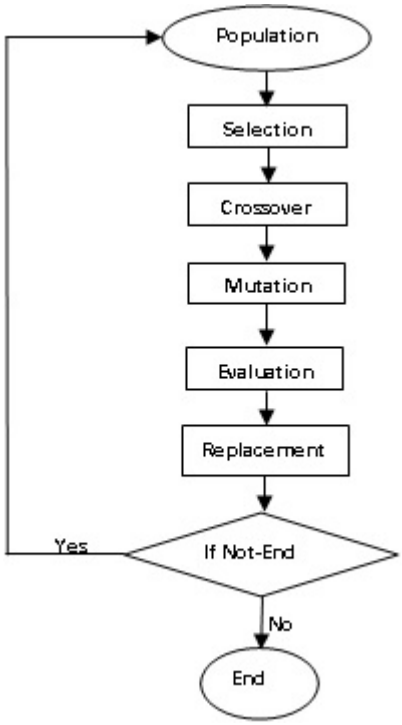

Figure 2-1: Genetic algorithm flowchart

2.4.1.1 Selection

The selection is the first operator of the genetic algorithm. At each iteration, part of the population will be selected to be the candidate for the other genetic algorithm operators to generate the new population, usually the new population better than the previous one. The basic selection method is the random selection by generating random number between

2 and the size of the population where each candidate solution has its own identification. Other selection methods have been proposed to improve the convergence of the genetic algorithm by guiding the selection to the high quality solutions (Blickle et al., 1995).

2.4.1.2 Crossover

The previous selected solutions will be the operands of the crossover operation in order to generate new solutions. The main objective of the crossover is to benefit from the two solutions to generate better solutions. The crossover operation is made in two steps, first a cut point is selected and secondly the cut parts are merged in order to create new solutions (see Figure 2.2). Several crossover methods have been proposed in the literature (Osaba et al., 2014).

The crossover operations may provoke a conflict in the new solutions by missing some solution part. However, in each of the solutions, some symbols are repeated and some are missed, but numbers of repeated and missed symbols are equal. Some problems need after each crossover operation a regulation step to check the feasibility and the correctness of the new generated solutions.

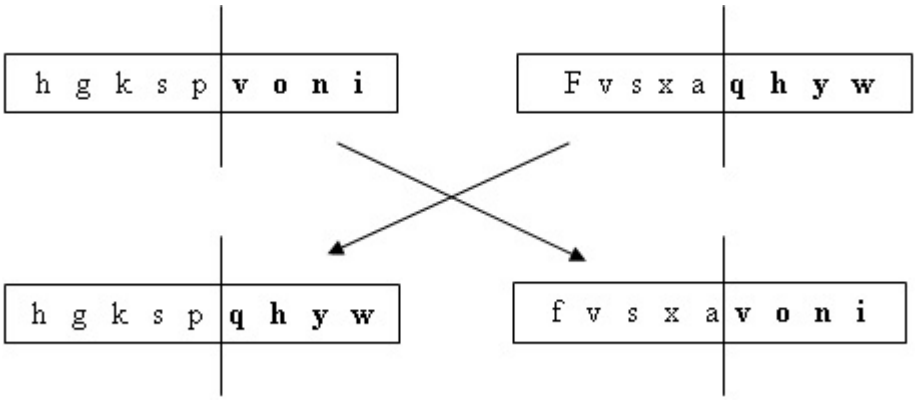

Figure 2-2: Crossover Operation

2.4.1.3 Mutation

After the creation of new solutions by the crossover operation, these solutions are mutated. The aim of the mutation operator is to ensure a good diversification of the new solution

(see Figure 2.3). The algorithm for this operator goes through the solution and changes the value of a given position. The selection of the position to be mutated is done following a fixed mutation rate (Osaba et al., 2014). The mutation operator has been merged in several other metaheuristics algorithms (Pant et al., 2008), (Zhao et al., 2010).

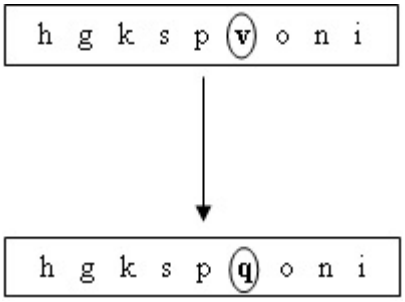

Figure 2-3: Mutation Operation

Genetic Algorithm has been widely used to solve combinatorial optimisation problems. The Travelling Salesmen Problem (Grefenstette et al., 1985) with all its variants such as multiple travelling salesmen problem (Bektas2006) and Vehicle Routing Problem (Baker2003) is one of the first uses of the genetic algorithm. Bioinformatics is one the widest areas that uses GA in different bioinformatics and computational biology, from phylogenetic tree construction (Lewis 1998), DNA fragment assembly (Nebro et al., 2008) and multiple sequences alignment (Notredame et al., 2000). The wide use of GA in bioinformatics is because of the similarity between the representation of biological data and the GA encoding scheme. Economics and scheduling application of GA such as the generalised assignment problem (Chu et al., 1997). GA has been applied also to industrial problems such as solving the machine-component grouping problem required for cellular manufacturing systems (Onwubolu et al., 2001).

2.4.2 Ant Colony Optimisation

Ant Colony Optimisation (ACO), is a nature inspired metaheuristic algorithm based on the collaborative strategy of Ants colonies (Dorigo et al., 2010). ACO based approach formulate the problem as a graph and the objective of the algorithm is to find the optimal path among the possible paths of the graph. The ACO is considered as Swarm intelligence

algorithms, the candidate solutions construct an ant population. At first ants straggle randomly to form the initial population of candidate solutions, after that each ant explores the graph based on two parameters the heuristic costs and the pheromones, the first represent the quality of the path according to the objective function and the second represent the quality of the path according to results obtained by the other ants. The pheromones considered as marker to communicate between ants, the path that contains maximum value of pheromones is the best compared with paths that have low pheromone. Ants use this marker with certain probability to choose the next move in the optimisation process; some ants are still search randomly for closer food sources.

Figure 2.4 shows the construction of pheromones by the ant colony, first the ant colony follow the simple pheromone trail (direct path) (see figure (2.4.1)), when an obstacle appear in the path, the ants have no idea about the new optimal path (see figure (2.4.2)), so ants avoid the obstacle usually in some possible paths around the obstacle (see figure (2.4.3)), each ant will updates the pheromone of the explored path, the best path among the explored paths will have the maximum value of the pheromone. After a number of iterations ants will follow the path that contains high value of pheromone and create new pheromone trail (see figure (2.4.4)).

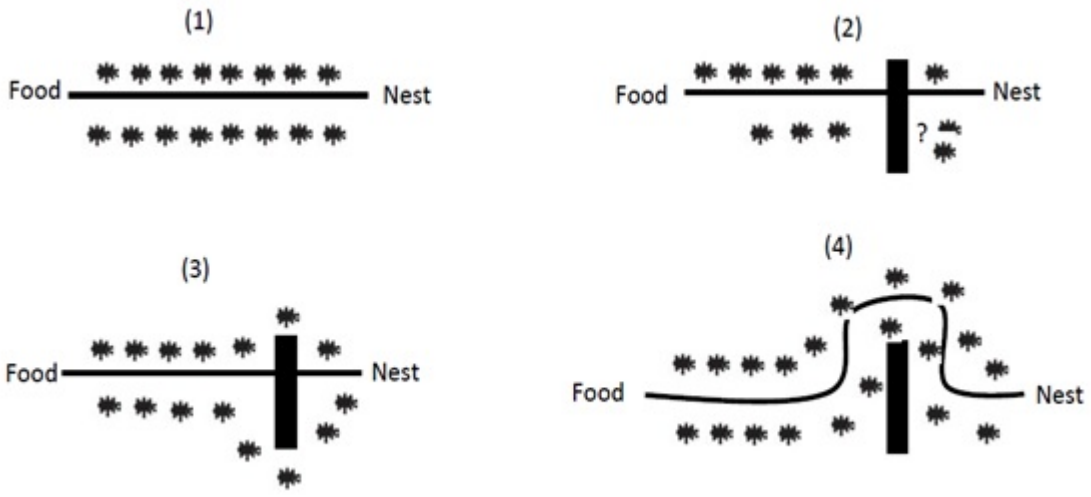

Figure 2-4: Ant colony pheromones construction

ACO algorithm works as follow (Figure 2.5): first the algorithm parameters are initialised to be used by ant to explore the search space, after that the population of the initial

candidate solutions is initialised. These solutions are evaluated by the objective function in order to check the goodness of the solutions and to use it for the pheromone update (deposition and evaporation). Finally the termination criteria is reached, if not the algorithm will reconstruct new solution based on the new pheromone values.

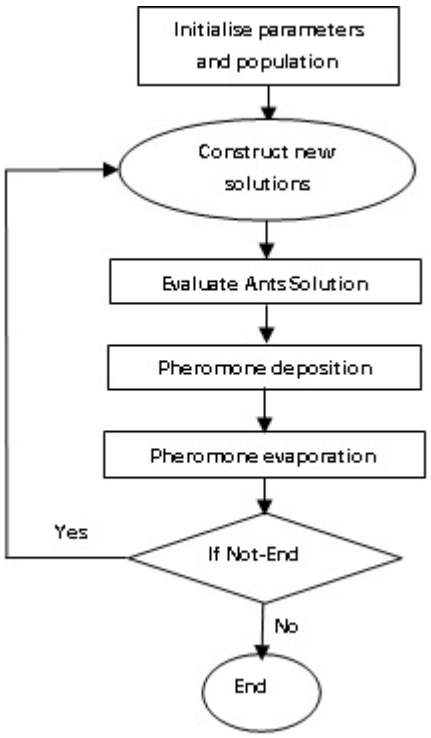

Figure 2-5: Ant colony Algorithm

Ant Colony Optimisation Algorithm has been widely used to solve NP-hard problem. ACO based algorithm always formulate the problem as graph, all problems with graph formulation have find a good framework with ACO such as Travelling Salesmen Problem (StÃijtzle et al., 1999) and Vehicle Routing Problem (Bell et al., 2004), Graph Colouring problem (Comellas et., al 1998). ACO find a lot of interest in image processing, such as image segmentation (Lee et al., 2009). Data mining main tasks wildly use the ACO algorithm in classification, clustering and association rule mining (Parpinelli et al., 2002).

2.4.3 Particle Swarm Optimisation

Particle Swarm Optimisation algorithm (PSO) is a swarm intelligence metaheuristic algorithm developed by (Kennedy2010). PSO is based on the research of bird and fish flock

movement behaviour. The food searching process of birds is a collaborative strategy to find the source of nurture. PSO is a very simple optimisation algorithm, PSO like GA and ACO, it manipulates a set of candidate solution at time (population) and unlike GA, PSO has no natural evolution operator such as crossover and mutation. The particle of the PSO population communicates using velocity to share to each other the location of food. The PSO algorithm work as follows (Figure 2.6): the PSO algorithm starts by creating the initial position of the particle, each particle on the initial population has its velocity. After that all particles will be evaluated using the objective function to determine the best position which has the lowest objective function (Minimisation problem). For each individual, a new velocity is computed based on the current position and the best solutions (PBest: Personal Best and GBest: Global Best)) of the whole population. This velocity is used to generate the new move of the particle. The new solution is conserved only if the objective function value is better than the previous one, the algorithm will iteratively repeat these operations until the termination criteria is reached.

PSO has been successfully applied to many areas such as machine learning, pattern recognition and data mining have used PSO as optimisation algorithm for several application area. Signal processing such as MIMO transceiver design (Chen et al., 2010) and ECG classification (Melgani et al., 2008). PSO has been also used in bioinformatics such as protein motifs discovery (Chang et al., 2004).

2.4.4 Tabu Search

Tabu Search (TS) is a metaheuristic algorithm proposed by Fred Glover in 1989, to allow hill climbing to defeat the local optima (Glover et al., 1989). Tabu search is trajectory method; it uses one candidate solution and changes this solution iteratively in order to improve the quality of it. TS is also a memory based algorithm, is uses the history of visited solution to guide the new moves. The TS can be considered as simple descent method where the objective is to minimise a variable x, the algorithm allows moves only to neighbour around the candidate solution. The use of memory by TS can be categorised in three classes (Short-term, intermediate-term and long-term). Short term memory use means

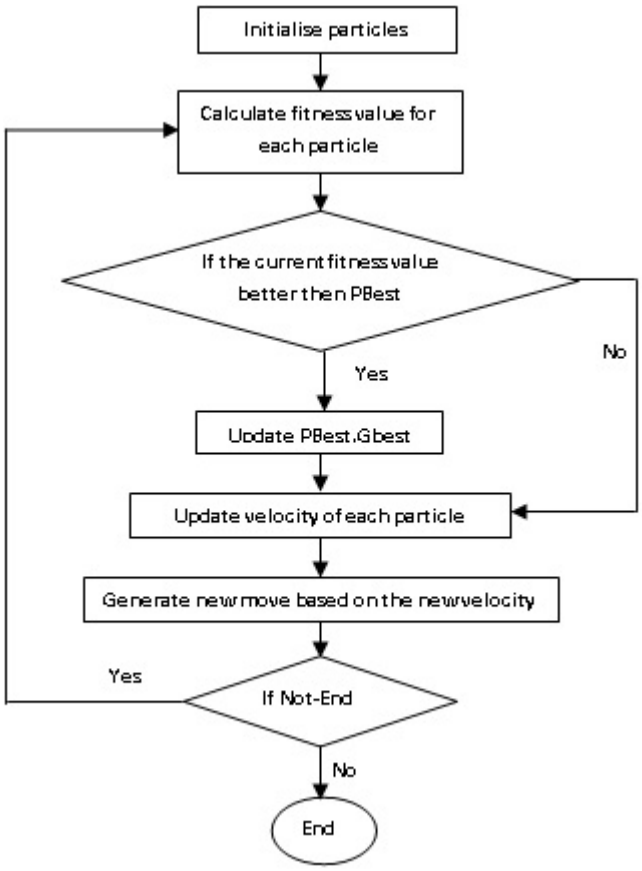

Figure 2-6: Particle Swarm Optimisation Algorithm

the recently visited solutions, the intermediate-term improves the intensification strategy to select active area. The long-term improves the diversification strategy of the algorithm to impose a good search strategy.

Tabu Search algorithm works as follows (Figure 2.7): First one initial solution is initialised randomly or by using another heuristic, after that a set of neighbour solutions is created based on the current solution, these solutions are evaluated with the objective function to choose among them to update the new position. These instructions are repeated until the termination criterion is achieved. Many applications of Tabu Search have been carried out to solve complex problems. Author in (Azevedo et al., 2013) used TS in Safety requirement in dependable system by finding the optimal allocation of different Safety integrity level to different components in a given system. Tabu Search has been used in bioinformatics, multiple sequence alignment (Riaz et al., 2004) and inferring ancestral genetic information in terms of a set of founders of a given population arises (Roli et al., 2009).

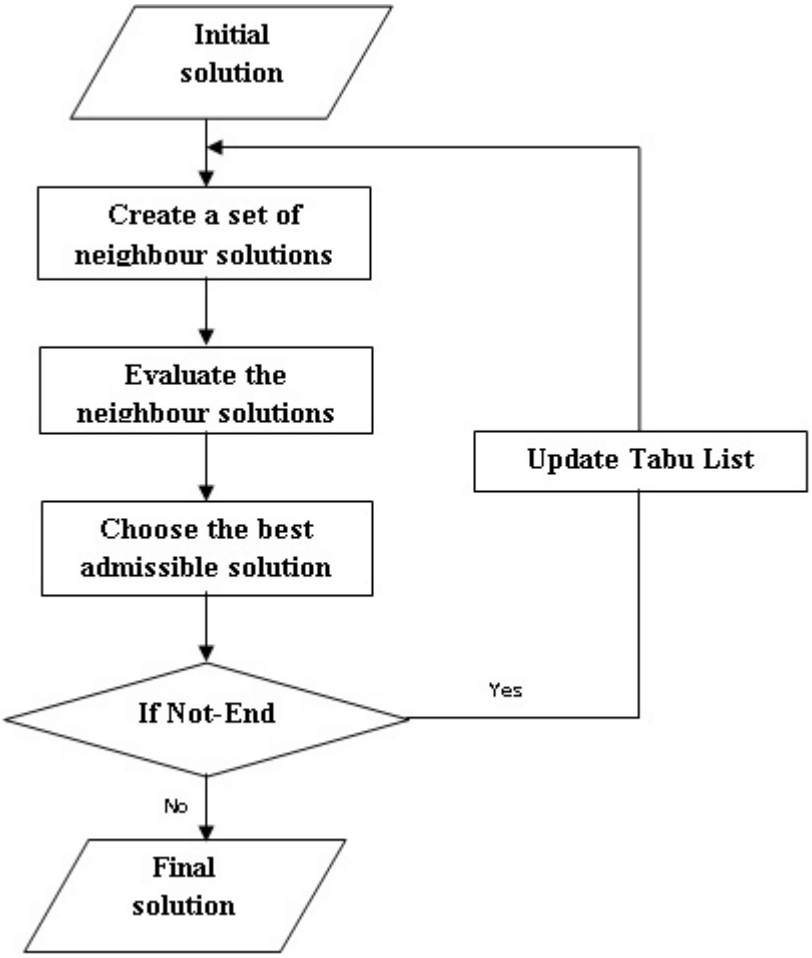

Figure 2-7: Tabu Search Algorithm

2.4.5 Discussion

Metaheuristic algorithms proved their effectiveness in different application area, after each application we confirm that there is no perfect method for all existent problems. A large number of metaheuristics have been proposed in the last years such as Cuckoo search (Yang et al., 2009), Firefly Algorithm (Yang et al 2010), etc.The table 2.1 shows the well used metaheuristics algorithms. These methods are inspired from nature, biological and collective swarm behaviour. Two major strategies can be used to compare between the metaheuristics algorithms, first is the intensification strategy aims to exploit previously-found promising regions in order to avoid local optima. Second is the diversification strategy aims to explore in an efficient way the search space to identify new trajectories that might contain the global optima. A good optimisation algorithm must combine these two properties by

certain randomisation in combination. Nature inspired metaheuristics have demonstrated success in a large number of problems and applications. However, there is always place to develop new methods inspired form nature in order to find more powerful methods for the hard problems.

Metaheuristic Algorithms	**Trajectory**	**Population**	**Memory**
GA (Genetic Algorithm)		•	
ACO (Ant Colony Optimisation)		•	•
DE (Differential Evolution)		•	
PSO (Particular Swarm Optimisation)		•	
ABC (Artificial Bee Colony)	•		
FA (Firefly Algorithm)		•	
CS (Cuckoo Search)		•	
BA (Bat Algorithm)		•	
SA (Simulated Annealing)	•		
TS (Tabu Search)	•		•

Table 2.1: Classification of different well-known nature-inspired metaheuristic algorithms

2.5 Conclusion

By looking at the different classes of problems in complex problems, it is clear that the optimisation takes a considerable part. The optimisation problems can be defined in different ways based on the nature of data, the number of the optimisation objectives, and the feasibilities of the solution (Constraint). The proposed algorithms for solving the optimisation problems prove their effectiveness in terms of times and space consuming. In this chapter we introduced some basic concepts related to combinatorial problems and optimisation, by classifying the optimisation problems and the well-known optimisation algorithms. We have shown that these algorithms find the optimal solution by manipulating two strategies: intensification and diversification, the first one is the strategy to improve the convergence of the algorithm and the second is to explore in an efficient way the space solutions.

Chapter 3

Biological Knowledge Discovery: Background Study

3.1 Introduction

The use of computer science to understand biological phenomena and analysing biological data becomes the body of any biological process. The size of biological databases augments from each day to another with very high amounts of new biological data, such as new sequenced data and a new prediction of protein structures. In this chapter, we will discuss the use of metaheuristics in bioinformatics and computational biology as well as their applications in the different tasks of bioinformatics and computational biology. We will describe after, the studied problems in this thesis by showing the hardness of the problem and general overview about the problem formulation. We present at the end of each problem the literature review for each one of them.

3.2 Metaheuristics in Bioinformatics and Computational Biology

In recent years, advances in the field of bioinformatics and genomics technology have increased significantly. Understanding biological system, proper representation of biological

data and their transmission efficiency has became a primary interest in the biological community. This area is an interdisciplinary field involving computer science, biology, physics and mathematics in order to improve the quality of results and in a faster way (Hogeweg 2011).

Most of bioinformatics and computational problems are formulated as combinatorial problems. As mentioned in chapter 2 the exact solvers don't give the optimal solution in polynomial complexity. The use of the optimisation methods such as metaheuristics is the optimal choice to handle bioinformatics hard problems (Clonis 2006). As defined in the chapter 2; metaheuristics are top level methods, aims to guide the heuristics to quickly find the optimal solution.

3.2.1 Biological Sequences Alignment

The alignment of different biological sequences (DNA and proteins) aims to find the common region in a given sequences (Polyanovsky 2011), the biological sequence alignment is divided in two categories: pairwise sequences alignment and multiple sequences alignment, these two problems are considered as combinatorial problems. The pairwise alignment aims to find the common region between two sequences by maximising the alignment score. The well-known methods for pairwise biological sequences alignments are SAGA: Sequence Alignment by Genetic Algorithm (Notredame 1996), and the particle swarm optimisation algorithm to biological pairwise sequence alignment problem (Juang 2008). The second category is the multiple sequences alignment, aims to align three or more sequences, the multiple sequences alignment is harder the pairwise alignments and take more time, some methods for solving multiple sequences alignment are based on the pairwise alignment by aligning each two together and construct the final model. Other methods use the optimisation algorithms to optimise the score of the alignment by maximising the scores of the common region.

3.2.2 The DNA Fragment Assembly (DFA)

DNA fragments assembly problem is another hot topic of bioinformatics (Pevzner et al., 2001), the objective of the DNA fragment assembly is to find the best order and orientation of a set of DNA fragments to reconstruct the original DNA sequence from them. As it has to consider all possible combinations among the DNA fragments, it is considered as a combinatorial optimisation problem (Pevzner et al., 2001).The DNA fragments assembly follows the OLC (Overlap, layout, consensus) model that is used in all currently available assembly methods (Staden 1980).

3.2.3 Protein Structures Prediction

The function of the protein has a strong relation with the structure of the proteins; two kinds of structures exist for each protein sequences, secondary and tertiary structures (Garnier1978). The first category aims to find the model of the structure by predicting the alpha helices, the beta sheets and loops of the structures, but the tertiary structure aims to predict the position of the amino acids in the space by taking into consideration the neighbourhood of this amino acid. Large numbers of methods have been proposed to solve the protein structure prediction problems, the most widely used algorithms is the use of multi-objective evolutionary approach to the protein structure prediction problem (Cutello et al., 2006). The prediction of 3D structure of proteins from its amino acids sequences become a hot challenge in bioinformatics in the last years, the most widely used methods for the 3D structure prediction are the parallel ant colony optimisation for 3d protein structure prediction using the HP lattice model systems (Chu et al., 2006).

3.2.4 Gene Prediction

Gene Finding and Identification aims to identify the regions of genomic DNA that encode genes (Besemer et al 1999). The process of understanding the genome species after sequences is the gene finding task. The metaheuristics algorithms have been applied to solve the gene prediction problem. The use of particle swarm optimisation algorithm in gene se-

lection in cancer classification, the PSO has been augmented with the SVM in this method to improve the accuracy of the prediction (Alba et al., 2007). The author in (Alba et al., 2007) proposes another hybridisation of PSO with Genetic algorithm for the gene selecting in cancer classification.

3.2.5 The Phylogenetic Reconstruction

the phylogeny is the searching of useful relationship between evolutionary species (Huelsenbeck et al., 2001). The evolution of the species is one of the challenging areas in bioinformatics the use of the optimisation algorithm to find the adaptive modification based on natural and sexual selection. The phylogenetic analysis is the estimation and the searching of these relationships. The phylogenetic inference problem has been considered as combinatorial optimisation problem, a wide use of metaheuristics to solve this problem, the simulated annealing algorithm has been used to solve this problem (Stamatakis et al., 2005), to avoid cycling of the simulated annealing algorithm, the Tabu search algorithm has been applied to the phylogenetic inferring problem, it avoids cycling records recent moves in one or more Tabu lists (Lin et al., 2005).

3.3 Similarity Search Methods in Genomic Sequences

3.3.1 Approximate Sequences Matching Problem

Homology search in biological sequences is a very important task for discovering and understanding similarities among genes and proteins, in order to find similar segments, or local alignments, between two DNA or protein sequences (Altschul et al., 1990). The sizes of DNA and protein databases became very large, such as the EMBL Nucleotide Sequence Database (EMBL-Bank) has increased in size from around 600 entries in 1982 to over 6.2×10^8 by MARCH 2015, so homology search is very time consuming and far to be done in reasonable time (Altschul et al., 1990).

Biological instances contain large biological information with different types of data such as sequences and structures. These latter are either proteins (Uniprot) or nucleic sequence

(EMBL, GenBank...). Thus, realising similarity search among these instances in a reasonable time is a difficult task. Undoubtedly, developing an efficient algorithm based on seeds alignment is a big challenge for bioinformatics community. The seed paradigms for biological sequences alignment start by finding possible seeds matches, after that the seed are extended to from possible alignments. Figure 3.1 shows the possible alignment between two biological sequences is constructed based on preliminary founded seeds. The Dot Plot shows that more than one alignment is possible, the difference between these possible alignments is the amount of matching between the sequences (Score).

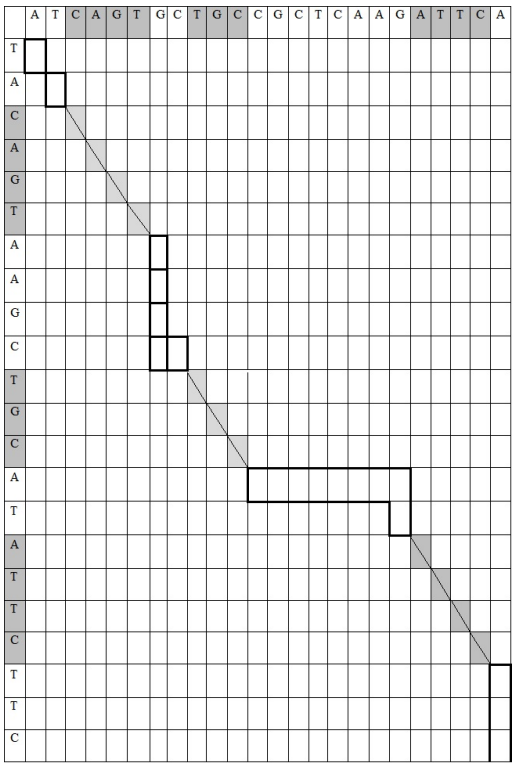

Figure 3-1: The dot plot of seed paradigm

The matching between different sequences can be continuous or spaced matching (see Figure 3.2). The use of spaced seeds performs specialised optimisation for next generation sequencing. Several methods for solving this problem have been recently proposed and can be classified in two categories. The first one employs dynamic programming and can find an exact solution with quadratic time complexity (Smith et al., 1981). As biological databases grow larger, this exact approach is usually required a high time consuming. The Second category is the heuristic algorithms (Lipman et al., 1985) which can achieve good

solutions in a reasonable time.

The alignment of two biological sequences based on seeds is an efficient technique used by several algorithms in order to produce other sequencing data generations. Among these algorithms, the 11 consecutive matches of BLAST (Altschul et al., 1990) is called a contiguous seed, denoted as 11111111111 for eleven consecutive matches. It is required to find an identical stretch of length 11 which is not always feasible. In order to increase the probability to find an alignment, PatternHunter II (Li et al., 2004) uses one or several non-contiguous seeds called spaced seeds. Concretely, each spaced seed S is a vector of n elements where n is the length of each seed and their position is defined as follows: S[i] = 1, if the position need required matching and S[i] = 0 if we do not care about position matching, the number of ones in the seed called the weight of the seed. This sensitivity is used to evaluate the quality of a spaced seed for matches alignment. The objective of spaced seed is to increase sensitivity without reducing the computation time performance. The sensitivity is approximated by matching the spaced seeds and a Bernoulli representation of the alignment.

The main two factors of any optimisation algorithm applied to sequences matching problem is computation time (searching speed) and solution quality (sensitivity). The aim of all previous methods is to design in reasonable time a good set of spaced seeds having high sensitivity. So there is a trade-off between the computation process and the sensitivity (Choi et al., 2004). Indeed, we can increase the sensitivity by decreasing the required weight of the hit, nevertheless, the decreasing of the weight of hits will increase the runtime and also increase the number of fallacious hits.

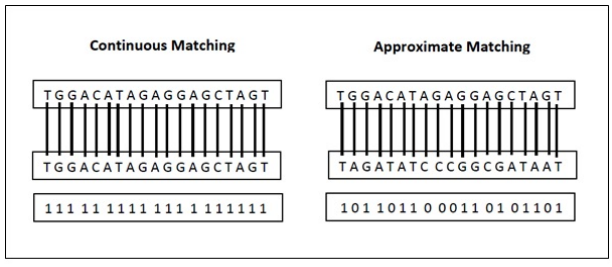

Figure 3-2: Contiguous matching Vs approximate matching

3.3.2 Previous Work in Sequences Matching

In the literature, two main categories of approaches address the sequence matching problem. The first one is a contiguous based approach which aims to search for contiguous matches on the similarity searching process (Altschul et al., 1990). Unlike the contiguous seed, the second one is a spaced seeds approach aiming to find a non-contagious seed on the similarity searching process (Li et al., 2004). The existing approaches for spaced seeds generation can be divided into three main classes:

3.3.2.1 Exact Solvers

First, the exact methods, the well-known exact method for the biological sequences matching is to use the dynamic programming to find an exact seed (Needleman 1970). The dynamic programming approaches are proving their efficiency only for finding one spaced seed. The problem of finding multiple spaced seed and evaluating these seeds with low sensitivity in reasonable time is an NP-Hard problem. The disadvantage of such dynamic based methods for multiple spaced seed is the large memory requirements for multiple dynamic programming tables and the difficulty of finding the optimal seed combination. As mentioned in (Ma et al., 2007) finding the optimal spaced seeds is an NP-Hard problem, and computing the hit probability is also an NP-hard problem. The problem of such algorithms is the memory usage and the space consuming, they provide the exact seeds matching but it takes a long time for the hard problems such as the biological data based problems.

3.3.2.2 Heuristic Solvers

The second category in sequences matching approaches is the heuristic methods, several methods based on heuristics have been designed for spaced seeds (Brown 2007). PatternHunter (Li et al., 2002), (Ma et al., 2007) is the single space based method for sequence matching problem. PatternHunter uses a novel seed model to increase the sensitivity in reasonable time. PatternHunterII, which is an improvement of PatternHunter, allows to increase the sensitivity by using multiple spaced seeds. The following algorithm present the dynamic programming algorithm for computing the hit probability (Algorithm 3.1), given

a set of binary strings (Seeds) compatible with part of 'a' included A. 0b and 1bis all seeds start with a, and b(x) is the prefix of the seed x. The Dynamic programming for multiple spaces seed find the optimal seeds by incorporating a greedy methods for choosing the optimal spaced seeds. It applies a new greedy method for finding near optimal multiple seeds as follows: firstly, it starts by computing the first seed that maximises the hit probability. Then, it fixes the previous seed and finds the second to maximise the hit probability with the first seed. This process must be repeated until the desired number of seeds or the given hit probability is reached.

Algorithm 3.1: Dynamic Programming Algorithm

Input : seed set A, similarity level p and length L
Output: The probability that A hits a p-random region of length L
Compute the compatible suffix set B
For i **from** 0 **to** L **do**
For b **in** B **from** longest **to** shortest **do**
If $i < |b|$
$f[i, b] := 0$
Else $f_0 := f[i - |b| + |b'|, 0b']$, where $0b' = B(0b)$
If A hits $1b$
Then $f_1 := 1$
Else $f_1 := f[i, 1b]$
$f[i, b] := (1 - p) \times f_0 + p \times f_1$
Return $f[L, \varepsilon]$.

Mandala is a software tool used for searching optimal seeds (Buhler et al., 2003). Indeed, Mandala proposed a new algorithm for evaluating the sensitivity of a spaced seed based on the Markov model of ungapped alignments, the problem is how to find the highest probability alignments of the set of seeds that match them, the result shows that the new model improves the sensitivity of sequence matching. Furthermore, this tool maximises the impact of seed design through the Markov model to find the best seeds representing similarities. Iedera is a program that computes the seed sensitivity, selects, designs and vectorises subset seed patterns (Kucherov et al., 2006). It uses a finite automata to represent the seed and is adapted to any model that can be represented by a non-deterministic probabilistic automaton (HMM equivalence).

In (Brona et al., 2005), authors proposed a new representation of matches in seed, called

vector seeds. It is a generalisation of the spaced seeds used in PatternHunter, the mismatching seeds of BLAT, and the minimum word score seeds used by BLASTP. The difference between vector seed and simple spaced seed is that the simple spaced seed uses a binary encoding only, but the vector seed can belong to all decimal values. Authors also proposed an algorithm based on vector seed to estimate the specificity and sensitivity that might be obtained from a given seed.

3.3.2.3 Metaheuristic Solvers

The third category for optimal finding of multiple spaced seed is the metaheuristic based methods; two recent works based on metaheuristic approaches introduced the space seed problem. Speed is a new method for finding the optimal multiple spaced seed based on the hill climbing algorithm. The Speed approach solves the problem of finding only one length seeds by computing minimum and maximum sees lengths. Experimental results have proved a high correlation between sensitivity and overlap complexity, this later has been used to evaluate the multiple spaced seed (Ilie et al., 2011). Hill climbing algorithm has been improved by FastHC (Ilie et al., 2012), in order to increase the sensitivity and to employ a new way for the overlap complexity estimator. The FastHC provided a much faster implementation of the standard hill climbing meta-heuristic.
FastHC using a new way to compute the sensitivity of the set of the spaced seed by developing the FASTOC, this later compute the overlap complexity as we described later that there is a strong correlation between the sensitivity of a set of seeds and the overlap complexity (see Figure 3.3). The FASTOC represent the seed as binary string (1*1**1 converted to 101001) each seed is represented in 64 bits integers, FASTOC compute the overlap between each two seeds by shifting the bits of the seeds and AND-ing the seeds integers (Algorithm 3.2).

Fast hill Climbing algorithm starts by constructing two matrices OM and OCM (see Figure 3.4), the first one represents as the similarity between each two seeds and the OCM matrix represent the overlap complexity for each two matrices. After that the FASTHC improves iteratively the quality of the set of spaced seed by swapping between 1 and 0, the objective function of the FASTHC is minimising the overall complexity.

Algorithm 3.2: The pseudocode of FastOC

Input : a seed s of length l
Output: OC(s)

```
1  oc ⟵ 0
2  b ⟵ 1
3  for i from 1 to l - 1 do
4  if (s[i] = 0) then b ⟵ b ≪ 1
5  else b ⟵ (b ≪ 1) | 1
6  b2 ⟵ b
7  for i from 1 to l - 1 do
8  b2 ⟵ b2 ≫ 1
9  b3 ⟵ b2 & b
10 σ ⟵ 1
11 for j from 1 to ⌈l/8⌉ do
12 σ ⟵ σ ≪ onesInBytes[b3 & 255]
13 b3 ⟵ b3 ≫ 8
14 oc ⟵ oc + σ
15 return(oc)
```

Consequently, the usefulness of metaheuristics has been used to reduce the computation time of the existing algorithms. AcoSeed is an ACO-based approach for tackling the problem of finding spaced seeds for biological sequence searching (Dong et al., 2012). AcoSeed uses a construction graph which contains a set of rectangles, and each rectangle represents a seed. Each ant builds k seeds by travelling on each rectangle either up or right according to quantities of pheromone. The optimal path represents the path with low overlap complexity that represents the optimal spaced seed.

3.4 Biological Data Compression

3.4.1 Optimisation of Biological Data Compression

If we attempt to transfer big files, e.g. DNA sequences, over a serial transmission link then it would take a significant amount of time. However, we cannot overlook this problem because at present parallel processing is widely used to increase throughput and in parallel processing architecture, processing units are usually distributed in different physical locations and task sharing is a must in such architecture.

Algorithm 3.3: FastHC(S)

Input : a multiple seed $S = \{s_1, s_2 ... s_k\}$
Output: modified S with very low OC(S)

Compute OM_{ij} ,σ_{ij}, for all i, j, and OCM
bestOC $\longleftarrow$ curOC $\longleftarrow$ $\sum_{i,j} OCM[i][j]$
repeat until bestOC **stops** decreasing
for q **from** 1 **to** k **do**
for ((i, j)$\in [1..k]^2$, $s_q[i] = 1, s_q[j] = *$) **do**
$oldOC \longleftarrow curOC$
for r **from** 1 **to** k **do**
$\sigma \longleftarrow \sigma_{rq}$
UpdateSigma (OM, σ, r, q, i, j)
curOC$\longleftarrow$ curOC + OCSigma(σ)- OCM[r][q]
if (curOC < bestOC) **then**
$(q_{best}, i_{best}, j_{best}) \longleftarrow$ (q, i, j)
bestOC$\longleftarrow$ curOC
curOC $\longleftarrow$oldOC
for r **from** 1 **to** k **do**
UpdateSigma (OM, σ_{rq}, r, q_{best},i_{best}, j_{best})
$(s_{abest}[i_{best}], s_{qbest}[j_{best}]) \longleftarrow$(*, 1)
for r **from** 1 **to** k **do**
UpdateOM (OM, r,$q_{best}, i_{best}, j_{best}$)
for r **from** 1 **to** k **do**
OCM[r][q_{best}] $\longleftarrow$ OCSigma(σ_r, q_{best})
return(S)

In recent years, application of power efficient system, e.g. biological data sequencers and sequences matching systems, has increased significantly. Proper representation of digital data and their transmission efficiency has became a primary concern for the digital community because it affects the performance, reliability, and the cost of computation in both portable and non-portable devices. CMOS technologies were developed in order to reduce the power consumption both in data processing and transmission. In order to increase transmission speed and reduce transmission cost, parallel data transmission methods are widely used. However, parallel transmission is limited to short distance communications, e.g. locally connected devices, internal buses. Ruling out the possible availability of parallel transmission links over long distance, we are left with its serial alternative only. Data encoding techniques came into being to improve the data transmission efficiency over the serial communication medium by compressing data before transmitting. Efficiency can be

measured in terms of incurred cost, required storage space, consumed power, time spent and so forth. Data must be encoded to meet a variety of purposes, including: unambiguous retrieval of information, efficient storage, efficient transmission and etc. Let a message consist of sequences of characters taken from an alphabet $\sum$ where $\delta_1, \delta_2 ... \delta_r$ are the elements that represent the characters in the source $\sum$. The length of δ_i represents its cost or transmission time, i.e., cost (δ_i) = length (δ_i). A codeword w_i is a string of characters in $\sum$, i.e., $w_i \in \sum$+. If a codeword is w_i = $\delta_{i1}, \delta_{i2} ... \delta_{in}$ then the length or cost of the codeword is the sum of the lengths of its constituent elements:

$$Cost(w_i) = \sum_{j=1}^{n} Cost(ij) \tag{3.1}$$

If all the elements of a codeword have unit cost or length then the cost of the codeword is equivalent to the length of the codeword. However, it is not necessary for the elements in the codeword to have equal length or cost. For example, in Morse Code all the ASCII characters are encoded as sequence of dots (•) and dashes (−) where a dash is three times longer than a dot in duration (Redmond, 1964). However, the Morse code scheme suffers from the prefix problem (Grunwald and Vitany, 2000) (see Figure 3.5). Ignoring the prefix problem, Morse code results in a tremendous savings of bits over ASCII representation. Using Morse code, we can treat the binary bits differently; 0 as a dot and 1 as a dash. Even if we consider the voltage level to represent the binary digits then they are still different.

A	.-	J	.---	S	...
B	-...	K	-.-	T	-
C	-.-.	L	.-..	U	..-
D	-..	M	--	V	...-
E	.	N	-.	W	.--
F	..-.	O	---	X	-..-
G	--.	P	.--.	Y	-.--
H	....	Q	--.-	Z	--..
I	..	R	.-.		

Figure 3-3: Morse code for English alphabet

As the unequal letter cost problem is not new, it has therefore been addressed by different researchers. The more general case where the costs of the letters as well as the probabilities of the words are arbitrarily specified was considered by Karp (1961). A number of other researchers have focused on uniform sources and developed algorithms for unequal letter costs encoding (Gilbert, 1995; Krause, 1962; Varn, 1971; Altenkamp et al., 1980; Perl et al., 1975).Let $p_1, p_2...p_n$ be the probabilities with which the source symbols occur in a message and the codewords representing the source symbols are $w_1, w_2...w_n$ then the cost of the code w is:

$$C(w) = \sum_{i=1}^{n} Cost(w_i).p_i \tag{3.2}$$

The aim of producing an optimal code with unequal letter cost is to find a codeword W that consists of n prefix code letters each with minimum cost ci that produces the overall minimum cost $C(w)$, given that costs $0 < c_1 \leq c_2 \leq ... \leq c_n$, and probabilities $p_1 \geq p_2 \geq ... \geq n > 0$.

Huffman code (Huffman, 1952) is an efficient data compression scheme that takes into account the probabilities at which different quantisation levels are likely to occur and results in fewer data bits on average. It is widely used to compress biological data, however, all the techniques use the classical form of the Huffman code where bits are treated equally. Out of many variations of the Huffman code where cost of bits are treated unequally, the most recent approach is described by Kabir et al. (2014). This approach treats binary bit 0 as a dot ($\bullet$) and 1 as a dash ($-$) like Morse code and reduces the transmission cost (time) significantly. Like other variations of the cost considering Huffman code, the compression performance (in terms of number of bits require to encode a message) of this approach is not better than the classical Huffman code. This approach only considers cost reduction but ignores bit reduction, and therefore number of total bits is rather high.

3.4.2 Previous Work in Optimised Biological Data Compression

The size of biological data including DNA sequences increase with an ever expanding rate and will be bigger and bigger in the future. These biological data are stored in biology

databases. The exponential growth of these databases is becoming a big problem for all biological data processing methods (Howe et al., 2008). Different operations are applied to these data such as searching (Valentin et al., 2010), e-mail attachment (Scott et al., 2009), alignment (Chenna et al., 2003), and transmission on distributed computing platforms (Tzu-Hao et al., 2014). Interestingly, biological data compression can play a key role in all sorts of biological data processing.

A recent deluge of interest in the development of new tools for biological data processing requires efficient methods for data compression. The main objective of data compression methods is minimising the number of bits in the data representation. Brandon et al. (2009) proposed a new general data structure and data encoding approach for the efficient storage of genomic data. This method encodes only the differences between a genome sequence and a reference sequence. For encoding, the method uses different fixed-length encoding schemes such as Golomb (Golomb, 1996), Elias codes (Elias, 1975) and variable length codes such as Huffman codes.

There are other methods based on the same idea of encoding only the difference between reference sequence and the target one. One such approach (Scott et al., 2009) uses Huffman code for encoding differences between sequences to send as an email attachment. The main limitation of this method is that it must send the reference sequence at least once for each species, and usually this sequence is too big to be sent as an email attachment.

(Wang et al., 2011) proposed a new scheme for referential compression of genomes based on the chromosome level. The algorithm searches for longest common subsequence between two sequences and then the differences between them are encoded using Huffman code. All previous studies focus only on the differences and the relation between continuations of the sequence, and use existing encoding approaches to encode biological datasets without considering possible improvement of the encoding schemes.

3.5 Optimisation of Original DNA Construction

3.5.1 DNA Fragment Assembly

Deoxyribonucleic acid (DNA) is a nucleic acid that contains the genetic information in all organisms including humans. The DNA is represented as a long file made up of four chemical bases: adenine (A), guanine (G), cytosine (C), and thymine (T) (see Figure 3.6). These bases are 99 percent similar for all humans and usually organisms of the same classes have high similarity in their DNA (Berk et al., 2000). A genome is a stretch of DNA that encodes a polypeptide (protein), which is a set of amino acids bound together in a specific order. The chemical bases are up with each other in the way that base A is always paired with base T, and base C is always with base G. This sequence consists of nucleotides bound together, which are interpreted by the cellular machinery in groups of three, called triplets (Cai et al., 2012).

Figure 3-4: DNA structure

DNA fragment assembly is a NP-hard problem in computational biology (Pevzner et al., 2001) (see Figure 3.7). The human DNA contains about 3 billion bases and cannot be read at once. Experimental techniques have been used to solve this problem by cutting the whole sequence at random positions to a set of fragments. The main objective of DNA fragment assembly is to construct the original DNA sequence from the small fragments by finding the best order of these fragments to maximise the overlapping scores between each two consecutive fragments (Bankevich et al., 2012).

The experimental tools for biological phenomena analysing and prediction is very expensive and can take long time to find a useful hidden knowledge. Bioinformatics tasks are

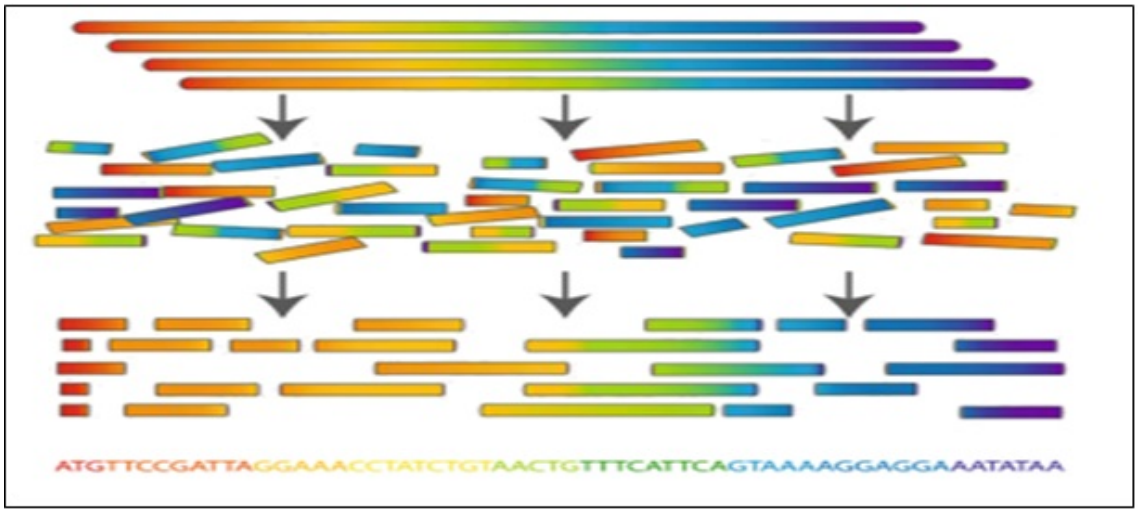

Figure 3-5: DFA problem

very interesting approaches for solving these kinds of problems. The number of possible arrangement of DNA fragment proves that the problem is an NP-hard problem as demonstrated in (Pevzner et al., 2001). Exact and approximate solvers can be proposed to solve the DNA fragment assembly problem.

Several optimisation-based methods have been proposed to deal with the DFA problem (Pevzner et al, 2001; Caserta et al,. 2014; Chang et al,. 2011; Bocicor et al,. 2011). The main objective of these methods is to construct the original DNA sequence based on the small DNA fragments by maximising the overlapping scores between consecutive sequences. Greedy methods were firstly applied to deal with the DNA fragments assembly problem by Staden (1980), these kinds of methods are naive and can be used only for small problem. Most of biological data based problems use a very high amount of data which is the nature of such problems. Afterwards, optimisation algorithms and specially metaheuristics were widely used to solve the DNA fragments assembly problem. The formulation of DNA fragment assembly problem can be the simple way by using the fragment of DNA as strings and try to arrange these set of strings, we can go to more complicated form but powerful representation by a complete graphs, and the problem of finding the DNA fragments assembly problem can be seen as path finder by finding a Hamiltonian cycle for the graph.

3.5.2 Previous Work in DNA Fragment Assembly

Many approaches have been applied in DNA fragments assembly to improve the quality of the solutions for this problem. Greedy methods were the first work on DNA assembly problem, these kinds of methods can be easily implemented. The well-known and the most used greedy method in DNA fragment assembly is proposed by (Staden 1980) (see Figure 3.8). These methods used the overlap-layout-consensus, this technique for the large-scale sequencing problems is not as good as the small problem.

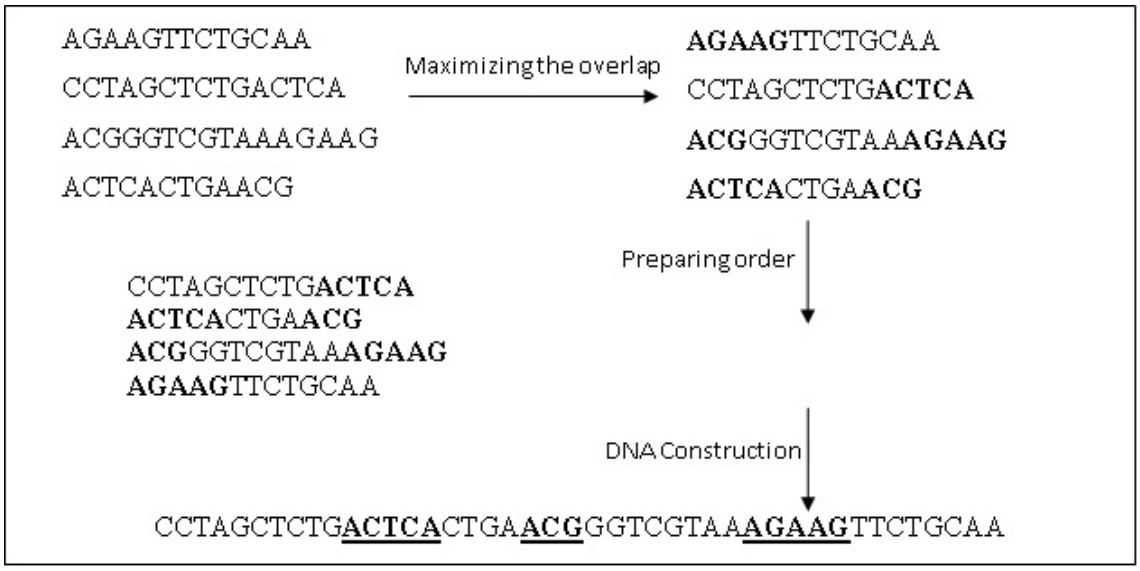

Figure 3-6: OLC for DNA fragment assembly problem

3.5.2.1 Genetic Algorithm for DNA Fragment Assembly

Genetic algorithm has been widely used to solve the DNA fragment assembly. Kim et al. (2013) proposed a new parallel hierarchical adaptive variation of evolutionary algorithms. Their approach incorporates a new measure (objective function) to evaluate the quality of a given solution. It implements a new variation of the island model of parallel GA, called 'Parallel Hierarchical Adaptive GA' (PHAGA). The solution is built with hierarchical model constructed by identifying subsequences with high confidence.

A grid-based genetic algorithm has been proposed by (Nebro et al. (2008). The approach uses a grid composed of up to 150 computers to solve large scale problem in a reason-

able time. It uses the master/slave model, using multiple slaves in parallel with one master population. An improved GA algorithm has been proposed by using two main operators Reduction and Refinement of the chromosomes to improve the speed and the quality of the results (Kikuchi et al., 2006). The reduction step aims at forming an efficient contiguous array of genes in the chromosome, and the refinement step is added to efficiently support the genes in the chromosome. This refinement of greed's mutation takes place in every set of iteration fixed as an initial parameter. The proposed approach follows the OLC model, it starts by finding the longest similarity between each two fragments, after that the optimal order is found by computing the score of a given full order, the last step is the consensus, for DNA fragment assembly problem the consensus construction is similar to that of the travelling salesman problem.

3.5.2.2 Swarm Intelligence for DNA Fragment Assembly

Swarm intelligence algorithms have been applied to solve the DNA assembly problem. Particle swarm optimisation algorithm (PSO) has been proven as a powerful nature inspired metaheuristics algorithm to solve problems in different areas (Kennedy et al., 1998). Huang et al. (2014) proposed a memetic PSO algorithm based on the tabu search (TS) (Glover et al., 1999). In this algorithm, simulated annealing (Aarts et al., 1988) is used as the initialisation method and two variable neighbourhood search (VNS) (Hansen et al,. 2001) methods are used to improve the intensification strategy. The VNS local search algorithm uses multiple operators such as swap between fragments, insertion, inversion and randomly displacement of substructures. The PSO is used to achieve the global best solution based on these methods. The limitation of this method is that it consumes high runtime by incorporating several methods.

Ant Colony System Algorithm (Dorigo et al., 1997) is a powerful nature inspired metaheuristics algorithm based on ant behaviour. It has been used also to solve the DNA fragment assembly problem. Meksangsouy et al. (2003) used Ant colony optimisation (ACO) to solve both single- and multiple-contiguous problems by using an asymmetric ordering representation. Ants cooperate their efforts to generate the solution path that has the maximum overlap score between each pair of DNA fragment. Another application of ACO

to DNA fragment assembly is by modelling the problem as path-finding problem (Yan et al., 2011). It considers multiple objective functions, similarity measure, continuity measure and hairpin with two constraints, which are the melting temperature and the guanine-cytosine content (uniform chemical characteristics). An integer programming algorithm has been proposed to solve DNA fragment assembly to find the global optimal solution for the problem. The approach has been applied to DNA data sets with errors, it incorporates a new efficient technique to identify alternative reconstruct. The algorithm has been applied to spectrum data of real human DNA (Chang et al., 2011).

Recently, several nature inspired metaheuristics algorithms have been proposed such as firefly algorithm (Yang et al., 2010), which has been applied later to DNA fragment assembly problem by formulating the score measure as the sensation distance of each firefly in the solution space (Ezzeddine et al., 2014). The Firefly for DNA fragment assembly algorithm starts by initialising parameters such as the number of allowed moves, the number of iterations and the light absorption gamma, after that the initial population is randomly generated. Each firefly tries to find the most attractive among the rest of the population, in the worst case it moves randomly, update their parameters and repeat this proves until no move is available

3.6 Conclusion

In the recent years the bioinformatics and computational biology have received a great attention due to the amount of biological data sequenced every year. The combinatorics nature of the most of bioinformatics and computational biology problems gives a good application field for the optimisation algorithm such as heuristics and metaheuristics. This chapter provided a general presentation of the use of metaheuristics in bioinformatics and computational biology problems. We described in this chapter the studied problems in this thesis, by showing for each algorithm the proposed algorithms in the literatures for each kind of problem. In the following chapters, we will present the contributions and the work achieved in this thesis. The mathematical formulation of each problem and the proposed approach, at the end, we present the experimental phases of each proposed algorithm.

Part III

Contributions

Chapter 4

PeSOA for Finding Optimal Spaced Seeds

4.1 Introduction

Our thesis as mentioned in the chapter 1 is about the using the metaheuristics algorithms for biological data based problems. The size of biological data augments every day, algorithm for biological data analysis needs to be as a good as finding the optimal solution in reasonable time. In this chapter we investigate the problem of similarity searching in biological data by handling the problem of finding multiple spaced seeds. The proposed approach is based on penguins search optimisation algorithm (PESOA), the penguins search algorithm is population based algorithm developed in (Gheraibia et al., 2013). It's a nature inspired metaheuristic algorithm based on the collaborative hunting behaviour of penguins. The work related to the current chapter is published in an international journal (Gheraibia et al., 2015a). The rest of the chapter is organised as follow: Section 4.2 we discuss the general problem of spaced seeds, section 4.3 the PeSOA algorithm is presented. Then, section 4.3 defines the proposed approach, section 4.4 describes the experiments and comparison study. We conclude the chapter in section 4.5.

4.2 Spaced Seed Problem

Any alignment between two DNA or protein sequences can be represented by a Bernoulli model representation as a binary string with 1 and 0, where 1 stands for a match and 0 stands for a mismatch. Seeds technology is an important filtration method used to speed up local alignment. The quality of a seed can be measured by its sensitivity, which is a measure of ability to detect similar segments between two sequences.

Spaced seed is a regular expression of binary strings. Moreover, it includes zeros and ones with starting and finishing by one. Let α be a seed, $|\alpha|$ denotes the length of the seed (i.e., the number of zeros and ones) and $\|\alpha\|$ denotes the weight of the seed (the number of the ones only) (Altschul et al., 1997). As an example, let R=1001101011 be a Bernoulli representation of biological sequence, so $|R|$= 10 and $\|R\|$=6. Spaced seed is used in the filtration stage of sequence comparison process in order to find appropriate matching. During the search process, certain positions are ignored which are represented by 0 in the seed and by 1 in the position of required matches. Each homology research program uses a specified seed, for example, the Blastn uses 11111111111, or 11 consecutive matches to generate a hit. PaternHunter default seed 111010010100110111, it has a weight of 1 and length 18.

Let R be a random binary sequence of length N, used to represent a sequence alignment with a matching probability P, let S be a spaced seed, the match represents similar positions on the two sequences. We say S hits R if we find at least one substring from position i up to the position j-i+1=$|S|$ denoted by R [i : (j-i+1)] match with S. Additionally, we say a set of K spaced seeds hit R if at least one of these hits is R. The probability that s (or S) hits R is called the sensitivity. Actually, this sensitivity depends on the length of the binary region to be hit and on the distribution of matches. Accordingly, computing the hit probability (the sensitivity) for a given (k>1) seeds under the uniform distribution is considered as an NP-hard problem (Li et al., 2004).

The number of the possible sets is exponential, so determining the optimal set of seeds that maximizes the hit to R is clearly a difficult task. Finding optimal space seeds and computing hit probability is a NP-hard problem (Li et al., 2002). The objective of such problem

is to find a set of K seeds of weight w that maximize the hit of matching sequence R, the matching probabilities p is related to the length of the sequences N (Ma et al., 2007). For instance with N=10000; K=25 and p=0.80, we obtain 33×10^8 potential solutions, consequently, the bio-inspired approaches are appreciated to handle this kind of problem.
In the next section, a recent bio-inspired approach penguins search optimisation (PeSOA) will be introduced. PeSOA is a bio-inspired algorithm ensures high diversification; it represents the individuals within a population. In this context, it is desirable to have a good diversification in our population in order to explore the search space properly. Also, when we get a high diversification we intensify the exploration; otherwise we intensify the local search.

4.3 Penguins Search Optimisation Algorithm (PeSOA)

Bio-inspired approaches have been revealed their efficiency in data analytics especially in computational biology (Rumble et al., 2009), (Julio et al., 2008), (Handl et al., 2007). The bio-inspired approaches can be subdivided into two categories. The first category talks about the evolutionary algorithms; it starts with an initial population, and then applies some evolutionary operators to produce another population which is more improved than the last one. The most used evolutionary algorithms are genetic algorithms. The second one is the swarm intelligence, the emerging field in computational intelligence; the solution space is intelligently explored based on the cooperation and the synergy of existing swarms. One of the recent swarm's methods is Penguins search optimisation algorithm.
The optimality theory of foraging behaviour was modeled in the works of (MacArthur et al., 1966), (Mori et al., 2002). These two studies hypothesized that dietary behaviour may be explained by economic reasoning: when the gain of energy is greater than the expenditure required to obtain this gain, so it comes to a profitable food search activity. Penguins, as biological beings, use this assumption to extract information about the time and cost of food searches and energy content of prey, on one hand, and the choice to hunt or not in the selected area, depending on its high resource and the distance between feeding areas, on the other. The behaviour of air-breathing aquatic predators, including penguins was

noticed by (Houston et al., 1985). The surface is a place for penguins as they are forced to return after each foraging trip. A trip implies immersion in apnea. The duration of a trip is limited by the oxygen reserves of penguins, and the speed at which they use it, that is to say their metabolism (Green et al., 1998), (Tremblay et al., 1999). The works in the field of animal behavioural ecology of penguins have given us clear and motivating ideas for the development of a new optimisation method based on the behaviour of penguins.

4.3.1 Metaphor: Hunting Behavior of Penguins

Penguins are sea birds, unable to fly because of their adaptation to aquatic life (Green et al., 1998), (Hanuise et al., 2010). The wings are ideal for swimming and can be considered as fins: penguins fly through water and can dive more than 520m to search for food. Although this is more efficient and less tiring to swim underwater than at the surface, they must regularly return to the surface to breath. They are able to keep breathing while swimming rapidly (7 to 10 km/h) (Green et al., 1998). During the dives, the penguin's heart rate slows down. Underwater, the haunting eyes of the penguin are wide open; his cornea is protected by a nictitating membrane. The retina allows him to distinguish shapes and colors.
Penguins feed on fish and squid. For this, they must hunt in groups and synchronize their dives to optimise the foraging (Takahashi et al., 2004). Penguins communicate with each other with vocalisations. These vocalisations are unique to each penguin (like fingerprints in humans). Therefore, they allow the unique identification of each penguin and the recognition of penguins to each other (Simpson et al., 1976). This factor of identification and recognition is important since there is a large size of the colonies and a great similarity of the penguins. The amount of the necessary food for a penguin is variable depending on species, age, variety and quantity of food available in each region. Studies had shown that a colony of 5 million of penguins may eat daily 8 million pounds of krill and small fish (Simpson et al., 1976).

4.3.2 The PeSOA Algorithm

Penguins search optimisation algorithm (PeSOA) is a bio-inspired meta-heuristics based on collaborative hunting strategy of penguins. Penguins swarms have several collaborative characteristics, they collaborate their efforts and synchronise their dives to save the global expenditure of energy in food searching process. In the hunting process, penguin population is divided into groups and each group is composed of a variable number of penguins depending on food availability in a specific location. In each group, each penguin searches separately for food according to their oxygen reserve (updated according to the objective function). After each food-foraging iteration, penguins return back on the surface (ice) to share with affiliates, the location of food (this rule ensures intra-group communication). If the number of fishes (calculated according to the objective function) in a specific location is not enough (or none) for a given group, part of the group (or the whole group) migrates to another hole (this rule ensures inter-group communication). The pseudo code of the penguin search optimisation algorithm (PeSOA) is shown in Algorithm 4.1.

Algorithm 4.1: PeSOA Algorithm

Generate K regions in the solution space based on distances
Generate penguins **for** each group i
while stopping criterion is not reached do
Initialise the oxygen reserve for each penguin
For each group i **do**
For each penguin j **in** this group **do**
Improve the penguin position by
$x_j^i(t+1) = x_j^i(t) + O_j^i(t) \times rand() \times (x^i_{LocalBest} - x_j^i(t))$
End
Update the food abundance degree for this group by
$O_j^i(t+1) = O_j^i(t) + (f(x_j^i(t+1)) - f(x_j^i(t))) \times \|x_j^i(t+1) - x_j^i(t)\|$
End
Update the global best solution
Update membership function values for each group
Redistribute penguins to groups according to the membership function
Abandon the group if it has no members
Endwhile
End

To summarize the observations from penguins' foraging behavior, the following rules

are presented.

Rule 1: A penguin population comprises of several groups. Each group contains a number of penguins that varies depending on food availability in the corresponding foraging region.

Rule 2: Each group of penguins starts foraging in a specific depth under the water according to the information about the energy gain and the cost to obtain it.

Rule 3: They feed as a team and follow their local guide which has fed on most food in the last dive. Penguins scan the water for food until their oxygen reserves are depleted.

Rule 4: After a number of dives, penguins return on surface to share with its local affiliates, via intra-group communication, the locations and abundance of food sources.

Rule 5: If the food support is less for the penguins of a given group to live on, part of the group (or the whole group) migrates to another place via inter-group communication.

4.4 The Proposed Approach

4.4.1 Encoding

A compact and efficient solutions encoding is used in the proposed seeds design. In PeSeed, two values are used for standards spaced seed (0,1) for interpreting the signification of similarities on the seed, where 1 means match between seed and the Bernoulli model, and 0 otherwise. For the bio-inspired algorithm we also have used an occurrence table to guide the search and which reduces significantly the search space.

Sequence 1: T A T T A C T A A C

Sequence 2: T T A T A A T A T C

4.4.2 QUICKLYOC: a Heuristic for Fitness Computing

The sensitivity of a given seed represents the matching probability that seed set hit an alignment. The dynamic programming algorithm has been used to compute the exact hit probability of a single seed. Since this problem of computing the sensitivity of K given seeds under the uniform distribution is considered as an NP-hard problem, the overlap complexity (OC) is a polynomial time heuristic giving an approximate solution of this

problem, the OC is independent of the similarity level p.

In (Ilie et al., 2011) an experimental correlation between the sensitivity and the overlap complexity has been shown (Seeds with low overlap complexity have high sensitivity). The overlap complexity can be computed in several ways: given two seeds S_1 and S_2, let σ_i be the number of pairs of ones aligned together when a copy of S_2 is shifted by i positions while aligned against S_1 while $S_1.Wight$ represents the index of ones position in the seed. The overlap complexity is to capture the overlap between seeds. It is very susceptible for the 1's position in the given seed. For two given seeds S_1 and S_2, we have $|S_1| + |S_2| - 1$ possibility of overlaps between them, we denote the number of 1's in similar position for each i shift time by σ_i, and the overlap complexity OC computed as follows:

$$OC = OC + \sum_{i=|S_2|}^{|S_1|-1} 2^{\sigma_i} \tag{4.1}$$

Each seed has an array called weight which represents the 1'position on the seed, the weight array is represented as follow:

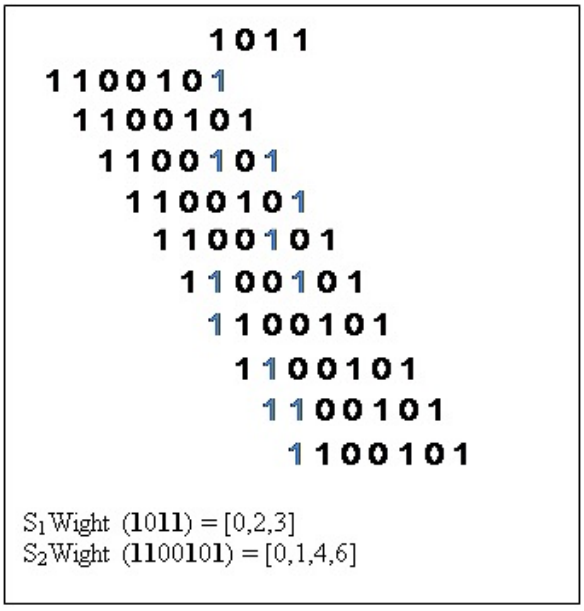

Figure 4-1: QuicklyOC technique

In this algorithm, the problem is to run through the 0 mismatch positions which have no effect on the final solution. The QUICKLYOC (Quickly Overlap Complexity) (Algorithm

Algorithm 4.2: QuicklyOC (S_1, S_2)

Input : a two seed S_1, S_2 of weights w_1, w_2
Output: OC (S_1, S_2)
Compute $S_1.wight$, $S2_2wight$
For i **from** 1 **to** w_1 **do**
For j **from** 1 **to** w_2 **do**
$\sigma[S_1.wight[i] + S2_2wight[j]] + +$
EndFor
EndFor
$OC = \sum_{i=|S_2|}^{|S_2|-1} 2^{\sigma_i}$
Retrun (OC)

4.2) is a new way to compute the OC without travel on these positions; two arrays (weight) are constructed and each one contains the 1's index of the seed, so the length of the array becomes the weight (number of ones) of each seed.

4.4.3 Pe-Seed: Penguins Search Optimisation Algorithm for Spaced Seeds

Our goal is to compute highly sensitive multiple spaced seed. Therefore, we apply the penguins search optimisation algorithm with a new definition of the table of frequently occurring pattern to improve the runtime and to reduce as much as possible the overlap complexity (sensitivity) between seeds. The table of frequently occurring pattern is constructed from the seeds. For each seed in the initial population, we compute the number of occurrences for a frequent sub-string (pattern) of length 4. For each set of seeds the table contains all possibilities of sub-strings of weight w=3 or w=2 and with fixed length L=4. Assume a set of seeds S = $\{S_1, S_2, ..., S_k\}$ each seed has a weight w and denote the length of Si by L_i , for $L \leq i \leq k$. We must construct for S a table of frequently occurring pattern; the value of an entry in the table is obtained by counting the number of occurrence for this sub-string in all seeds, and also for each entry of the table we compute for each sub-string the occurrence of the sub-string of the same model. We include with same model the sub-strings that have a subset of 1 occurring at exactly the same position from the set of 1 of the original pattern. Table 1 shows an example of an OCT for the following set of seeds:

S = $\{1011010110101; 1110111000101; 1001110101101\}$

L	W	Pattern	Same model	Occurrence on S
4	2	0011	0011,1011,0111	7
4	2	0101	0101,1101,0111	8
4	2	0110	0110,1110,0111	8
4	2	1001	1001,1101,1011	9
4	2	1010	1010,1110,1011	10
4	2	1100	1100,1101,1110	8
4	3	0111	0111	2
4	3	1011	1011	4
4	3	1101	1101	4
4	3	1110	1110	3

Table 4.1: The table of frequently occurring pattern

The Pe-Seed works as follow (Algorithm 4.3): Firstly, the table of frequently occurring pattern (OCT) as shown previously is computed according to the definition. The OCT is used with the penguins search optimisation algorithm for the optimisation process. Secondly, we distribute the penguins population on different seeds. Each penguin (j=1 ... m) represent a candidate solution with active seed (several penguins may dock on the same seed so penguins are distributed in groups). In this process, penguins are sorted in order to their groups and start searching in a specific seed according to the probability of food availability (food availability computed from the amelioration of the objective function). The computation of solution update is repeated for each penguin in each group as follows. Each penguin chooses a deep point on the present seed and chooses a sub-string of length 4, the function FindPosition that gives the number of occurrence in the table of this substring, and after several iterations controlled with the oxygen reserve, penguins communicate to each other the best solution which is determined by the quantities of eaten fish (objective function), and we calculate the new distribution probability.

The main objective of the Pe-SeeD is to find the optimal composition of k seeds (S = $\{S_1$,S_2,..., $S_k\}$), precisely is to minimise the similar region in seeds and thus reducing the power number (σ_i) of the overlap complexity. Each penguin does a local search in its active seed and try to improve the overlap complexity of the seed with the table of frequently occurring pattern. If the selected sub-string is present frequently on other seeds, the penguin

changes this sub-string with other sub-string of the same weight. The choice of the new sub-string is also based on the table of frequently occurring pattern (choose the sub-string that is not frequently present in other seed). Also in the present iteration, penguin search is guided with the oxygen reserve to continue the search in the present seed, the reserve of oxygen is changed according to the amelioration of the overlap complexity (objective function). When all penguins achieve their best solution, we choose the best solution for each seed (penguin). Penguins communicate to each other the best solution, which is determined by the number of eaten fish. We compute also the best solution for all seeds and the probability of penguin as reported by the overlap complexity amelioration, and redistribute penguins for the next iteration according to these probabilities.

Algorithm 4.3: Pe-Seed(S)

Input : a multiple seeds S = $S_1, S_2...S_k$
Output: modified S with very low OC(S)
Compute the table of frequently occurring pattern OCT
Compute BestOC(S)
Initialise the population of P penguins (j = 1...m) with S
Initialise: P_{Si}, O_{pj}, Q_{fj}
Repeat until BestOC **stops** decreasing
For each penguins P_j do
Chose a random position $R_p$on its seed S_i
While oxygen reserves are not depleted (O_{pj}> 0) do
Take a Substring S_t = S_i[R_p,R_p + 4]
If OCT [FindPosition(S_t)] not min
S_t = OCT[min]
S_i[R_p,R_p + 4] = S_t
Endif
R_p= R_p + random step
O_{pj} = O_{pj}-1
EndWhile
Compute sensitivity for this penguin: F($P_{j(}S$))
Update Qf_j,p_j
EndFor
Update P_{S_i}
Redistributes the penguins according to the probabilities
BestOC(S) = argmax(F(P_j(S)))
EndRepeat
Return(S)

4.4.4 Experimental Results

4.4.4.1 Parameter Settings

Penguins Search Optimisation Algorithm has several parameters (initial oxygen reserve, population size, number of generations, etc.) allowing the diversification and the intensification properties. Determining these parameters values is an NP-hard problem; it is difficult to find the exact parameter values to reduce execution time for the evolution of the optimal objective value. In Pe-Seed, we have used the Hill Climbing algorithm to search the optimal parameter values (Algorithm 4.4), the pseudo code is shown below. The objective function of this algorithm is to maximise the ratio between the Pe-Seed objective function (the overlapping value) and the CPU execution time. In each iteration, we try with a new set of parameter values to maximise the overlap and to reduce the execution time. Figure 4.2 shows the experimentation for finding the best parameter setting by the ratio between the overlap complexity of used seeds and the runtime. The optimal parameters values are 25 penguins on the initial population with 150 iterations and 2 iterations for oxygen reserve.

Algorithm 4.4: Hill climbing Algorithm for parameters settings (S)

Input : a multiple seeds S = $S_1, S_2...S_k$
Output: a good parameters
1 **Generate** initial random values: P_{Best} = (Op_j,Number of iteration, Number of penguins)
2 **Apply** Pe-Seed algorithm with calculating the fitness function F (The ratio between Overlapping objective function and the execution time)
3 **While** F (P_{Best}) stopsdecreasing **do**
4 **Take** a random walk in each parameters P_{new}
5 **Apply** Pe-Seed algorithm with this new parameters (P_{new})
6 **If** (F*(P_{new}) > F(P_{Best})) then
7 $P_{Best} = P_{new}$
8 **EndIf**
9 **EndWhile**
10 **Return**(P_{Best})

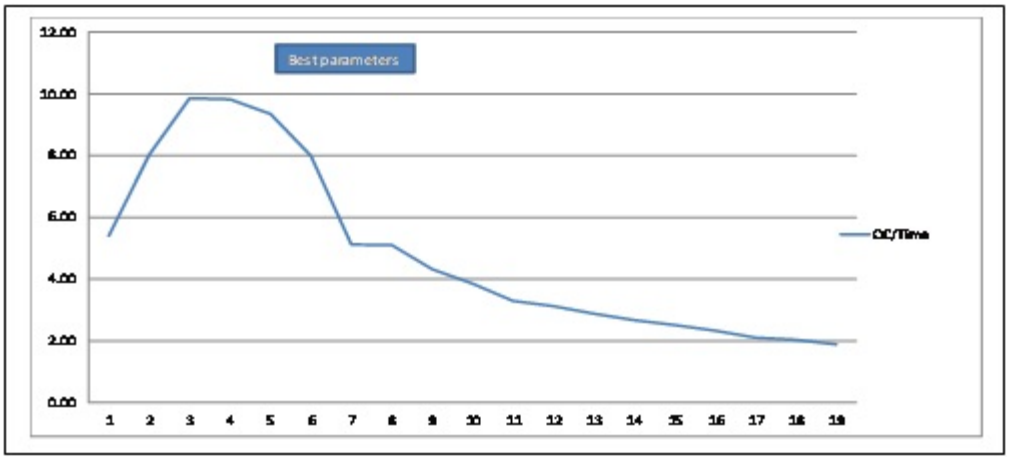

Figure 4-2: Parameters settings

4.4.4.2 QUICKLYOC Evaluation

We have compared the runtime of the QuicklyOC algorithms with the VFastOC algorithm used in the improved SpEED method in Table 2. These comparisons show that the proposed algorithm is very fast and its runtime is lower than that of the VFastOC. The QuicklyOC has a complexity of O($w_1 \times w_2$), which is the lowest complexity compared to all the other existing algorithms. Amelioration is approximately between two times to ten times, according to seeds length. Table 4.2 represents a comparison of the new QuicklyOC algorithms and the VFastOC for the required running time. The runtime of the Pe-Seed has been well ameliorated with the use of the QuicklyOC, we obtain a good set of spaced seed in a few minutes for long seeds.

4.4.4.3 Pe-SeeD Evaluation

We experimentally evaluated the Pe-SeeD and compared it with other software programs, Mandala, and the FastHC heuristic of the improved method of SpEED, based on the parameter settings that were practically used in several software of biological sequence alignment programs such as SHRiMP ,PatternHunter II , and BFAST. In all these data sets we have multiple seeds of various weight w (11, 20 and 22 respectively), and with different matching probability p from 0.70 to 0.95. We have performed Pe-SeeD search for 5000 solutions as done in the improved implementation of SpEED. We have also used a penguin's popula-

w	L	VFastOC	QuicklyOC
7	14	0.0033	0.0015
8	16	0.0037	0.0016
9	17	0.0041	0.0020
10	18	0.0045	0.0022
11	20	0.0048	0.0024
12	20	0.0068	0.0036
13	22	0.1011	0.0092
14	23	0.1750	0.0108
15	25	0.7842	0.0542
16	26	1.4209	0.1214
17	26	1.8648	0.1451
18	27	2.8648	0.2360
19	29	3.1251	0.2981
22	32	4.0022	0.3463
23	34	4.3523	0.3791

Table 4.2: Comparison of OC running time algorithms

tion of size 25, and each of them uses the OCT table and QuicklyOC to improve the sensitivity. Table 4.3 shows the sensitivity comparison of Pe-Seed with Mandala, AcoSeed, and

w	N	P	PHII	Mandala	FastHC	Pe-Seed
11	64	0.70	92.4114	92.3967	93.3406	94.0015
11	64	0.75	98.4289	98.4251	98.7156	98.9362
11	64	0.80	99.8449	99.8401	99.8859	99.9837

Table 4.3: Comparison of computed spaced seeds Sensitivity for PatternHunter (16 seeds, N=64)

the FastHC of SpEED. We have used the same parameters that are used in Pattern-Hunter II; it is evident from the table that the Pe-Seed has the best sensitivity in all cases.

Table 4.4 depicts the sensitivity comparison of Pe-Seed with Mandala, AcoSeed, and the FastHC of SpEED. We have used the same parameters that are used in BFAST. It seems from the table that the Pe-Seed also has the best sensitivity in all cases.

w	N	P	BFAST	FastHC	Pe-Seed
22	50	0.85	58.6907	60.9329	61.8214
22	50	0.90	87.3359	88.7120	89.6523
22	50	0.95	99.2249	99.3959	99.9215

Table 4.4: Comparison of computed spaced seeds Sensitivity for BFAST (16 seeds)

W	N	P	SHRiMP	Mandala	FastHC	Pe-Seed
10	50	0.75	89.6113	90.6608	90.7752	90.9254
10	50	0.80	97.3159	97.7316	97.8125	98.0014
10	50	0.85	99.6613	99.7283	99.8332	99.8925
11	50	0.75	81.6772	83.0512	83.2015	83.3111
11	50	0.80	94.1141	94.7845	94.9952	95.1024
11	50	0.85	99.0145	99.1929	99.3124	99.4685
12	50	0.80	89.3037	90.2615	90.5002	90.7012
12	50	0.85	97.7253	98.0841	98.2362	98.4126
12	50	0.90	99.8330	99.8832	99.9852	99.9921
16	50	0.85	84.0995	84.4142	84.5962	84.7102
16	50	0.90	97.1676	97.1895	97.3712	97.5638
16	50	0.95	99.9260	99.9365	99.9912	99.9989
18	50	0.85	71.1961	72.1954	72.8032	72.9100
18	50	0.90	92.5652	93.0855	93.5624	93.6463
18	50	0.95	99.6299	99.6603	99.7652	99.9490

Table 4.5: Comparison of computed spaced seeds Sensitivity for SHRiMP (4 seeds)

The Pe-Seed have improved also the computational time, we have compared the proposed approach with the FastHC Hill Climbing algorithm for a variety of parameters, the computational runtime (in seconds) given for a single multiple spaced seed with the given parameters. The results are summarized in Table 4.5 and 4.6.

W	N	P	k	[l1,lk]	FastHC	Pe-Seed
11	64	0.70	16	[14,27]	1.12	1.10
22	50	0.85	10	[25,37]	1.18	1.17
28	100	0.90	8	[36,56]	3.21	2.98
28	150	0.90	8	[39,63]	5.01	4.39
28	200	0.90	8	[41,70]	7.85	6.99
28	100	0.90	16	[33,59]	38.36	31.25
28	150	0.90	16	[36,66]	58.64	46.12
28	200	0.90	16	[39,72]	75.98	64.32

Table 4.6: Comparison of computation time for Pe-SeeD and FastHC algorithms

4.5 Conclusion

In this part of the thesis we investigated the problem of finding optimal spaced seeds for similarity searching biological data. We used in this approach the penguin search optimisation algorithm for finding the optimal spaced seeds. The approach starts by constructing the OccurrenceTable (OCT), this table presents the occurrence for each model of the same weight in the Bernoulli model of the similarity. The optimisation process starts by finding the optimal position of each model. The approach has been evaluated with differences benchmark for the spaced seed problem and compared with the well-known proposed approach in the literature. The success of the approach is due to the use of the penguins search space of solutions exploration strategy and the collaboration between penguins to converge quickly to the optimal spaced seed with the low sensitivity.

Chapter 5

Genetic Algorithm for Biological Data Compression

5.1 Introduction

The knowledge discovery process contains three main parts, (Pre-processing, Data mining, and Post-processing). Before the application of any machine learning algorithm to extract useful data, these data need to be prepared for more than one reason such as the heterogeneity of the data, and the representation of the data. In this chapter, we discuss the problem of data representation for biological data by using genetic algorithm. The work of this part of the thesis is published in an international journal (Gheraibia et al., 2015b). The aim of the proposed approach is to find the optimal codes of each part of a given biological sequences for finding the optimal size of the sequence. The rest of the chapter is organised as follows: The section 5.3 presents the cost considering Huffman code for data compression, in section 5.4 we discuss the use of genetic algorithm for the optimal allocation of this codewords. In section 5.5 we describe the results founded by the proposed approach when applied to a real biological data. The last section 5.6 we conclude our work and discussing the advantage of the proposed approach.

5.2 Cost Considering Approach for Huffman Code

The classical Huffman algorithm aims to reduce the total number of bits and it constructs a tree in a bottom up fashion. It is shown in (Golin and Rote, 1998) that if the costs of letters are considered unequal then the straightforward bottom up greedy approach does not work. Kabiret al. (2014) uses a top down approach to build a binary tree considering unequal letter cost of bits. They considered cost (length) of 0 and 1 as integer constants α and β, $\alpha < \beta$. Using the analogy of Morse code's '$\bullet$' and '$-$', the value of α and β is set as 1 and 3 respectively.

The complete algorithm to obtain an optimal prefix-free code for unequal letter cost is shown below (algorithm 1). The inputs to the algorithm are the distinct triplets contained in the genome sequence to be encoded and their frequencies. The process of creating the binary tree starts with a single node (root node) and it is initialised with cost 0. After that, two child (leaf) nodes are created for the root node, i.e., level of the tree is increased by one. Cost of the left child is calculated as the summation of the cost of its parent node and the length of the left arc, and cost of the right child is calculated as the summation of the cost of its parent node and the length of the right arc. Length of left and right arcs are actually the cost (length) of 0 (α) and 1 (β) respectively. The next step is to take a child node with least cost and create two child nodes for it and make it a parent node. In this way, in each iteration the child node with least cost becomes a parent node with two new child nodes. Creation of new child nodes is stopped when total number of child nodes becomes equal to the number of distinct triplets needed to be encoded.

Algorithm 5.1: Cost Considering Algorithm (CCA)

Input : Distinct symbols contained in the message to be encoded and their frequencies

Output: Non-uniform / variable letter cost i.e, Cost-considering balanced tree

1 **For** each distinct symbol i **do**
2 **Enqueue** (max-Q, frequency [i])
3 **Endfor**
4 **Create** a root node
5 **Cost** [root] ←0
6 **Enqueue** (min-Q, cost [root])
7 **Define** costs of the left and right child of the binary tree
8 **Repeat**
9 Cost-of-parent-node ← Dequeue (min-Q)
10 **Create** left and right child for this node
11 **Cost** [left-child] ← cost-of-parent-node + left-child-cost
12 **Enqueue** (min-Q, cost [left-child])
13 **Cost** [right-child] ← cost-of-parent-node + right-child-cost
14 **Enqueue** (min-Q, cost [right-child])
15 **Mark** parent node as explored
16 **Until** 2(n -1) nodes are created
17 **While** min-Q $\neq \emptyset$ do
18 Leaf-node← Dequeue (min-Q)
19 **Frequency** [leaf-node] ← Dequeue (max-Q)
20 **End while**
21 **For** each parent node j **do**
22 Frequency [j] ←frequency [left-child] + frequency [right-child]
23 **End for**
24 **Repeat**
25 **If** conflict between nodes **then**
26 **Resolve** conflict by swapping conflicted nodes
27 **Calculate** and reassign cost of all affected nodes
28 **Calculate** and reassign frequency of all affected nodes
29 **End if**
30 **Until** all conflicts are resolved

Now the tree T is constructed and the cost of the tree actually depends on how the frequencies are assigned to the leaf nodes. The overall cost will be minimised if the leaves with the highest cost always have smaller or equal weight (frequency). To fulfil this condition the leaves of the T are enumerated in non-decreasing order of their cost, i.e., $cost(l_1) \leq cost(l_2) \leq ... \leq cost(l_n)$; and that $f_1 \geq f_2 \geq ... \geq f_n$, where l_i and f_i are leaf nodes and frequencies of distinct triplets respectively for i = 1,2 ... n . The frequency or

weight of parent nodes are calculated as the sum of the frequencies of its child nodes, and it continues upwards until the root node is reached. After that, the algorithm checks for any possible conflicts between all pair of nodes. Two nodes are considered to be in conflict if the node with a higher cost has higher frequency violating the above condition, i.e., if cost $(l_i) >$ cost (l_j) and $f(l_i) > f(l_j)$, then there remains a conflict. If there remains a conflict between nodes, then it is resolved by swapping the nodes and recalculating the cost of the tree downward and frequency of the nodes upward. When all the conflicts are resolved, the algorithm generates codes for each of the distinct triplets.

5.3 The Proposed Approach

A genome is a stretch of DNA that encodes a polypeptide (protein) which is a set of amino acids bound together in a specific order. Each genomic sequence consists of nucleotides bound together, which are interpreted by the cellular machinery in groups of three, called triplets (Lodish et al., 1999). This is the main reason to divide the whole sequence into a set of triplets and give a code to each triplet. As each DNA sequence contains a combination of four nucleobases-guanine (G), adenine (A), thymine (T), and cytosine (C), it is possible to have 43=64 triplets. The first step in the optimised cost considering algorithm is to cut the genome sequence into triplets, then compute the frequency of each triplet in the whole sequence. This table of frequencies is used by the cost considering Huffman code to generate minimal cost code for each triplet (frequency). Finally, these codes with frequencies are used by the optimised cost considering algorithm to generate the optimal allocation with a given penalty on cost. The whole process is shown in Figure 5.1.

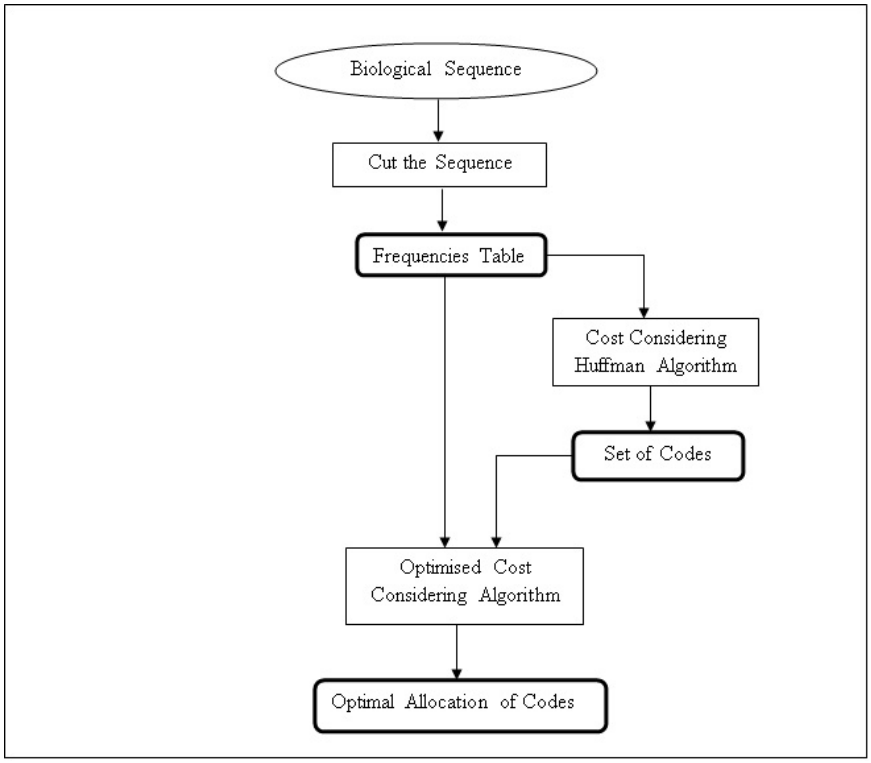

Figure 5-1: The proposed scheme

5.4 Optimal Allocation of the Codes

5.4.1 Problem formulation

The problem of finding the best allocation of codes to each triplet can be modelled as an assignment problem and is formulated as follows:

Definition: Given a set of codes C = $\{C_1, C_2...C_n\}$, and a set of frequencies F = $\{F_1, F_2...F_n\}$. For each code we have the length of the code $|C_i|$ (number of bits) and the cost of the code $S(C_i)$.

The objective is to assign to each frequency a code in order to minimise the total number of bits, while respecting the initially assigned total cost S_t with a given penalty $\lambda \rightarrow [0, 1]$. This penalty coefficient represents the allowed amount of cost that can be sacrificed to optimise the total number of bits. The Objective Function is to:

$$Minimise \sum (|C_i| \times F_j) \tag{5.1}$$

while:

$$(|C_i| \times F_j) \leq (\lambda + 1)S_t \tag{5.2}$$

5.4.2 Basic Genetic Algorithm

The Genetic Algorithm (GA) is a bio-inspired meta-heuristic algorithm developed by Mitchell (1998) (See figure 5.2). GA is a stochastic optimisation algorithm that imitates the natural evolution process of genomes. GA started by generating an initial population of random feasible solutions. The optimisation process of GA takes the initial population and generates a new population based on it. The process can be described as follows: First, select two or more solutions from the current population by using one of the well-known selection techniques (Blickle and Thiele, 1995). These selected solutions will be considered as parents. The second operation of the genetic algorithm is the crossover, which takes these parents as inputs to generate other new solutions considered as children. The third operation of the genetic algorithm is the mutation which ensures a good diversification in the search process. The new solutions can be mutated according to a given mutation probability. The mutation operation changes the value of one or more positions in the solution. The quality of each solution is verified by the fitness function which controls the evolution of the GA population by deletion of the worst solutions and insertion of the good solutions among parents and children. This process is repeated until the stopping criterion is reached which can be the number of generations or whether or not the population is stabilised.

5.4.3 Optimised Cost Considering Algorithm

The main objective of the optimised cost considering algorithm (OCCA) is to apply GA to find an optimal allocation of the codewords to each triplets to reduce the total number of bits. In GA, the encoding of the solution into a chromosome is one of the most important issues in obtaining good optimisation results. The OCCA uses two fixed length arrays of size 64 which is the number of combination for all nucleotides. The first array contains the

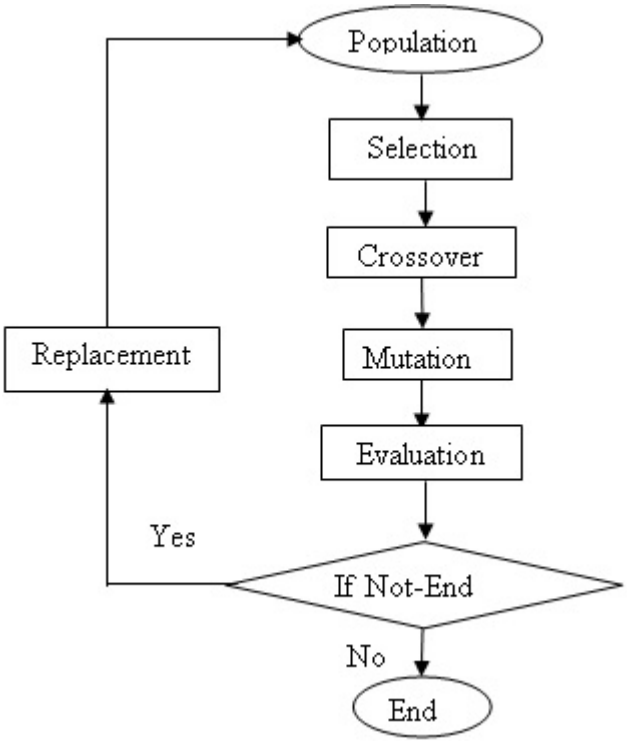

Figure 5-2: Genetic Algorithm

frequencies of each triplet and the second array contains the cost of the codewords assigned to each triplets. Our genetic algorithm utilises the two arrays and uses the index of each entry in the allocation process. The algorithm 5.2 describes the outline of the OCCA. Generally the initial population is generated in a random assignment of codes to different triplets. In the OCCA algorithm, the population firstly contains the assignment given by the CCA. The rest of the population is randomly generated. All these generated solutions must satisfy the initial cost constraints, which is the cost (St) of the solution produced by CCA and previously added to the population with a given penalty (step 1).

The evolution of the population is the key of the genetic optimisation algorithm. During each generation, the process starts with the selection of a part (set of solutions) of the population to breed a new generation. In the literature many selection methods have been proposed to guide the population evolution (Blickle and Thiele, 1995). The different methods for selecting a part (set of solutions) of the population vary from a random selection method to heuristic based selection methods. We have chosen to randomly select the part of the population to be processed as the heuristic methods are very time consuming (step

Algorithm 5.2: Optimised Cost Considering Algorithm (OCCA)

Population initialization (P)
repeat
Select two solutions S_1, S_2 form P
Crossover S_1, S_2 to generate S_{11}, S_{21} (Children)
Mutate S_{11},S_{21}
Validate children with cost constraint (equation 2)
Add children to population
Rank the population by fitness
Remove worst candidates until population limit
until maximum number of generation not achieved
Display the best solution from the population P

3). After that the operations of the genetic algorithm are applied on the initial population to generate a new generation of the population (see Figure 5.3). Firstly, the crossover operation is applied to these two selected solutions (considered as parents) to generate two new solutions (considered as children) (step 4).

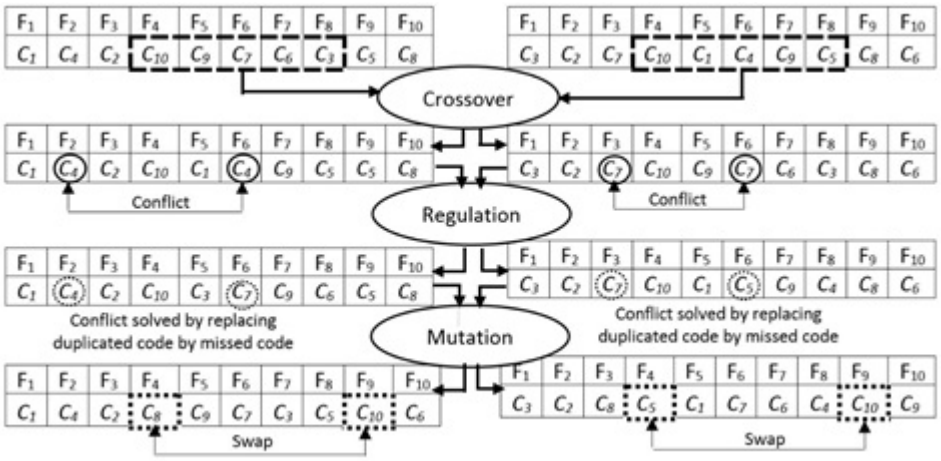

Figure 5-3: Operations of genetic algorithm

In the literature many crossover techniques have been used in genetic algorithms (Osaba et al., 2014), such as one-point crossover which divides the chromosomes into two fragments and recombines the second fragments with the other chromosome's second fragment. The two-point crossover divides the chromosomes into three fragments and recombines the middle fragment with the middle fragment of the other chromosome. There are many other crossover techniques to allow a good convergence of the algorithm. In our case, we have

used the two point crossover with two parameters. The first parameter pc is a random value in [0, 63], which represents the first cut point and the second parameter pm is also a random value in [0, 63], which represents the number of positions to be crossed. These two random parameters are to ensure a good diversification on the whole search space (step 4).

After the crossover operation, the newly generated children may contain conflicts, for example, a single code is allocated to two different frequencies in the solution. To overcome such a conflict, a regulation operation is performed to refine the solution to ensure the correctness of the solution (see Figure 5.). Secondly these two new solutions are mutated according to a predefined probability λ, the best value of the mutation rate is problem specific (step 5). In our case, the value of λ is fixed to 0.2 to explore a few positions in the solution. The mutation operator is used to maintain genetic diversity from one generation of the population of genetic algorithm to the next. In our case, we have used the mutation as a random swap mutation operator (see Figure 5.4). Each newly generated solution must satisfy the cost constraint. The next step is to add these two new solutions (children) to the current population (step 7) (see Fig. 3). Finally, the new population are ranked by fitness (step 8), and the worst solutions are deleted until the initial size of the population is obtained (step 9). The whole processes are repeated until the maximum number of operations is performed (step 10).

Figure 5-4: Population update for genetic algorithm

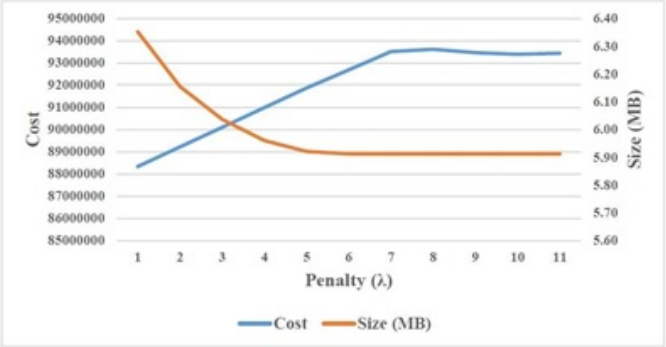

Figure 5-5: Convergence of OCCA for Genome 2

5.4.4 Results and Discussion

The effectiveness of the approach has been evaluated with different real genomic biological data. These genomes were downloaded from a recent version of The National Centre for Biotechnology Information (NCBI) available on (http://www.ncbi.nlm.nih.gov) (Pruitt et al., 2009). We focused on the sequences alone, ignoring any header and any other exogenous information. In table 3, the different data sets are described with the size of each of them in megabytes (MB) and the references on the biological data bank. Table 4 presents the results obtained by the classical Huffman code, the cost considering algorithm (CCA) and the optimised cost considering algorithm (OCCA) without penalty on cost. The results show that the number of bits required by the classical Huffman algorithm to encode genomic data is the minimum among the other algorithms but the cost is maximum.

Data sets	**Name**	**Size (MB)**	**Reference**
Genome 1	Mycobacterium smegmatis	6.66	CP009496
Genome 2	Amycolatopsisbenzoatilytica	8.30	KB912942
Genome 3	Mycobacterium rhodesiae NBB3	6.11	CP003169
Genome 4	Streptomyces bottropensis ATCC 25435	8.54	KB911581
Genome 5	Mycobacterium smegmatis str MC2 155	6.66	CP009494
Genome 6	Mycobacterium smegmatis MKD8	6.76	KI421511
Genome 7	Bradyrhizobium WSM471	7.42	CM001442
Genome 8	Amycolatopsisthermoflava N1165	8.27	CM001442
Genome 9	Bacillus thuringiensis Bt407	5.74	CM000747
Genome 10	Bacillus thuringiensisserovarthuringiensis	6.03	CM000748
Genome 11	Pseudomonas aeruginosa 9BR	6.48	AFXI010001
Genome 12	Bacillus thuringiensisserovarberliner	5.97	CM000753
Genome 13	Bacillus thuringiensisserovarpakistani	5.75	CM000750
Genome 14	Pseudomonas aeruginosa LES400	6.28	CP006982
Genome 15	Musmusculus chromosome 1	25.58	GL456087
Genome 16	Daniorerio chromosome 1	56.14	CM002885
Genome 17	Homo sapiens chromosome 18	76.64	CM000680
Genome 18	Homo sapiens chromosome 22	99.94	CM000684

Table 5.1: Datasets description

Data sets	**Huffman Algorithm**	**CCA**	**OCCA (λ=0)**
Genome 1	76787151	67416213	67416213
Genome 2	10042540	88430665	88430665
Genome 3	75940155	66745619	66745619
Genome 4	103552729	90821835	90821835
Genome 5	82234926	71963876	71963876
Genome 6	83454842	73038795	73038795
Genome 7	92539488	81416359	81416359
Genome 8	99613856	87102639	87102639
Genome 9	71876739	62998800	62998800
Genome 10	75324432	66084958	66084958
Genome 11	80766360	70620666	70620666
Genome 12	74560825	65359604	65359604
Genome 13	71562941	62758225	62758225
Genome 14	78261299	68354090	68354090
Genome 15	324439242	286008420	286008420
Genome 16	703734840	618859291	618859291
Genome 17	901032667	791840455	791840455
Genome 18	983434816	867299889	867299889

Table 5.2: Comparison of code words cost (Cost(1)=3,Cost(0)=1) among classical Huffman code, CCA, and OCCA without penalty

Data sets	**Huffman Algorithm**	**CCA**	**OCCA (λ=0)**
Genome 1	4.44	4.92	4.82
Genome 2	5.81	6.51	6.35
Genome 3	4.39	4.87	4.79
Genome 4	5.96	6.69	6.47
Genome 5	4.74	5.27	5.17
Genome 6	4.81	5.34	5.24
Genome 7	5.36	5.93	5.85
Genome 8	5.74	6.41	6.30
Genome 9	4.15	4.58	4.49
Genome 10	4.36	4.81	4.73
Genome 11	4.66	5.41	5.04
Genome 12	4.31	4.75	4.67
Genome 13	4.14	4.56	4.50
Genome 14	4.51	4.80	4.89
Genome 15	18.77	20.66	19.94
Genome 16	40.77	45.08	43.18
Genome 17	52.08	57.36	55.12
Genome 18	56.98	62.42	60.70

Table 5.3: Comparison of code words size (MB) among classical Huffman code, CCA, and OCCA without penalty

Data sets	**Cost**	**Size (MB)**	λ **(%)**	**Compression ratio %**
Genome 1	70760174	4.50	4%	32.72%
Genome 2	92010490	5.91	5%	28.79%
Genome 3	69421783	4.47	3%	26.84%
Genome 4	96294638	5.99	5%	29.85%
Genome 5	74855668	4.81	5%	27.77%
Genome 6	76738330	4.86	4%	28.10%
Genome 7	84667949	5.47	3%	26.28%
Genome 8	92416243	5.76	5%	30.35%
Genome 9	66145783	4.21	4%	26.65%
Genome 10	68751359	4.44	3%	26.63%
Genome 11	72737300	4.79	2%	26.08%
Genome 12	67981988	4.38	3%	26.63%
Genome 13	65896779	4.18	4%	27.30%
Genome 14	71762305	4.55	4%	27.54%
Genome 15	297188002	19.31	3%	24.51%
Genome 16	643785655	41.68	3%	25.75%
Genome 17	823506882	53.05	3%	30.78%
Genome 18	901515198	58.99	3%	40.97%

Table 5.4: Effects of the penalty on the compression performance of the OCCA

The cost considering algorithm improves the representation of the generated codes in terms of cost but the number of bits. However, still it compresses the data by 37.54% in the best case, 16.51% in the worst case, and 22.08% on an average. In terms of cost, in the best case the CCA improves cost over classical Huffman code by 12.66%, in the worst case by 11.80%, and on an average by 12.24%. The optimised cost considering algorithm tries to find the best allocation of codes to frequencies by giving a penalty on cost. The outcome of this process is a fall in the total number of bits and a rise in the total cost. However, the cost is always lower than the cost incurred by the classical Huffman algorithm. At first,

the OCCA optimises number of bits without applying any penalty on the cost (see table 4 and 5). Afterwards, it continues giving penalty ranging from 1% to 10% on the cost until a balance is found between total cost and bits.

Figure 5.5 shows the convergence of the OCCA for minimising the number of bits for Genome 2 according to the cost constraint. For each genome, a maximum amount of effective penalty is identified; after this maximum value, increasing the penalty no longer helps to reduce number of bits, i.e., the number of bits reaches the minimum and cost reaches the maximum. Table 6 presents the best found number of bits for different datasets with different effective penalties. As seen in the table, the OCCA improves the compression ratio from 37.54% to 40.97% in the best case, from 16.51% to 24.51% in the worst case, and from 22.08% to 28.53% in the average case. It is evident from the table 5 that this improvement is obtained without increasing the cost significantly.

5.5 Conclusion

In this part of the thesis we discussed the problem of efficient biological data representation. We showed how the genetic algorithm can be used to find the optimal allocation of Huffman codes for genetic representation of the sequences. The approach starts by generating the Huffman codes for each triplet of the DNA sequence and after that the genetic operations are used for finding the optimal allocation of these codes to each triplet of the DNA sequence. The optimisation process tries to find at each iteration a set of codeword that doesn't have any conflict between each other and allocated perfectly to each DNA triplet.

Chapter 6

PeSOA for DNA fragment Assembly

6.1 Introduction

The DNA fragments assembly problem is one the most challenging problems of bioinformatics. In this chapter we investigate the problem for reconstructing the original DNA from small fragments of the primary sequence. The penguins search optimisation algorithm is applied to find the optimal position of each DNA fragment in order to find the optimal final order of the fragment to reconstruct the original DNA sequence. The optimisation process starts by generating random allocation of each fragment and tries to optimise the positions by maximising the overlap between the fragments. The rest of the chapter is organised as follows: the next section is about the DNA fragment assembly problem description, the section 6.3 presents the penguins search optimisation algorithm. The section 6.4 describes the proposed approach by presenting the general algorithm the fitness function and the encoding schema. The experiments and the comparison study are described in section 6.5 and at the end we conclude our work in the section 6.6.

6.2 DNA fragment Assembly Problem

The second generation sequencing (shotgun sequencing) technologies can quickly read a huge number of DNA fragment at a time while the size of the fragment must be short (each several hundred base-pairs long) (Metzker et al., 2010). This operation (sequencing) aims

to recognise the genetic information on the ground level by determining its sequence of bases of each fragment and then by assembling these fragments to construct the original DNA chain. The objective of DNA fragment assembly is to find the optimal order of the DNA fragment in order to maximise the overlapping scores between consecutive fragments. Mathematically speaking, the DNA fragment assembly can be formulated as follows:
Let I = $\{i_1, i_2, i_3...i_n\}$ be a set of DNA fragment with different size, and let $w\left(i_j, i_{j+1}\right)$ be the similarity measure (amount of overlap) between the two fragment i_j and i_{j+1}. The fragment assembly problem aims to find the best order in which these fragments have to be assembled back to construct the original full DNA chain. The objective function is to maximise the overlap between each two consecutive fragment:

$$F(I) = \sum_{j=1}^{n-1} w\left(i_j, i_{j+1}\right) \tag{6.1}$$

While: $i \in I$

As mentioned in (Meksangsouy et al., 2003), the DNA fragment assembly problem is similar to Travelling Salesman Problem (TSP) (Kruskal 1956). The fragment order in a given solution is like the specific position of each town on the TSP solution. DNA fragment assembly can be modeled as an Asymmetric Travelling Salesman Problem in which the distance between the town t_1 and the town t_2 is different from the distance between town t_2 and the town t_1. In DNA fragment assembly the similarity measure $w\left(i_j, i_{j+1}\right)$ is different from $w\left(i_{j+1}, i_j\right)$. The only difference between these two problems is that the similarity between the first and the last fragment is not included and does not affect the quality of the solution.

6.3 Penguins Search Optimisation Algorithm (PeSOA)

In recent years, many metaheuristics have been proposed to solve the combinatorial problems (Yang 2010). Penguins search optimisation algorithm is a new form of nature inspired metaheuristic optimisation algorithm, inspired from the collaborative hunting behaviour of penguins, proposed by (Gheraibia et al,. (2013)). Penguins synchronise their dives and

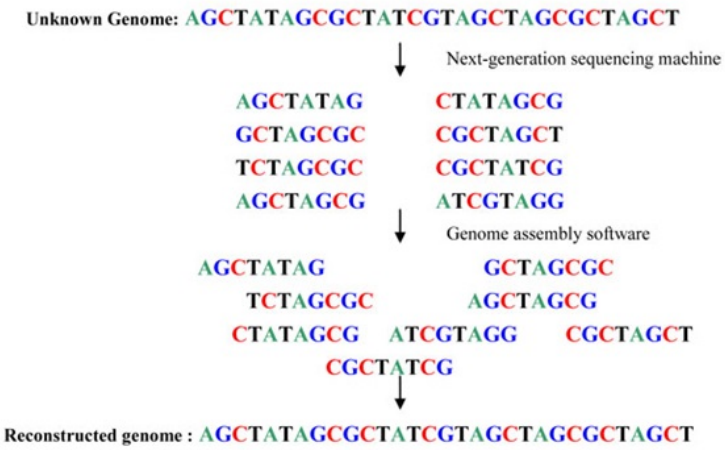

Figure 6-1: Example of the DFA greedy technique

collaborate their efforts to optimise the hunting process. The sea represents the whole solution space. The main objective of penguins is to find the best location rich in foods. The position under the water of the penguins represents a candidate solution. Each penguin expends energy by swimming under the water to catch prey, this prey contains energy, and the penguins search through the solution space to maximise the gain of energy. The health condition of a penguin is represented by its oxygen reserve, which serves as an acceleration coefficient in an instance of swimming.

The penguin's algorithm population comprises of several groups, and each group start searching from a specific location. The size of each group can vary depending on the availability food in the group's search area. The penguins follow their local leader, and they use the oxygen reserve to control the size of the search space to be explored at each iteration. A penguin uses two kinds of communications: intra-group communication by hunting together in a single group, and inter-group communication to control the distribution of the penguins' population on the whole solution space. The status of each penguin is represented by its position and oxygen reserve. The algorithm of penguins search optimisation algorithm is described in chapter 04.

6.4 PeSOA for DNA Fragment Assembly

6.4.1 Encoding

The DNA fragment problem consists of finding the best order of a set of DNA fragments with different length. To represent the fragment assembly problem more closely, the PeDFA uses the TSP without a circular solution. In the original PeSOA algorithm, the penguin swims through the solution space engaged in collaborative hunting strategy of penguins. In PeDFA encoding scheme, each solution is represented by a vector S of n elements where n is the number of fragment.

Let I = $\{i_1, i_2, i_3...i_n\}$ be a set of DNA fragment with different sizes. Each fragment is a vector of nucleotides Guanine (G), Thymine (T), Cytosine (C) and Adenine (A). The length of the final solution founded by the DNA fragments assembly algorithm will be less than the sum the initial DNA fragments.

6.4.2 Pe-DFA Algorithm

Pe-DFA algorithm (See Figure 6.2) starts with the generation of a random population P=$\{p_1, p_2, p_3...p_n\}$. This population is divided into groups in order to explore the whole solution space in an efficient way. Firstly, the algorithm divides the total set of fragments into small sets of fragments randomly. Each group has its active fragments; the penguins of each group are allowed to change only the positions of the active fragments. The division of the whole population in groups is made randomly by assigning to a set of fragments to a given group. After that, these small sets are assigned to different groups of penguins (see Figure 6.2). Initially, all the groups have the same size.

The penguins of a given group have two kinds of fragments: the active fragments which the penguins of this group are allowed to modify and the fixed fragments which the penguins are not allowed to modify. The aim of the group is to find the best combination for the set of the active fragments. Each penguin has its own active fragments and it must concentrate on its active fragments in the search process to achieve the goal of the group it belongs to. Each penguin (P_i) generates new solution by using the local search to increase the local

searching ability. PeDFA selects randomly a sub-solution and select another random cut point and then insert the sub-solution before the cut point position. After each generation, penguins communicate to each affiliates of the same group the best combination of its active fragments to ensure a good convergence to the optimum. In this stage, each penguin (P_i) will update its oxygen reserve (O_i) which represents the health of the penguin. This parameter allows the penguin to decide whether to hunt or not in a given area and also to compute the number of positions to be visited per iteration.

Penguins with depleted oxygen level can either leave the present group to join another existing group or without leaving the group continue searching in the same position including infeasible solution with low oxygen reserve. After that, the set of each group's local best solutions are ranked by fitness from high to low. The global solution is constructed based on these local best by cutting from each local solution the best combination of the local best (see Figure 6.3). This constructed solution can contain conflict with duplicated fragments which results in loosing other fragments on the final solution. This problem is resolved by deleting the duplicated fragments from the worst set of fragments and replacing them by the missed ones. The new global solution is compared with the previously ranked best local solution, and keeps the best as the global optimal solution.

Our main objective is to find the optimal order of DNA fragments in the optimal solution. Each penguin generates from its position (solution) a set of solutions (neighbours) if its oxygen reserve is not depleted, and choose the best solution that minimises the objective function. The oxygen reserve (acceleration coefficient) is updated according to the objective function. If a penguin ameliorates its solution, then the oxygen reserve for this penguin is increased to allow this penguin to move to other positions in the next iteration. After each dive, the oxygen reserve of the penguin is updated as follows.

$$O_j^i(t+1) = O_j^i(t) + \left(f\left(x_j^i\left(t+1\right)\right) - f\left(x_j^i\left(t\right)\right)\right) \times \|x_j^i\left(t+1\right) - x_j^i\left(t\right)\| \qquad (6.2)$$

The number of neighbours is updated according to the oxygen reserve. After each iteration, the number of neighbours is increased/decreased with propositional to increasing/decreasing level of the oxygen reserve (the number of neighbours is initialised to 1).

The distribution of penguins in different group is based on the improvement of the objective function of each penguin. Penguins that have improved their objective function means that they are in a good hunting area rich of fishes. The Quantity of Eaten Fish (QEF) by a group is the sum of the eaten fishes of each penguin of this group which is calculated by the following expression.

$$QEF^i(t+1) = QEF^i(t) + \sum_{j=1}^{d_i} \left(O_j^i(t+1) - O_j^i(t) \right) \tag{6.3}$$

The algorithm allows those penguins which continuously improve their objective function to move to more positions on the next generation, this means that the penguins are in rich area. The probability of the penguins of a given group is the ratio of the amount of eaten fishes of this group and the amount of eaten fishes of the whole population. The penguin function value of joining the group i is a probability given as follows.

$$P_i(t+1) = \frac{QEF^i(t)}{\sum_{j=1}^{k} QEF^j(t)} \tag{6.4}$$

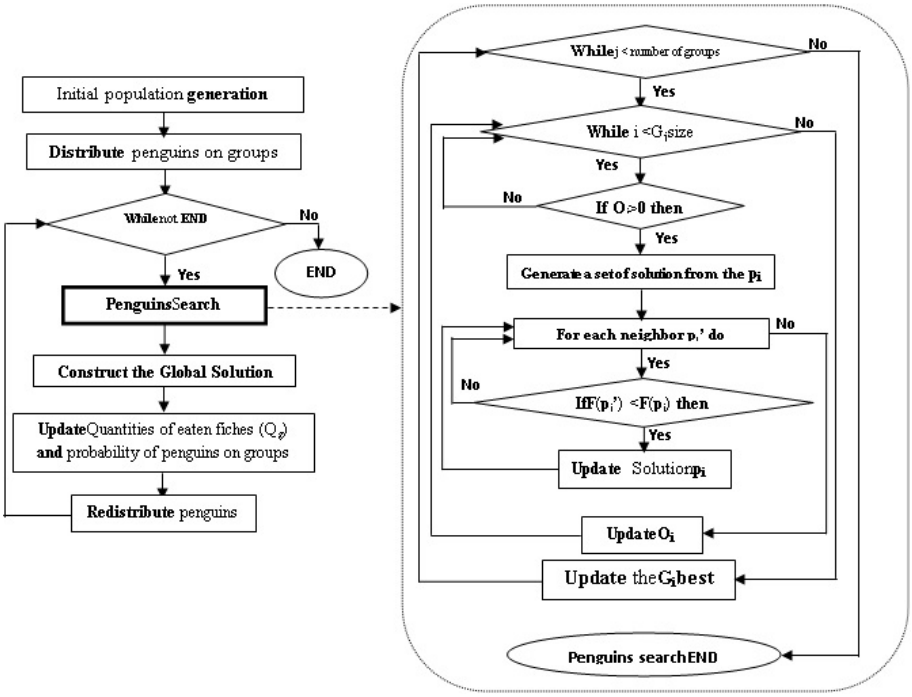

Figure 6-2: Pe-DFA algorithm

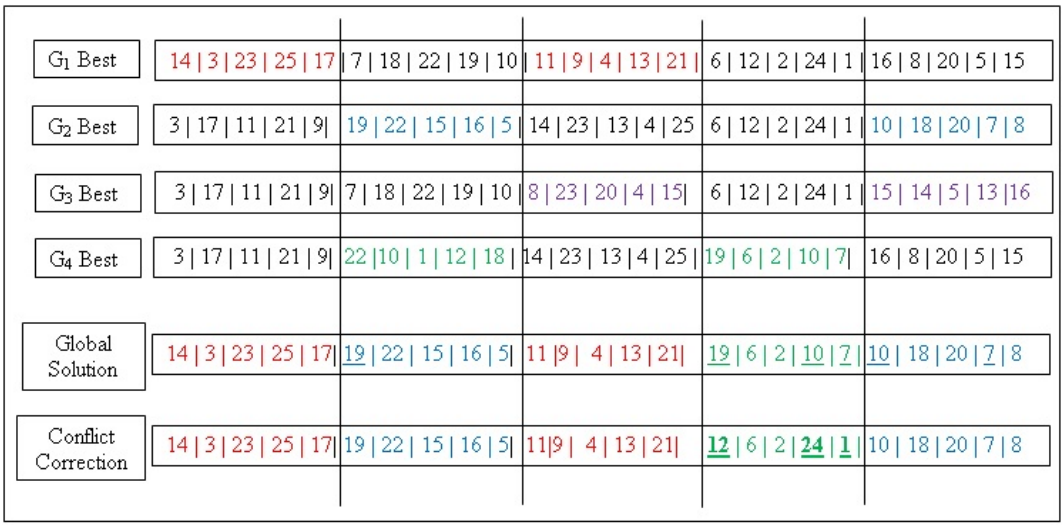

Figure 6-3: Global solution construction from each best group

6.5 Experimental Results

6.5.1 Parameter Settings

Penguins Search Optimisation Algorithm has several parameters (initial oxygen reserve, population size, number of generations, etc.) allowing the diversification and the intensification of the search process. Determining these parameter values is a NP-hard problem; it is difficult to find the exact parameter values to reduce the execution time for the evolution of the optimal objective value. In Pe-DFA, we have used the Hill Climbing algorithm to search the optimal parameter values; the pseudo code is shown below.

The objective function of this algorithm is to maximise the ratio between the Pe-DFA objective function (the overlapping value) and the CPU execution time. We try in each iteration with a new set of parameters values to maximise the overlap and to reduce the execution time. Figure 6.4 shows the experimentation for finding the best parameter setting by the ratio between the overlapping amount of the final DNA sequence construction and the run time. The optimal parameters values are 25 penguins on the initial population with 150 iterations, five groups and 1 step for oxygen reserve initialisation value.

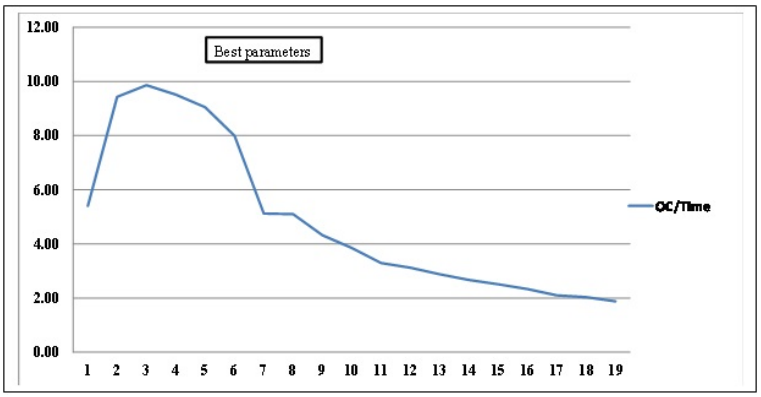

Figure 6-4: Parameters settings

6.5.2 Results and Comparison

This section shows the evaluation of the performance of the proposed DNA fragment assembly algorithm, Pe-DFA, using 16 problem instances from GenFrag (Engle et al., 1993) and DNAgen (Guillermo et al., 2013) tools available from the DNA Assembly Problem Benchmark Repository (www.mallen.mx/fragbench). The repository contains a variety of benchmarks from small instances to large scale instances obtained from real world problems. The first set of problems is obtained from GenFrag where the number of fragment varied from 39 to 773 fragments. The GenFrag tool takes as an input a DNA sequence and a set of specific criteria, after that, a set of DNA fragments is produced. The DFA problem instances used by GenFrag are shown in Table 6.1. The second set of instances is obtained from DNAgen, these sequences are called the ACIN sequences. These sequences are relatively longer than the GenFrag sequences, the number of fragments start from 307 to 1049 fragments. The DFA problem instances used by DNAgen are shown in Table 6.2.

Instances	Mean fragment length	Number of fragments	Coverage	length
x60189-4	395	39	4	3,835
x60189-5	286	48	5	3,835
x60189-6	343	66	6	3,835
x60189-7	387	68	7	3,835
m15421-5	398	127	5	10,089
m15421-6	350	173	6	10,089
m15421-7	383	177	7	10,089
j02459-7	405	352	7	20,000
bx842596-4	708	442	4	77,292
bx842596-7	703	773	7	77,292

Table 6.1: Data sets description (GenFrag instances)

Instances	Mean fragment length	Number of fragments	Coverage	length
acin1	182	307	26	2,170
acin2	1,002	451	3	147,200
acin3	1,001	601	3	200,741
acin5	1,003	751	2	329,958
acin7	1,003	901	2	426,840
acin9	1,003	1049	7	156,305

Table 6.2: Data sets description (DNAgen instances)

Benchmark	**QEGA**	**SA**	**PALS**	**LKH**	**PPSO**	**Pe-DFA**
x60189-4	11,476	11,478	11,478	11,478	11,478	11,618
x60189-5	14,027	14,027	14,021	14,161	13,642	14,425
x60189-6	18,266	18,301	18,301	18,301	18,301	18,266
x60189-7	21,208	21,271	21,210	21,271	20,921	21,804
m15421-5	38,578	38,583	38,526	38,746	38,686	38,746
m15421-6	47,882	48,048	48,048	48,052	47,669	48,097
m15421-7	55,020	55,048	55,067	55,171	54,891	55,020
j02459-7	116,222	116,257	115,320	116,700	114,381	116,818
bx842596-4	227,252	226,538	225,782	227,920	224,797	228,000
bx842596-7	443,600	436,739	438,215	445,422	429,338	443,600

Table 6.3: Comparison of the performance of the Pe-DFA with the well-known DFA methods (GenFrag instances)

Benchmark	**LKH**	**QEGA**	**SA**	**PALS**	**PPSO**	**Pe-DFA**
acin1	47,618	47,115	46,955	46,876	47,264	47,666
acin2	151,553	144,133	144,705	144,634	147,429	151,920
acin3	167,877	156,138	156,630	156,776	163,965	167,979
acin5	163,906	144,541	146,607	146,591	161,511	163,906
acin7	180,966	155,322	157,984	158,004	180,052	181,318
acin9	344,107	322,768	324,559	325,930	335,522	344,107

Table 6.4: Comparison of the performance of the Pe-DFA with the well-known DFA methods (DNAgen instances)

Benchmark	**QEGA**	**SA**	**PALS**	**PPSO**	**LKH**	**Pe-DFA**
x60189-4	300.54	198.24	221.14	124.32	102.02	100.34
x60189-5	165.24	132.48	199.01	107.94	96.33	90.04
x60189-6	145.24	162.97	148.35	124.65	81.75	81.21
x60189-7	102.57	98.26	85.38	97.51	76.17	71.87
m15421-5	312.05	285.18	245.93	261.11	265.08	257.14
m15421-6	278.27	264.37	219.58	232.76	198.92	189.20
m15421-7	197.86	169.35	178.44	188.62	175.24	164.83
j02459-7	118.75	108.37	100.26	120.82	99.32	91.03
bx842596-4	7.05	7.85	6.76	6.15	4.51	3.87
bx842596-7	4.15	3.89	3.25	3.84	3.14	3.01

Table 6.5: Comparison of the computational times (in Millisecond) of the Pe-DFA with the well-known DFA methods (GenFrag instances

Benchmark	**QEGA**	**SA**	**PALS**	**PPSO**	**LKH**	**Pe-DFA**
acin1	288.64	274.33	254.51	233.68	205.62	195.27
acin2	300.21	291.84	287.41	274.67	245.98	239.21
acin3	338.98	334.62	301.95	288.21	264.15	261.34
acin5	362.50	350.11	324.87	305.74	297.24	290.34
acin7	395.68	390.08	361.84	337.14	310.64	304.21
acin9	432.51	421.54	405.24	387.94	362.87	359.14

Table 6.6: Comparison of the computational times (in Millisecond) of the Pe-DFA with the well-known DFA methods (DNAgen instances)

Table 6.3 and 6.4 present the results obtained by the Pe-DFA compared with the well-known DNA fragment assembly methods for the GenFrag and DNAgen instances respectively. The results show that the order founded by the Pe-DFA improves the total overlapping score of the global solution with compared to other existing methods. For GenFrag

instances of smaller dimension, the proposed algorithm performs either better or similar to that of the existing methods. For DNAgen instances, the proposed method improves the results for the most of the instances. The advantage of the Pe-DFA is by combining the final solution from the best parts founded by each group of Penguins.

The penguins concentrate the search on its active fragments and fixing the non-active fragments. We evaluated each assembly result in terms of the overlapping scores of the assembled DNA fragments. Since the results of Pe-DFA vary depending on the different parameters of the algorithm, we performed parameters settings to find the different parameters values to achieve the optimal solutions. Table 6.5 and 6.6 present the run time comparison of the Pe-DFA algorithm with the well-known approaches. The experiments shows that the proposed algorithm improves the computational time a bit with compared to the other algorithms. This improvement is because of the acceleration coefficient which allows the penguin to converge quickly to the optimal solution. The approach has been evaluated with 10 independent runs for each test to gain sufficient accurate results.

6.6 Conclusion

In this chapter we discussed the use of penguins search optimisation algorithm for the DNA fragments assembly problem, this work has been published in an international journal (Gheraibia et al., 2015c). The proposed approach uses the penguins search optimisation algorithm for finding the optimal position of each DNA fragment in the final DNA reconstructing process. The success of the approach is the use of simple encoding scheme with the OLP model to describe the behaviour of the solution. The approach has been evaluated by the well-known benchmarks for the DNA fragment problem. The results are very promising compared with the well-known approach in the literature.

Conclusion and Future Work

his part provides a summary of the proposed contributions of this thesis, the conclusion of these works and future works. We have proposed several approaches for handling challenging problems in bioinformatics by using existing and new developed nature inspired metaheuristics algorithms. For each developed approach an experimental comparison has been carried out to prove the efficiency of these approaches and the benefits among the existing work in the literature. In this thesis we set out to study the use of nature inspired metaheuristics algorithms for solving combinatorial problems in bioinformatics. The aim of this thesis is to show the importance of using nature inspired metaheuristics algorithm where solving bioinformatics combinatorial problems. This thesis provides several applications of bio-inspired metaheuristic algorithms for the bioinformatics combinatorial problems and biological data representation. The contributions of this thesis are based on new developed metaheuristic and existing ones. The proposed optimisation approaches are scalable; they can be the centre of any optimisation applications in other fields. The contributions of this thesis can be summarised as follows:

- The optimal spaced seed finding for similarity searching: The problem of sequences alignment and similarity searching for biological data is one of the hard problems of bioinformatics, the large amount of biological sequence and the size of the biological sequence evolved a hard problem of similarity searching between these sequences. The first contribution of this thesis is using the new developed metaheuristics penguins search optimisation algorithm for finding the optimal spaced seed that have the high sensitivity between them. The results of the proposed approach are very promising compared with the existing approaches in the literature such as Idera and

Mandala.

- The biological data compression: as we mentioned earlier that the biological data banks have a very big amount of data, using these data for different machine learning algorithms is one of the hard problems in bioinformatics. In this thesis we worked on finding an efficient representation of the biological sequence. In this contribution we show how a genetic algorithm can be used to improve the compression performance of big biological data. The approach has been evaluated with real biological data such as the human genome.

- The last contribution of this thesis is for the DNA fragment assembly problem. We developed a new approach based on penguins search optimisation algorithm for DNA fragment assembly problems. The DNA fragment assembly problem is an NP-hard optimisation problem aims to find an optimal length of the original DNA sequence. This approach investigating the use of penguins search algorithm to classify each fragment in its appropriate position on the whole reconstructed DNA fragments. The approach has been evaluated with real biological data and proved its efficiency to finding the Original DNA sequence.

The experiments are based on the well-known benchmarks and the comparison also with the well-known methods in the literature. The results show the benefits of each proposed methods. The work of this thesis can be enhanced and augmented in order to improve the proposed contributions. The parameters of the penguins search optimisation algorithm will get the major interest; because the parameters of each optimisation algorithm affect directly the performance of the algorithm and the convergence to the optimal solution. A good neighbourhood search strategy can be incorporated to the main process of penguins search algorithm to improve the search strategy. The adaptation of the algorithm to other bioinformatics problems such as phylogenetic inferences and gene prediction and identification to be more effective and robust in handling multi-objective problems.

References

(Aarts et al., 1989): Aarts Emile, and Jan Korst. Simulated annealing and Boltzmann machines: a stochastic approach to combinatorial optimization and neural computing John Wiley and Sons, Inc. New York, NY, USA 1989.

(Altschul et al., 1990): Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ: Basic local alignment search tool. J Mol Biol 1990, 215(3):403-410.

(Altschul et al., 1997): Altschul SF, Madden TL, Schaffer AA, Zhang J, Zhang Z, Miller W, Lipman DJ: Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Res 1997, 25(17):3389-3402.

(Azevedo et al., 2013): Azevedo, L. S., Parker, D., Walker, M., Papadopoulos, Y., & Araujo, R. E. (2013). Automatic Decomposition of Safety Integrity Levels: Optimization by Tabu Search. Workshop CARS (2nd Workshop on Critical Automotive applications: Robustness & Safety) of the 32nd International Conference on Computer Safety, Reliability and Security - SAFECOMP 2013. Toulouse, France.

(Altenkamp et al., 1980): Altenkamp, D. and Mehlhorn, K. (1980) 'Codes: Unequal probabilities, unequal letter costs', Journal of the Association for Computing Machinery, Vol. 27, No. 3 pp.412-427.

(Amsterdam et al., 1986): Amsterdam, J. (1986) 'Data compression with Huffman coding', BYTE, Vol. 11, No. 5, pp.98-108.

(Alba et al., 2007): Alba, E., Garcia-Nieto, J., Jourdan, L., & Talbi, E. G. (2007, September). Gene selection in cancer classification using PSO/SVM and GA/SVM hybrid algorithms. In Evolutionary Computation, 2007. CEC 2007. IEEE Congress on (pp. 284-290). IEEE.

(Brona et al., 2005): Brona Brejova, Daniel G.Bro wn, Tomas Vina. Vector seeds: An

extension to spaced seeds Journal of Computer and System Sciences 70 (2005) 364-380.

(Blickle et al., 1995): Blickle, T. and Thiele, L. (1995) 'A comparison of selection schemes used in genetic algorithms', TIK-Report.

(Bradford et al., 2002): Bradford, P., Golin, M. J., Larmore, L. L., and Rytter,W. (2002) 'Optimal prefix-free codes for unequal letter costs: Dynamic programming with the Monge property', Journal of Algorithms, Vol. 42, No. 2, pp.277-303.

(Brandon et al., 2009): Brandon, M. C., Wallace, D. C., and Baldi, P. (2009) 'Data structures and compression algorithms for genomic sequence data', Bioinformatics, Vol. 25, No. 14, pp.1731-1738.

(Belegundu et al., 2011): Belegundu, A. D., & Chandrupatla, T. R. (2011). Optimization concepts and applications in engineering. Cambridge University Press.

(Betts 1998): Betts, J. T. (1998). Survey of numerical methods for trajectory optimization. Journal of guidance, control, and dynamics, 21(2), 193-207.

(Blickle et al., 1995): Blickle, T. and Thiele, L. (1995) 'A comparison of selection schemes used in genetic algorithms', TIK-Report.

(Blum et al., 2003): Blum, C. and Roli, A., 2003. 'Metaheuristics in combinatorial optimization: Overview and conceptual comparison', ACM Comput. Surv., 35, 268-308.

(Bellman 1958): R. Bellman. Dynamic programming and stochastic control processes. Information and control, 1(3):228-239, 1958.

(Boyd et al., 2004): Boyd, Stephen P.; Vandenberghe, Lieven (2004). Convex Optimization (pdf). Cambridge University Press. p. 129. ISBN 978-0-521-83378-3.

(Bell et al., 2004): Bell, J. E., & McMullen, P. R. (2004). Ant colony optimization techniques for the vehicle routing problem. Advanced Engineering Informatics, 18(1), 41-48.

(Bektas 2006): Bektas, T. (2006). The multiple traveling salesman problem: an overview of formulations and solution procedures. Omega, 34(3), 209-219.

(Baker 2003): Baker, B. M., & Ayechew, M. A. (2003). A genetic algorithm for the vehicle routing problem. Computers & Operations Research, 30(5), 787-800.

(Bankevich et al., 2012): Bankevich, A., Nurk, S., Antipov, D., Gurevich, A. A., Dvorkin, M., Kulikov, A. S. & Pevzner, P. A. (2012). SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. Journal of Computational Biology, 19(5),

455-477.

(Buhler et al., 2003): Buhler J, Keich U, Sun Y: Designing seeds for similarity search in genomic DNA. In Proceedings of RECOMB'03 New York: ACM; 2003, 67-75.

(Brown et al., 2007): Brown DG: A survey of seeding for sequence alignments. In In Bioinformatics Algorithms: Techniques and Applications. Edited by: Mandoiu I, Zelikovsky A. Hoboken: J. Wiley and Sons Inc; 2007:117-142.

(Besemer et al 1999): Besemer, J., & Borodovsky, M. (1999). Heuristic approach to deriving models for gene finding. Nucleic Acids Research, 27(19), 3911-3920.

(Berket al., 2000): Berk, A., & Zipursky, S. L. (2000). Molecular cell biology (Vol. 4). New York: WH Freeman.

(Caserta et al., 2014): Caserta, M., & Vob, S. (2014). A hybrid algorithm for the DNA sequencing problem. Discrete Applied Mathematics, 163, 87-99.

(Chang et al., 2011): Chang, YoungJung, and Nikolaos V. Sahinidis. "An integer programming approach to DNA sequence assembly." Computational biology and chemistry 35.4 (2011): 251-258.

(Cai et al., 2012): Cai, Y. H., & Huang, H. (2012). Advances in the study of protein-DNA interaction. Amino acids, 43(3), 1141-1146.

(Chenna et al., 2003): Chenna, R., Sugawara, H.,Koike,T., Lopez, R., Gibson,T. J., Higgins, D. G., and Thompson, J. D. (2003) 'Multiple sequence alignment with the clustal series of programs', Nucleic acids research, Vol. 31, No. 13, pp.3497-3500.

(Choi et al., 2004): Choi, K. P., Zeng, F., & Zhang, L. (2004, May). Good spaced seeds for homology search. In Bioinformatics and Bioengineering, 2004. BIBE 2004. Proceedings. Fourth IEEE Symposium on (pp. 379-386). IEEE.

(Cutello et al., 2006): Cutello, V., Narzisi, G., & Nicosia, G. (2006). A multi-objective evolutionary approach to the protein structure prediction problem. Journal of The Royal Society Interface, 3(6), 139-151.

(Chu et al., 2006): Chu, D., & Zomaya, A. (2006). Parallel ant colony optimization for 3D protein structure prediction using the HP lattice model. In Parallel Evolutionary Computations (pp. 177-198). Springer Berlin Heidelberg.

(Chen et al., 2010): Chen, S., Yao, W., Palally, H. R., &Hanzo, L. (2010). Particle swarm

optimisation aided MIMO transceiver designs. In Computational Intelligence in Expensive Optimization Problems (pp. 487-511). Springer Berlin Heidelberg.

(Chu et al., 1997): Chu, P. C., & Beasley, J. E. (1997). A genetic algorithm for the generalised assignment problem. Computers & Operations Research, 24(1), 17-23.

(Comellas et., al 1998): Comellas, F., & Ozon, J. (1998). An ant algorithm for the graph colouring problem. In ants'98-from ant colonies to artificial ants: first international workshop on ant colony optimizationn ant colony optimization.

(Chang et al., 2004): Chang, B. C., Ratnaweera, A., Halgamuge, S. K., & Watson, H. C. (2004). Particle swarm optimisation for protein motif discovery. Genetic Programming and Evolvable Machines, 5(2), 203-214.

(Clonis 2006): Clonis, Y. D. (2006). Affinity chromatography matures as bioinformatic and combinatorial tools develop. Journal of Chromatography A, 1101(1), 1-24.

(Dorigo et al., 2000): Dorigo, M., & Birattari, M. (2010). Ant colony optimization. In Encyclopedia of machine learning (pp. 36-39). Springer US.

(Dong et al., 2012): Dong Do Duc, Huy Q. Dinh, Thanh Hai Dang,Kris Laukens, and Xuan Huan Hoang AcoSeeD: An Ant Colony Optimisation for Finding Optimal Spaced Seeds in Biological Sequence Search ANTS 2012, LNCS 7461, pp. 204211, 2012.

(Daniel et al., 1994): Daniel Pierre Bovet; PierluigiCrescenzi. Introduction to the Theory of Complexity. Prentice Hall (1994). p. 69. ISBN 0-13-915380-2.

(Farah et al., 2011): Farah, I., Kansou, A., Yassine, A., & Galinho, T. (2011, May). Ant Colony Optimization for aircraft landings. In Logistics (LOGISTIQUA), 2011 4th International Conference on (pp. 235-240). IEEE.

(Grefenstette et al., 1985): Grefenstette, J., Gopal, R., Rosmaita, B., & Van Gucht, D. (1985, July). Genetic algorithms for the traveling salesman problem. In Proceedings of the first International Conference on Genetic Algorithms and their Applications (pp. 160-168). Lawrence Erlbaum, New Jersey (160-168).

(Green et al., 1998): Green, K., Williams, R., & Green, M. G. (1998). Foraging ecology and diving behaviour of macaroni penguins Eudyptes chrysolophus at Heard Island. Mar Ornithol, 26, 27-34.

(Garnier 1978): Garnier, J., Osguthorpe, D. J., & Robson, B. (1978). Analysis of the accu-

racy and implications of simple methods for predicting the secondary structure of globular proteins. Journal of molecular biology, 120(1), 97-120.

(Gheraibia et al., 2013): Gheraibia Y, Moussaoui A: Penguins Search Optimisation Algorithm (PeSOA). Recent Trends in Applied Artificial Intelligence, 2013 pp 222-231.

(Gheraibia et al., 2015a): Youcef Gheraibia, Abdelouahab MOUSSAOUI, Youcef Djenouri, Sohag KABIR, Peng-Yeng Yin, Smaine Mazouzi. Penguin Search Optimisation Algorithm for Finding Optimal Spaced Seeds. International Journal of Software Science and Computational Intelligence (IJSSCI). (In Press)

(Gheraibia et al., 2015b): Youcef Gheraibia, Sohag Kabir, Abdelouahab MOUSSAOUI, Smaine Mazouzi. Optimised Cost Considering Huffman Code for Biological Data Compression. International Journal of Information and Communication Technology. (In Press)

(Gheraibia et al., 2015c): Youcef Gheraibia,Abdelouahab MOUSSAOUI,Sohag Kabir, Smaine Mazouzi. Pe-DFA: Penguins Search Optimisation Algorithm for DNA Fragment Assembly.International Journal of Applied Metaheuristic Computing. Vol 7(2),2015.

(Glover et al., 198): F. Glover (1989) Tabu Search-Part I, ORSA J. Comput., 1(3): 190-206.

(Golin et al., 2012): Golin, M. J., Mathieu, C., and Young, N. E. (2012) 'Huffman Coding with Letter Costs: A Linear Time Approximation Scheme', SIAM Journal on Computing, Vol. 41, No. 3, pp.684-713.

(Golomb 1996): Golomb, S.W. (1996) 'Run-length encodings', IEEE Transactions on Information Theory, Vol. 12, No. 3, pp.399-401.

(Goldberg et al., 1988): Goldberg, D. E., & Holland, J. H. (1988). Genetic algorithms and machine learning. Machine learning, 3(2), 95-99.

(Grunwald et al., 2003): Grunwald, P. D. and Vitany, P. M. B. (2003) 'Kolmogorov complexity and information theory: With an interpretation in terms of questions and answers', Journal of Logic, Language and Information, Vol. 12, pp.497-529.

(Gloveret al., 1999): Glover, Fred, and Manuel Laguna. Tabu search. Springer US, 1999.

(Huang et al., 2014): Ko-Wei Huang, Jui-Le Chen, Chu-Sing Yang, Chun-Wei Tsai "A memetic particle swarm optimization algorithm for solving the DNA fragment assembly problem." Neural Computing and Applications (2014): 1-12.

(Hogeweg 2011): Hogeweg, P. (2011). The roots of bioinformatics in theoretical biology. PLoS Comput Biol, 7(3), e1002021.

(Homer et al., 2009): Homer N, Merriman B, Nelson SF: BFAST: An Alignment Tool for Large Scale Genome Resequencing. PLoS One 2009, 4(11).

(Hwang et al., 1979): Ching-Lai Hwang; Abu Syed MdMasud (1979). Multiple objective decision making, methods and applications: a state-of-the-art survey. Springer-Verlag. ISBN 978-0-387-09111-2.

(Jon 1982): Jon Louis Bentley (1982). Writing Efficient Programs. Prentice Hall. p. 11.

(Horst et al., 2000): R. Horst, P.M. Pardalos and N.V. Thoai, Introduction to Global Optimization, Second Edition. Kluwer Academic Publishers, 2000.

(Hall 1999): Hall, T. A. (1999, January). BioEdit: a user-friendly biological sequence alignment editor and analysis program for Windows 95/98/NT. In Nucleic acids symposium series (Vol. 41, pp. 95-98).

(Houston et al., 1985): Houston, A. I., & McNamara, J. M. (1985). A general theory of central place foraging for single-prey loaders. Theoretical Population Biology, 28(3), 233-262.

(Hanuise et al., 2010): Hanuise, N., Bost, C. A., Huin, W., Auber, A., Halsey, L. G., & Handrich, Y. (2010). Measuring foraging activity in a deep-diving bird: comparing wiggles, oesophageal temperatures and beak-opening angles as proxies of feeding. The Journal of experimental biology, 213(22), 3874-3880.

(Hazewinkel 2001): Hazewinkel Michiel, ed. (2001), "Computational complexity classes", Encyclopedia of Mathematics, Springer, ISBN 978-1-55608-010-4

(Huelsenbeck et al., 2001): Huelsenbeck, J. P., & Ronquist, F. (2001). MRBAYES: Bayesian inference of phylogenetic trees. Bioinformatics, 17(8), 754-755.

(Engle et al., 1993): Engle, Michael L., and Christian Burks. "Artificially generated data sets for testing DNA sequence assembly algorithms." Genomics 16.1 (1993): 286-288.

(Howe et al., 2008): Howe, D., Costanzo, M., Fey, P., Gojobori, T., Hannick, L., Hide,W., Hill, D. P., Kania, R.,Schaeffer, M., Pierre, S. S., Twigger, S., White, O., and Rhee, S. Y. (2008) 'Big data: Thefuture of bio-curation', Nature, Vol. 455, pp.47-50.

(Handl et al., 2007): Handl, J., Kell, D. B., & Knowles, J. (2007). Multiobjective opti-

mization in bioinformatics and computational biology. IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB), 4(2), 279-292.

(Julio et al., 2008): Julio R Banga: Optimisation in computational systems biology BMC Systems Biology 2008.

(Kabir et al., 2014): Kabir, S., Azad, T., Alam, A. S. M. A. and Kaykobad, M. (2014) 'Effects of unequal bitcosts on classical Huffman codes', In 17th International Conference on Computer andInformation Technology, IEEE, pp.96-101.

(Kennedy et al., 2010): Kennedy, J. (2010). Particle swarm optimization. In Encyclopedia of Machine Learning (pp. 760-766). Springer US.

(Ilie et al., 2011): Ilie L, Ilie S, Mansouri Bigvand A: SpEED: fast computation of sensitive spaced seeds. Bioinformatics 2011, 27(17):2433-2434.

(Ilie 2012): Ilie S: Efficient computation of spaced seeds. BMC Research Notes 2012 5:123.

(Kann 1992): KannViggo (1992), On the Approximability of NP-complete Optimization Problems, Royal Institute of Technology, Sweden, ISBN 91-7170-082-X.

(Kucherov et al., 2006): Kucherov G, Noe L, Roytberg MA: A Unifying Framework for Seed Sensitivity and its Application to Subset Seeds. J Bioinformatics and Computational Biology 2006, 4(2):553-570.

(Juang 2008): Juang, W. S., & Su, S. F. (2008). Multiple sequence alignment using modified dynamic programming and particle swarm optimization. Journal of the Chinese institute of engineers, 31(4), 659-673.

(Land et al., 1960): A. H. Land and A. G. Doig (1960). "An automatic method of solving discrete programming problems". Econometrica 28 (3). pp. 497âĂŞ520. doi:10.2307/1910129.

(Luscombe et al 2001): Luscombe, N. M., Greenbaum, D., & Gerstein, M. (2001). What is bioinformatics? A proposed definition and overview of the field. Methods of information in medicine, 40(4), 346-358.

(Lee 2004): Lee, Jon (2004), A First Course in Combinatorial Optimization, Cambridge Texts in Applied Mathematics 36, Cambridge University Press, p. 1, ISBN 9780521010122.

(Lodish et al., 1999): Lodish, H., Berk, A., Zipursky, S. L., Matsudaira, P., Baltimore, D., and Darnell, J. (1999)'Molecular Cell Biology', W.H.Freeman and Co Ltd, 4th edition.

(Mannan et al., 2003): Mannan, M. A., and Kaykobad, M. (2003) 'Block huffman coding', Computers andMathematics with Applications, Vol. 46, No. 10-11, pp.1581-1587.

(Mitchell et al., 1998): Mitchell, M. (1998) 'An introduction to genetic algorithms', MIT press.

(Osaba et al., 2014): Osaba, E., Carballedo, R., Diaz, F., Onieva, E. , de la Iglesia, I., and Perallos, A. (2014)'Crossover versus mutation: A comparative analysis of the evolutionary strategy ofgenetic algorithms applied to combinatorial optimization problems', The ScientificWorldJournal,, Vol. 2014, pp.1-22.

(Perl et al., 1975): Perl,Y., Garey, M. R. and Even, S. (1975) 'Efficient generation of optimal prefix code:Equiprobable words using unequal cost letters', Journal of the ACM(JACM),Vol. 22, No.2, pp.202-214.

(Pevzner et al., 2001): Pevzner, P. A., Tang, H., & Waterman, M. S. (2001). An Eulerian path approach to DNA fragment assembly. Proceedings of the National Academy of Sciences, 98(17), 9748-9753.

(Pruitt et al., 2009): Pruitt, K. D., Tatusova, T., Klimke, W., and Maglott, D. R. (2009) 'NCBI referencesequences: current status, policy and newinitiatives', Nucleic acids research,Vol. 37, No.suppl 1, pp.D32-D36.

(Redmond 1964): Redmond, W. A. (1964) 'International morse code', Microsoft Encarta 2009(DVD), pp.275-278.

(Scott et al., 2009): Scott, C., Yiming, L., Chen, L., and Xiaohui, X. (2009) 'Human genomes as emailattachments', Bioinformatics, Vol. 25, No. 2, pp.274-275.

(Tzu et al., 2014): Tzu-Hao, C., Shih-Lin, W., Wei-Jen, W., Jorng-Tzong, H., and Cheng-Wei, C. (2014)'A novel approach for discovering condition-specific correlations of gene expressionswithin biological pathways by using cloud computing technology', BioMed researchinternational, Vol. 2014.

(Mori et al., 2002): Mori, Y. (2002). Optimal diving behaviour for foraging in relation to body size.Journal of Evolutionary Biology, 15(2), 269-276.

(Melgani et al., 2008): Melgani, F., & Bazi, Y. (2008). Classification of electrocardiogram signals with support vector machines and particle swarm optimization. Information Technology in Biomedicine, IEEE Transactions on, 12(5), 667-677.

(Ma et al., 2007): Ma B, Li M: On the complexity of the spaced seeds. J Comput Syst Sci 2007, 73(7):1024-1034.

(MacArthur et al., 1966): MacArthur, R. H., & Pianka, E. R. (1966). On optimal use of a patchy environment. American Naturalist, 603-609.

(Lewis 1998): Lewis, P. O. (1998). A genetic algorithm for maximum-likelihood phylogeny inference using nucleotide sequence data. Molecular Biology and Evolution, 15(3), 277-283.

(Li et al., 2002): Li M, Brown D. Mouse Genome Sequencing Consortium. Initial sequencing and comparative analysis of the mouse genome . Nature. 420(6915):520-522. December 2002.

(Lin et al., 2007): Lin, Y. M., Fang, S. C., & Thorne, J. L. (2007). A tabu search algorithm for maximum parsimony phylogeny inference. European Journal of Operational Research, 176(3), 1908-1917.

(Smith et al., 1981): Smith, Temple F.; and Waterman, Michael S., Identification of Common Molecular Sub-sequences, Journal of Molecular Biology, 1981, p. 195-197

(Lipman et al., 1985): Lipman D, Pearson W: Rapid and sensitive protein similarity searches. Science 1985, 227(4693):1435-1441.

(Li et al., 2004): Li M, Ma B, Kisman D, Tromp J: PatternHunterII: Highly Sensitive and Fast Homology Search. J Bioinformatics and Computational Biology 2004, 2(3):417-440.

(Rumble et al., 2009): Rumble SM, Lacroute P, Dalca AV, Fiume M, Sidow A, Brudno M: SHRiMP: Accurate Mapping of Short Color-space Reads. PLoS Comput Biol 2009, 5(5):e1000386.

(Stamatakis 2005): Stamatakis, A. (2005, April). An efficient program for phylogenetic inference using simulated annealing. In Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International (pp. 8-pp). IEEE.

(Ma et al., 2007): Ma B, Li M: On the complexity of the spaced seeds. J Comput Syst Sci 2007, 73(7):1024-1034.

(Notredame et al., 2000): Notredame, C., Higgins, D. G., &Heringa, J. (2000). T-Coffee: A novel method for fast and accurate multiple sequence alignment. Journal of molecular biology, 302(1), 205-217.

(Metzkeret 2010): Metzker, M. L. (2010). Sequencing technologiesâĂŤthe next generation. Nature Reviews Genetics, 11(1), 31-46.

(Notredame et al., 1996): Notredame, C., & Higgins, D. G. (1996). SAGA: sequence alignment by genetic algorithm. Nucleic acids research, 24(8), 1515-1524.

(Onwubolu et al., 2001): Onwubolu, G. C., &Mutingi, M. (2001). A genetic algorithm approach to cellular manufacturing systems. Computers & industrial engineering, 39(1), 144.

(Nocedal et al., 2006): Nocedal, J., & Wright, S. (2006). Numerical optimization. Springer Science & Business Media.

(Polyanovsky et al., 2011): Polyanovsky, V. O. & Roytberg, M. A. & Tumanyan, V. G. (2011). "Comparative analysis of the quality of a global algorithm and a local algorithm for alignment of two sequences". Algorithms for Molecular Biology 6 (1): 25. doi:10.1186/1748-7188-6-25. PMC 3223492. PMID 22032267.

(Stutzle et al., 1999): Stutzle, T., & Dorigo, M. (1999). ACO algorithms for the traveling salesman problem. Evolutionary Algorithms in Engineering and Computer Science, 163-183.

(Lee et al., 2009): Lee, M. E., Kim, S. H., Cho, W. H., Park, S. Y., & Lim, J. S. (2009, June). Segmentation of brain MR images using an ant colony optimization algorithm. In Bioinformatics and BioEngineering, 2009. BIBE'09. Ninth IEEE International Conference on (pp. 366-369). IEEE.

(Parpinelli et al., 2002): Parpinelli, R. S., Lopes, H. S., & Freitas, A. (2002). Data mining with an ant colony optimization algorithm. Evolutionary Computation, IEEE Transactions on, 6(4), 321-332.

(Papadopoulos et al., 2010): Papadopoulos, Y., Walker, M., Reiser, M.-O., Weber, M., Chen , D., TOrngren, M., Sandberg, A.(2010). Automatic Allocation of Safety Integrity Levels. 1st Workshop on Critical Automotive applications: Robustness and Safety (CARS'10) (pp. 7-10). Valencia, Spain: ACM.

(Pavzner 2000): Pevzner, P. (2000). Computational molecular biology: an algorithmic approach. MIT press.

(Parpinelli et al., 2011): Parpinelli, R. S., & Lopes, H. S. (2011). New inspirations in

swarm intelligence: a survey. International Journal of Bio-Inspired Computation, 3(1), 1-16.

(Pant et al., 2008): Pant, M., Thangaraj, R., & Abraham, A. (2008, September). Particle swarm optimization using adaptive mutation. In Database and Expert Systems Application, 2008. DEXA'08. 19th International Workshop on (pp. 519-523). IEEE.

(Rudich et al., 2004): Rudich, S., & Wigderson, A. (2004). Computational complexity theory. American Mathematical Soc.

(Riaz et al., 2004): Riaz, T., Wang, Y., & Li, K. B. (2004, January). Multiple sequence alignment using tabu search. In Proceedings of the second conference on Asia-Pacific bioinformatics-Volume 29 (pp. 223-232). Australian Computer Society.

(Roli et al., 2009): Roli, A., & Blum, C. (2009). Tabu search for the founder sequence reconstruction problem: A preliminary study. In Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living (pp. 1035-1042). Springer Berlin Heidelberg.

(Rego et al., 2006): Rego, C., & Alidaee, B. (Eds.). (2006). Metaheuristic optimization via memory and evolution: tabu search and scatter search (Vol. 30). Springer Science & Business Media.

(Simpson et al., 1976): Simpson, G. G., Simpson, G. G., Simpson, G. G., Paleontologiste, Z., Simpson, G. G., & Palaeontologist, Z. (1976). Penguins: past and present, here and there. Yale University Press.

(Sayers et al., 2009): Sayers EW, Barrett T, Benson DA, Bryant SH, Canese K, Chetvernin V, Church DM, DiCuccio M, Edgar R, Federhen S, Feolo M, Geer LY, Helmberg W, Kapustin Y, Landsman D, Lipman DJ, Madden TL, Maglott DR, Miller V, Mizrachi I, Ostell J, Pruitt KD, Schuler GD, Sequeira E, Sherry ST, Shumway M, Sirotkin K, Souvorov A, Starchenko G, Tatusova TA, Wagner L, Yaschenko E, Ye J (2009). Database resources of the National Center for Biotechnology Information. Nucleic Acids Res. 2009 Jan;37(Database issue):D5-15. Epub 2008 Oct 21.

(Takahashi et al., 2004): Takahashi, A., Sato, K., Nishikawa, J., Watanuki, Y., & Naito, Y. (2004). Synchronous diving behavior of AdÃl'lie penguins. Journal of Ethology, 22(1), 5-11.

(Tremblay et al., 1999): Tremblay, Y., & Cherel, Y. (1999). Synchronous underwater foraging behavior in penguins. Condor, 179-185.

(Valentin et al., 2010): Valentin, F., Squizzato, S., Goujon, M., McWilliam, H., Paern, J., and Lopez, R. (2010)'Fast and efficient searching of biological data resourcesusing ebeye', Briefings in bioinformatics, Vol. 11, No. 4, pp.375-384.

(Varn et al., 1971): Varn, B. (1971) 'Optimal variable length codes- Arbitrary symbol cost and equal code wordprobability', Information and Control, Vol. 19, No. 4, pp.289-301.

(Vazirani 2003): Vazirani Vijay V. (2003). Approximation Algorithms. Berlin: Springer. ISBN 3-540-65367-8.

(Verfaillie et al., 1996): Verfaillie Gerard, Michel Lemaitre, and Thomas Schiex. "Russian doll search for solving constraint optimization problems." AAAI/IAAI, Vol. 1. 1996.

(Yang et al., 2009): Yang, X. S., & Deb, S. (2009, December). Cuckoo search via LÃl'vy flights. In Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on (pp. 210-214). IEEE.

(Yang et al 2010): Yang, X. S. (2010). Firefly algorithm, Levy flights and global optimization. In Research and development in intelligent systems XXVI (pp. 209-218). Springer London.

(Wang et al., 2011): Wang, C., and Zhang, D. (2011) 'A novel compression tool for efficient storage of genome resequencing data', Nucleic acids research, Vol. 39, No. 7, pp.e45-e45.

(Zhao et al., 2010): Zhao, N., Wu, Z., Zhao, Y., &Quan, T. (2010). Ant colony optimization algorithm with mutation mechanism and its applications. Expert Systems with Applications, 37(7), 4805-4810.