

Multiple Sequence Alignment using Particle Swarm Optimization

by

Fabien B. R. Zablocki

Submitted in partial fulfillment of the requirements for the degree
Magister Scientiae (Computer Science)
in the Faculty of Engineering, Built Environment and Information Technology
University of Pretoria, South Africa

November 2007

A research publication of

C I R G
Computational Intelligence Research Group

Visit the research group online at
cirm.cs.up.ac.za

An electronic, hyperlinked PDF version of this work is available online at:

<http://cirm.cs.up.ac.za/thesis/>

A complete, BibTeX format, reference for this work is available online at:

<http://cirm.cs.up.ac.za/>

Multiple Sequence Alignment using Particle Swarm Optimization

by

Fabien B. R. Zablocki

E-mail: fzablocki@cs.up.ac.za

Abstract

The recent advent of bioinformatics has given rise to the central and recurrent problem of optimally aligning biological sequences. Many techniques have been proposed in an attempt to solve this complex problem with varying degrees of success. This thesis investigates the application of a computational intelligence technique known as particle swarm optimization (PSO) to the multiple sequence alignment (MSA) problem. Firstly, the performance of the standard PSO (S-PSO) and its characteristics are fully analyzed. Secondly, a scalability study is conducted that aims at expanding the S-PSO's application to complex MSAs, as well as studying the behaviour of three other kinds of PSOs on the same problems. Experimental results show that the PSO is efficient in solving the MSA problem and compares positively with well-known CLUSTAL X and T-COFFEE.

Keywords: Artificial Intelligence, Bioinformatics, Particle Swarm Optimization, Computational Intelligence, Multiple Sequence Alignment, DNA.

Supervisor : Prof. A. P. Engelbrecht
Department : Department of Computer Science
Degree : Magister Scientiae

“All truth passes through three stages. First, it is ridiculed. Second, it is violently opposed. Third, it is accepted as being self-evident.”

- Arthur Schopenhauer

“Research is what I’m doing when I don’t know what I’m doing.”

- Wernher Von Braun

Acknowledgements

I would like to express my sincere thanks to the following people for their assistance during the production of this work:

- Prof. Andries P. Engelbrecht for his valuable guidance through the process of achieving the work presented in this thesis.
- Roman Zablocki for his contagious motivation, constant supportive presence and the laptop I used to finish this thesis.
- Nadine Noel for believing in me in a very special mother's way.
- Dr. Fourie Joubert and Prof. Andries P. Engelbrecht for making it possible for me to attend the three-day seminar at the bioinformatics department.
- Techteam and particularly Ian Galpin for providing me with machine support.
- All my friends, inside and outside CIRG, who supported me.
- Last, but not least, everyone who helped proofreading my writing which made the production of this final version possible, in particular Vanessa Smeets, Nelis Franken and Roland Canivet.

This thesis was typeset exclusively with the use of standard L^AT_EX 2_ε. Bibliographic references were maintained using B_IB_TE_X, while T_EX_{sh}ade [13] was used to typeset alignments.

Contents

List of Figures	v
List of Algorithms	vii
List of Tables	viii
1 Introduction	1
1.1 Problem Statement and Overview	2
1.2 Objectives	2
1.3 Contributions	3
1.4 Thesis Outline	4
2 Multiple Sequence Alignment	6
2.1 Introduction	6
2.2 Biological Aspects	7
2.2.1 The Code of Life	7
2.2.2 Life Origins	9
2.3 Definitions, Notations and Examples	10
2.3.1 Building Blocks	10
2.3.2 Alignment Evaluation	14
2.3.3 Scoring Enhancement	16
2.4 Use of Sequence Alignments	19
2.4.1 Computer-Aided Molecular Biology	20
2.4.2 Briteny Spears	22
2.5 Where Does the MSA Problem Lie?	23

2.5.1	Choice of Sequence	23
2.5.2	Is This Alignment Any Good?	23
2.5.3	Computer Complexity	24
2.6	State of the Art	25
2.6.1	The <i>Exact</i> Approach	26
2.6.2	The <i>Progressive</i> Approach	27
2.6.3	The <i>Iterative</i> Approach	28
2.6.4	The <i>Consistency-based</i> Approach	29
2.7	Summary	30
3	Computational Intelligence Paradigms	31
3.1	Introduction	31
3.2	Classical Optimization	32
3.2.1	Minimization Definition	32
3.2.2	NP-Complete Problems	33
3.3	Genetic Algorithms	34
3.3.1	Genetic Algorithm Essentials	34
3.3.2	Evolutionary Operators	36
3.3.3	Cooperative Coevolutionary Genetic Algorithm	38
3.4	Particle Swarm Optimization	39
3.4.1	Algorithm Essentials	39
3.4.2	Neighbourhood Topologies	41
3.4.3	PSO Parameters	43
3.4.4	Modifications to PSO	47
3.5	Swarm Intelligence versus Evolutionary Computation	52
3.6	Summary	53
4	Particle Swarm Optimization for Multiple Sequence Alignment	54
4.1	Introduction	54
4.2	Representation Schemes	54
4.2.1	Integer Search Space	56
4.2.2	Binary Search Space	56
4.3	Fitness Evaluation	57

4.4	Summary	58
5	Empirical Analysis	59
5.1	Introduction	59
5.2	S8 Analysis	60
5.2.1	DNA Data Set S8	61
5.2.2	Characteristic Selection	62
5.2.3	Experimental Procedure	63
5.2.4	Experimental Results for S-PSO	64
5.2.5	Experimental Results for B-PSO	79
5.2.6	Conclusion	84
5.3	Scalability Analysis	85
5.3.1	MSA Data Sets	85
5.3.2	PSO Algorithms and Configurations	87
5.3.3	Experimental Procedure	89
5.3.4	Experimental Results	90
5.3.5	Comparison with Other MSA Programs	96
5.3.6	Conclusion	100
5.4	Summary	101
6	Conclusion and Future Work	102
6.1	Conclusion	102
6.2	Future Work	105
	Bibliography	107
A	Data Sets	120
B	Cilib XML Configuration	122
C	Acronyms & Abbreviations	128
D	Symbols	131
E	Alphabets and S. Matrices	135

List of Figures

2.1	The DNA double helix model	8
2.2	Steps for protein synthesis	9
2.3	The alignment and consensus of two DNA sequences	13
2.4	The alignment and consensus of five protein sequences	13
3.1	Three cross-over operators for bit representations	36
3.2	Common PSO topologies	42
5.1	Visualization of DNA data set S8	62
5.2	Best alignment of S8 optimized by the OF using the similarity method .	68
5.3	Influence of weights ξ_1 and ξ_2 on S8 (similarity method)	69
5.4	Best alignment of S8 optimized by the OF using the matches method .	73
5.5	Influence of weights ξ_1 and ξ_2 on S8 (matches method)	73
5.6	Visualization of aligned data set S8 by CLUSTAL X	75
5.7	Visualization of aligned data set S8 by T-COFFEE	76
5.8	Alignment of S8 (best similarity variation) – first iteration	77
5.9	Alignment of S8 (best similarity variation) – 50 th iteration	77
5.10	Alignment of S8 (best similarity variation) – 100 th iteration	77
5.11	Alignment of S8 (best similarity variation) – 150 th iteration	78
5.12	Alignment of S8 (best similarity variation) – 200 th iteration	78
5.13	Alignment of S8 (best similarity variation) – 250 th iteration	78
5.14	Final alignment of S8 with similarity method (55.0) – last iteration (272 th)	78
5.15	Progression in sub-objectives for S8	79
5.16	Visualization of the best S8 alignment for the similarity method	81
5.17	Visualization of the best S8 alignment for the matches method	83

5.18	Number of full columns aligned in S2 with three PSOs	92
5.19	Number of full columns aligned in S4 with three PSOs	94
5.20	Number of full columns aligned in S5 with three PSOs	95
5.21	Number of full columns aligned inaligned in S6 with three PSOs	96
5.22	Best score comparison between PSO, T-COFFEE and CLUSTAL X	98
E.1	The PAM250 S. matrix	137
E.2	The BLOSUM62 S. matrix	138

List of Algorithms

3.1	Pseudocode outline of the genetic algorithm	35
3.2	The standard PSO algorithm	41
3.3	The binary PSO algorithm	48
3.4	The mutating PSO algorithm	49
3.5	The cooperative split PSO algorithm	51

List of Tables

2.1	Score produced according to the SoP function	16
2.2	Score produced according to the linearly scaled match count	17
5.1	Properties of DNA data set S8	61
5.2	Performance table for S-PSO aligning S8 - similarity method	66
5.3	Performance table for S-PSO aligning S8 - matches method	70
5.4	Comparison of S8 alignments: S-PSO vs. two common MSA programs .	75
5.5	Results for B-PSO aligning S8 - similarity method	80
5.6	Results for B-PSO aligning S8 - matches method	82
5.7	Properties of the seven new data sets (S1 through S7)	85
5.8	Split factors for the four selected data sets	89
5.9	PSO performance on hard data sets (similarity method)	91
5.10	PSO performance on hard data sets (matches method)	92
5.11	Results from T-COFFEE and CLUSTAL X on seven data sets	97
E.1	Residue alphabet for DNA	135
E.2	Residue alphabet for RNA	135
E.3	Residue alphabet for amino acids (proteins)	136
E.4	A typical DNA identity matrix	137

Chapter 1

Introduction

“An idea that is not dangerous is unworthy of being called an idea at all.”

- Oscar Wilde

Specialized data banks all over the world offer curated biological data of high quality. The amount of genomic information stored from various biological sequences has grown exponentially in recent years and has become extensive. This sudden rise in biological data has led to the need for automated tools and powerful processing devices. The blooming field of bioinformatics has opened the door to make the most of what the combination of biology and computers has to offer.

Directly emanating from the bioinformatics discipline, a central and fundamental task has come to surface: aligning molecular sequences. Undoubtedly, analyses such as recreating phylogenetic trees, finding homologies or detecting protein functions are all achieved by first applying some degree of sequence alignment. Consequently, sequence alignment has become a major tool in molecular sequence analysis.

In recent years, computational intelligence (CI) [38], a sub-discipline of artificial intelligence (AI), has provided systems that are capable of adaptive behaviour within changing environments. The most notable algorithms have been based on a variety of natural systems, ranging from ant colonies [33] and bird flocks [62], to the human neurological [95] and immune [30] systems. One approach in particular, particle swarm optimization (PSO) [62], has proved to be successful in a wide range of optimization

problems [39, 64]. This thesis investigates the application of PSO algorithms as a viable approach to solving the problem of aligning multiple molecular sequences.

1.1 Problem Statement and Overview

Multiple sequence alignment (MSA) is in essence a double objective optimization problem: Gaps have to be inserted into the original sequences in such a way that (1) the number of matching characters is maximized and (2) the number of gaps inserted is minimized. Bearing in mind that the two requirements are conflicting, an optimal alignment solution cannot always be found.

Many previous sequence alignment techniques have been employed, such as tree-based algorithms [3] which combine results from pairwise alignments. The main problem with these algorithms is that they assume the existence of a tree that correctly describes the relationships between sequences. Because such trees cannot always be derived or calculated, a shortage still remains in the techniques available to solve the MSA problem. Most of the commonly used MSA methods [113] are based on dynamic programming (DP) [80]. However, DP requires time and memory proportional to the product of the sequence lengths. Hence, many heuristic methods [67, 82] have been developed to find good alignments, which are not necessarily optimal, within a reasonable time.

1.2 Objectives

The main objective of this thesis is to conduct an empirical analysis of PSO as a new iterative approach to solving the MSA problem. In this research, the applicability of the PSO family of algorithms as a successful sequence aligning optimizer is investigated. Several other techniques, such as T-COFFEE [84] and CLUSTAL X [112], are compared with PSO. Advantages and disadvantages of these methods are discussed, and ways to improve MSA using PSO are explored.

In fact, PSO algorithms have never been applied in the manner described in this thesis. This thesis therefore represents a leading experiment of its kind.

The primary objectives of this thesis are summarized as follows:

- To provide an acquaintance with the concept of biological sequence alignment, with an emphasis on MSAs.
- To provide an overview of the relevant CI techniques that would serve as valuable candidates for solving MSA problems.
- To study the effects and performance of optimizing different sequence alignment objective functions (scoring schemes).
- To conduct a scalability analysis by observing the impact of PSO on MSA problems that differ by their complexity (number of sequences in the set, length of sequences and overall similarity).
- To investigate the performance of four variations of PSOs, namely the standard PSO, the binary PSO, the mutating PSO, and the cooperative split PSO, as applied to the MSA problem.
- To compare solutions from PSOs with solutions from other MSA programs in order to position PSO's performance with respect to several factors, such as alignment accuracy.

1.3 Contributions

The novel contributions of this work include the following:

- A novel application of swarm intelligence to the field of bioinformatics dealing with sequence alignment.
- The derivation of two representation schemes for the MSA problem, namely binary-valued and integer-valued.
- The introduction of a gap-reducing factor based on overall similarity in order to lower the complexity in MSAs.
- The development and analysis of new ways of optimizing sequence alignment objective functions.

- The implementation of a generic MSA package within the computational intelligence library (Cilib¹) framework [91].

1.4 Thesis Outline

The list below presents the organization of the chapters which make up this thesis. Also given is a brief description of the topics each chapter deals with.

- **Chapter 2** covers the necessary background relating to methods designed to align molecular sequences. This chapter includes a theoretical basis on sequence alignment, building blocks, concepts, uses and current alignment methods.
- **Chapter 3** provides an overview of population-based optimization methods. This chapter covers genetic algorithms (GA) and particle swarm optimization (PSO) algorithms in detail.
- **Chapter 4** describes how PSO can be applied to MSA. Representation schemes and the fitness evaluation mechanism are given.
- **Chapter 5** provides an empirical analysis of PSO solving the MSA problem. The core of this thesis resides in this chapter. All aspects relating to the experimental procedure are covered, comprising mainly the experimental methodology, description of experiments, experiments themselves and analysis of the results.
- **Chapter 6** provides a summary of the findings of this thesis, and considers possible future research that emanates from it.

The following appendices are also included. They contain supplementary material related to the main text of this thesis, as well as a number of lists containing relevant information for quick referencing purposes.

- **Appendix A** offers complementary information about all data sets used throughout the experimental procedure.

¹Cilib is freely available at <http://cilib.sourceforge.net>

- **Appendix B** lists XML code snippets that were used to generate simulations within CLib.
- **Appendix C** provides a list of the acronyms used in this work, as well as their meanings.
- **Appendix D** alphabetically lists and defines the mathematical symbols used in this work.
- **Appendix E** shows tables for all correspondences between names of the DNA, RNA and proteins residue alphabets. Substitution matrices for both PAM and BLOSUM are also displayed.

Chapter 2

Multiple Sequence Alignment

“Man is still the most extraordinary computer of all.”

- John F. Kennedy

There is much to learn from biological sequences and much to do in order to learn from them. However simple a few DNA sequences may seem at first, the intrinsic information contained in them is phenomenal and complex. Since biological sequences represent the essential blueprints of any living species, it is a crucial and fundamental task to study those sequences. This chapter provides detailed material on the theory behind sequence alignment, and more particularly focuses on the most common task in bioinformatics today, namely multiple sequence alignment [19].

2.1 Introduction

The continuous motivation to better understand how the human body works has always been a priority in the biological sciences. Amongst the flourishing research in the field of bioinformatics, recent techniques like multiple sequence alignment (MSA) have allowed researchers to accomplish what is claimed to be the greatest achievement in scientific history. Better known as the human genome project (HGP) [85], the venture tackled the gargantuan task of sequencing the entire human genome¹. At the heart of

¹The human genome comprises 23 pairs of chromosomes, each of which is made up of several thousand genes, with a grand total of just over 3 billion nucleotides.

such an achievement lies the ability to extrapolate information from related sequences. The development of automated techniques for sequence analysis has therefore become preponderant in bioinformatics.

Researching new tools to perform sequence comparisons in a reasonable amount of time has been in the spotlight for a number of years [81]. Along with the birth of advanced techniques came the creation of biological databases, which are doubling in size every year.

The aim of this chapter is to provide an overview of the current landscape of MSA research. Furthermore, the reader will become familiar with all the concepts and notions related to the topic. Section 2.2 gives a brief biological background on both nucleic and amino acids, while Section 2.3 provides formal definitions and notations. As a core discipline of bioinformatics, MSA has many uses, which are discussed in Section 2.4. Next, Section 2.5 discusses why no single technique is best suited for solving MSA problems. For completeness, Section 2.6 surveys existing sequencing techniques. Finally, the chapter is summarized in Section 2.7.

2.2 Biological Aspects

What is so interesting in biological cells that cannot be seen? To the naked eye, nothing. A journey on a microscopic level must be undertaken to discover the invisible. The next two sections provide answers from a scientific perspective to the fundamental questions: *Where do we come from? And, why are we what we are?*

2.2.1 The Code of Life

Nucleic acids such as *deoxyribonucleic acid* (DNA) and *ribonucleic acid* (RNA) represent the chemical carriers of a cell's genetic information [9]. All the information that determines the *phenotype* and *genotype* of a cell is coded in its DNA. In short, DNA represents the building blocks of life and, as such, also controls cell growth and division and arbitrates biosynthesis of the enzymes, as well as other proteins required for all cellular functions. Nucleic acids are made up of *nucleotides* bonded together to form a long chain (or *sequence*) in the form of a double helix, as illustrated in Figure 2.1.

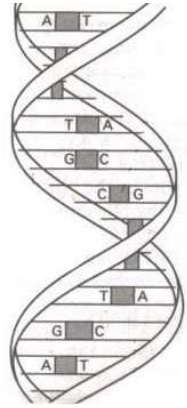


Figure 2.1: The famous Watson-Crick model of the DNA double helix, adapted from the *Oxford Dictionary of Biology* [52]

However, the notion of sequences is an artificial one as it only serves the purpose of being a conceptual entity. In reality, all these molecules take the form of an intricate three-dimensional shape.

From a chemical point of view, RNA and DNA are very similar. The main differences lie in their respective sizes and roles within the cell. Molecules of DNA are gigantic: they have a molecular weight of up to 150 billion and lengths of up to 100 centimetres when stretched out [25]. DNA molecules are mostly found in the nuclei of cells. By contrast, RNA molecules are much tinier, with a molecular weight of only 35,000, and generally reside outside the cell nucleus.

The function of DNA is to store an organism's blueprint and pass it on to RNA. In turn, the role of RNA is to read, decode and use the information received from DNA to make proteins. Proteins are chains of amino acids and serve as instruments in virtually everything organisms do. Humans have thousands of different proteins, each having a specific structure along with a unique function. As a matter of fact, proteins are the most structurally sophisticated molecules ever identified [69].

The procedure of transferring genetic information comprises the following three fundamental steps [18], as illustrated in Figure 2.2:

1. **Replication** is the action by which exact copies of DNA are made, so that information can be safeguarded and passed on to offspring.

2. **Transcription** is the action by which the genetic messages are read and carried out of the cell nucleus, where protein synthesis occurs.
3. **Translation** is the action by which the genetic messages are decoded and employed to synthesize proteins.

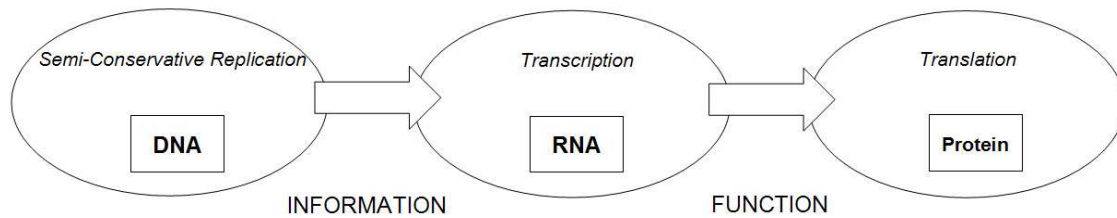


Figure 2.2: Steps for protein synthesis

2.2.2 Life Origins

Darwinian theories [28] promote a natural evolutionary process. Under the umbrella of evolution, the whole process can be interpreted as a race to become the best, from which the term ‘survival of the fittest’ was coined. From time immemorial, within any population the least fit individual dies fast, while the fittest one lives longer. Through many cycles of generations following the evolutionary concept, only those individuals that evolved to become more adapted to the environment remain.

As a result of evolution, several ‘changes’ have taken place within individuals. Changes are represented by *mutations* within DNA sequences. Mutations in a sequence refer to the random occurrence of genetic transformations under the influence of external and chemical factors. Transformations may be expressed as a combination of either *insertions*, *deletions* or even *substitutions* of nucleotide pairs within a DNA sequence [18]. As a result, the mutation phenomenon is at the origin of species evolution. It is precisely these mutations that gave rise to the task of aligning sequences.

The process of aligning sequences does not have the goal of reversing the effect of evolution. On the contrary, the objective is to discover what relationship *remains*. It is

then necessary to analyze both evolutionary similarities and differences in order to find out whether several sequences have a common origin or not. A wide range of parameters, such as ageing and species relatedness, may influence mutations and can thus render the aligning task quasi-impossible. Nevertheless, in cases where a high similarity percentage can be found, evolutionary scenarios can be reconstructed. Such scenarios make extensive use of phylogenetic trees to illustrate paths from descendants to their ancestors.

Sequences may be either closely or loosely related. Three degrees of relatedness between a pair of sequences exist (which can be generalized to more than two sequences) [9]:

- **Identity:** two sequences are said to be identical if all characters from one sequence match all the characters in the other sequence.
- **Similarity:** two sequences are said to be similar if they are relatively close without being identical.
- **Homology:** this is a special case of similarity, where two similar sequences are considered to be coming from the same root. This means that the sequences are mutated sequences originating from a *common* initial sequence.

2.3 Definitions, Notations and Examples

As for every concept, it is necessary to define the topic in a formal way. This section presents theoretical background on sequence alignment.

2.3.1 Building Blocks

In an effort to better understand the action of aligning sequences, consider the following formal definition: The *Oxford Advanced Learner's Dictionary* proposes the following meaning of the verb 'align' [119]: 'to change something slightly so that it is in the correct relationship to something else.' Attention should be given here to the fact that something has to be changed, but only slightly.

Since the task is specifically about aligning *sequences*, it means that sequences must therefore be changed slightly in order to be in the correct relationship with the other sequence(s). To understand how the sequences must be changed, it is crucial to expose

how sequences are defined. Firstly, the notion of an *alphabet* is provided:

Definition 2.1 Alphabet: *An alphabet, \mathcal{A} , is a finite set of arbitrary symbols.*

In molecular biology, two different alphabets (see Appendix E for symbol references) are used to represent sequences:

1. A **nucleic acid alphabet** to model DNA sequences: $\mathcal{A}_{DNA} = \{A, C, T, G\}$. Similarly, the RNA alphabet is also composed of four symbols: $\mathcal{A}_{RNA} = \{A, C, U, G\}$, where the symbol U replaces T.
2. The **protein alphabet**, which consists of 20 different amino acids, as follows:
 $\mathcal{A}_{Protein} = \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y\}$.

Using these alphabets it is possible to construct sequences where a sequence is defined as:

Definition 2.2 Sequence: *A sequence, \mathcal{S} , is a series of ordered symbols taken from a specific alphabet. That is,*

$$\mathcal{S} = \{s_1, s_2, \dots, s_n\}$$

where s_p or $\mathcal{S}[p]$ is the p^{th} symbol in the sequence \mathcal{S} . The length, n , of a sequence is denoted by $|\mathcal{S}|$.

Definition 2.3 Sub-sequence: *A sub-sequence, $\mathcal{S}[p, q]$, from sequence \mathcal{S} is a sequence of symbols from the p^{th} position to the q^{th} position inclusive.*

Sequences must be aligned in order to extract meaningful information about homology. A sequence alignment is the adaptation of two or more sequences in a way that highlights their relationship. More precisely, the result obtained from the alignment is the identification of conserved zones (identical sub-sequences) within sequences. In other words, the level of similarity between involved sequences is determined after a successful alignment.

The core process of aligning consists of *inserting* a minimum number of *gaps* represented by ‘–’ in the sequences so that the columns contain a maximum number of

matching symbols (each row of symbols represents one sequence). It is important to note that during the course of this work the meanings of the terms ‘symbol’ and ‘character’ are equivalent and are used interchangeably. Formally, sequence alignment is defined as follows:

Definition 2.4 Sequence Alignment: *An alphabet $\hat{\mathcal{A}} = \mathcal{A} \cup \{-\}$ is defined. Let $\mathcal{T} = \{\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_k\}$ be a set of k sequences. Then, an alignment of the set \mathcal{T} results in a new set $\hat{\mathcal{T}} = \{\hat{\mathcal{S}}_1, \hat{\mathcal{S}}_2, \dots, \hat{\mathcal{S}}_k\}$ of sequences over the alphabet $\hat{\mathcal{A}}$.*

An alignment is considered to be valid if and only if all the following properties are verified:

1. All aligned sequences, $\hat{\mathcal{S}}_i$, must be of equal length, \hat{n} . That is,

$$\hat{n} = \max\{|\hat{\mathcal{S}}_i|\}, \quad \forall \hat{\mathcal{S}}_i \in \hat{\mathcal{T}} \quad (2.1)$$

2. $\forall i \in \{1, 2, \dots, k\}$: $\hat{\mathcal{S}}_i$ reverts to the original \mathcal{S}_i by removing all the inserted gap characters.
3. Columns which only consist of gap characters must be removed.

From a mathematical point of view, a sequence alignment can be represented as a matrix with k rows of length \hat{n} , where the i^{th} row contains a sequence $\hat{\mathcal{S}}_i$. In this work, the general matrix structure will be used for both manipulating and displaying alignments (see [77] for a thorough mathematical definition of sequence alignment). Two types of sequence alignments are recognized:

- A *global* sequence alignment of k sequences represents an alignment involving all characters in these sequences. If not mentioned, a sequence alignment is global (by default). Global sequence alignment is the most widely used alignment type in practice.
- A *local* sequence alignment is a specialization of global alignment, in which the best alignment of a sub-sequence within entire sequences or sub-sequences is found. Local sequence alignments offer the flexibility to locate specific related regions within sequences.

Sequence alignments can be classified further according to the number of participating sequences, k . An alignment is referred to as a *pairwise sequence alignment* (PSA) when $k = 2$. In this case, the alignment process consists of searching for a series of the best matching $|\widehat{S}|$ pairs. A *pair* is made up of one character from the first sequence \widehat{S}_1 and its corresponding character from the second sequence \widehat{S}_2 .

For all the other cases where $k \geq 3$, the expression *multiple sequence alignment* (MSA) is used. In short, MSA is simply a logical expansion of PSA which includes more than two sequences in the alignment. However, it is not necessarily true that one MSA is constructed from several PSAs. As a result, the techniques for processing PSAs and MSAs are not always the same. During the course of this work, the focus will mainly be on global MSA. Figure 2.3 illustrates an example of a possible pairwise DNA sequence alignment.

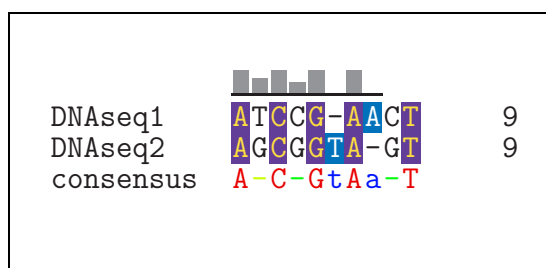


Figure 2.3: The alignment and consensus of two DNA sequences

Figure 2.4 illustrates what a multiple protein sequence alignment looks like.

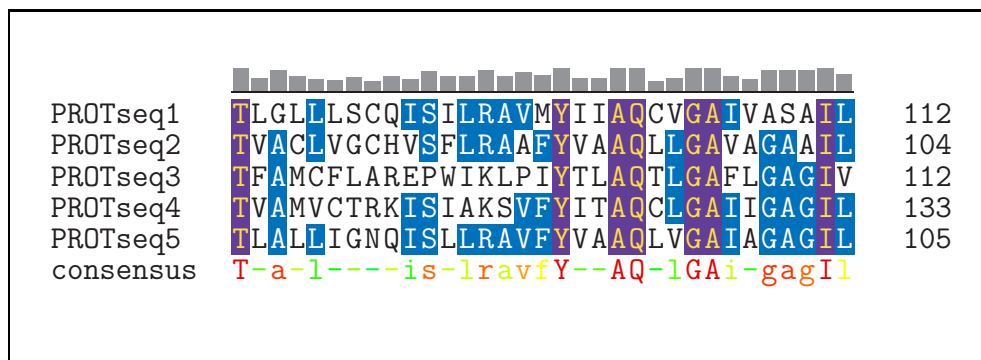


Figure 2.4: The alignment and consensus of five protein sequences

Figures 2.3 and 2.4 have the following properties:

- a header name (e.g. DNaseq1 or PROTseq3) for sequence identification;
- a per column highlight illustrating best character similarity;
- an index, at the end of each line, which stops at the last character of the aligned sequence;
- a bar chart, above each column, giving a quick view of the similarity level (higher is better); and
- a consensus sequence at the bottom. The consensus constructs a representative sequence by taking per column the character with highest similarity. An upper case letter indicates maximum similarity (entire column of identical characters). A lower case letter indicates intermediate degrees of similarity, while ‘—’ represents no similarity.

2.3.2 Alignment Evaluation

From a general point of view, a sequence alignment can be seen as a morphing operation with the objective to make two or more sequences as *similar* as possible. In order to find a good alignment, a kind of measurement is needed to *quantify* which pairs of characters are better than others. A numerical value given to quantify the ‘goodness’ of an alignment is called the *score* of an alignment. Two popular scoring schemes are now provided.

The Similarity Method

A ‘similarity’ approach [97] suggests the following to determine how alike sequences actually are. If, for each possible pair, an arbitrary positive value or *cost* is assigned to an atomic mutation/edit operation (e.g. substitution, insertion or deletion), then a minimization of the costs will lead to sequences of higher similarity. Conversely, instead of minimizing the dissimilarity, an alignment could be scored by maximizing the level of similarity. An effective way of achieving an alignment maximization is by rewarding matching characters while penalizing gaps and mismatches.

Let a gap represent a character deletion or a character insertion (i.e. holes in either of the sequences). Additionally, let a mismatch represent a character substitution (character has mutated). The score, w , is computed by adding a cost assigned to each set of pairs, using the common *sum of pairs* (SoP) formula defined as

$$w(\hat{T}) = \sum_{i=1}^{k-1} \sum_{j=i+1}^k \text{Cost}(\hat{S}_i, \hat{S}_j) \quad (2.2)$$

To clarify, the SoP scores all possible pairwise combinations in aligned sequences, which requires $\frac{k(k-1)}{2}$ comparisons for k sequences. Then, the total score is the sum of scores obtained over all pairs of sequences. Thus, the alignment problem can be stated as the search for an alignment, \hat{T} , that maximizes its score, defined as

$$w^*(\hat{T}) = \max \sum_{i=1}^{k-1} \sum_{j=i+1}^k \text{Cost}(\hat{S}_i, \hat{S}_j) \quad (2.3)$$

The optimization process is inevitably driven by the costs. *But, what cost is best suited for comparisons?* As previously stated, a reward is given for a match, while anything else is penalized. A cost is given according to the nature of the pair in question. Particular attention must be given to both gap and character substitution costs because they can corrupt the score maximization if chosen wrongly.

A plain model for character comparison, with its three possible scoring cases for pairs, along with their arbitrary cost, is established as follows [97]:

- Character match: The reward for matching characters is +2. A match occurs when a pair contains identical non-gap characters.
- Character mismatch (substitution): When a mismatch occurs, zero penalty is given. An evolutionary mutation into another non-gap character has occurred and is not rewarded. Comparatively, a mismatch must be rewarded less than a match.
- Indel (INsertion/DEletion): For an indel, a penalty of -1 is assigned. An indel is a special case of character mismatch that involves a gap. An indel penalty, also referred to as a gap penalty, GP , prevents unrelated ‘gappy’ sequences to achieve high scores. Lastly, this penalty must be (1) higher than the character mismatch

penalty because, biologically, a mutation is more probable and desirable than a sequence being cut by a gap; and (2) not severe enough to cancel out a match reward.

The model is illustrated in Table 2.1. A SoP score is computed on the alignment previously shown in Figure 2.3. The total score is derived by adding up all the arbitrary pairwise costs per column.

Table 2.1: Score produced according to the SoP function

<i>DNAseq1</i> :	A	T	C	C	G	-	A	A	C	T	<i>Total</i>
<i>DNAseq2</i> :	A	G	C	G	G	T	A	-	G	T	<i>score</i>
<i>Scores</i> :	<i>+2</i>	<i>0</i>	<i>+2</i>	<i>0</i>	<i>+2</i>	<i>-1</i>	<i>+2</i>	<i>-1</i>	<i>0</i>	<i>+2</i>	<i>= 8</i>

Match Counts

Chellapilla and Fogel [22] proposed a method where the goal is ultimately to get fully matched columns. Matches are taken into account to compute the score (gaps do not count at this stage as they are penalized separately). The principle relies on counting solely the number of matched symbols over all columns and linearly scaling them. The effect of counting the number of matches over columns rather than pairwise (as in SoP) promotes growing more matches per column. Growing matches per column is obtained by linearly scaling the number of matches in each column so that if a column is fully matched, the score is doubled. The total score is computed using

$$w(\hat{\mathcal{T}}) = \sum_{c=1}^{\hat{n}} M_c \left(1 + \frac{M_c}{k}\right) \quad (2.4)$$

where M_c is the number of matches in the c^{th} column. The example in Table 2.2 illustrates the match count measure.

2.3.3 Scoring Enhancement

In most cases, the simplistic similarity model described above results in good approximate alignments [111]. An alignment process may also be adjusted by modifying the

Table 2.2: Score produced according to the linearly scaled match count

<i>DNAseqA</i> :	A	T	C	C	<i>Total</i>
<i>DNAseqB</i> :	-	T	-	G	<i>score</i>
<i>DNAseqC</i> :	A	T	T	G	
<i>Scores</i> :	1.34	6.00	0.00	1.34	= 8.68

cost of a specific edit operation. However, the model needs to be further improved in a way that reflects other biological constraints. An example of a biological constraint is defining a more realistic gap repartition, which tends to reflect the way natural indels occur. The SoP function with fixed arbitrary values performs reasonably well when dealing solely with DNA/RNA sequences, but soon becomes limited when aligning protein sequences that have more properties [81]. In effect, protein sequences embody biochemical properties which greatly influence their respective evolutionary substitution probability. A valid protein alignment cannot be obtained when these important affinity factors are ignored. Therefore, a scoring scheme that takes the evolutionary aspects into account must be used. A brief presentation of potential scoring enhancements follows.

Penalizing Gaps

The first scoring enhancement aims at refining the way that gaps are penalized. A biologically more accurate way of producing gaps in a sequence is to have κ contiguous indels appearing at once, which is more plausible from an evolutionary perspective than successively getting κ single indels. Hence, gaps are rather defined as *groups* of κ contiguous indels, as gaps usually tend to appear under the form of many indels at once. Consequently, instead of using a length independent, *linear* gap penalty model where

$$lGP(\kappa) = \kappa(g_{fixed}) \quad (2.5)$$

with g_{fixed} as the fixed indel penalty, the more accurate *affine* gap penalty model is broadly preferred. This length dependent, affine gap penalty – with $O(\hat{n}^2)$ complexity – is defined as

$$aGP(\kappa) = g_{op} + (\kappa - 1)g_{ext} \quad (2.6)$$

where g_{op} is the penalty for opening a gap (i.e. always the first indel) and g_{ext} is the penalty for extending a gap (second indel and so forth). A higher penalty is usually assigned for opening a gap because it is less difficult to extend an existing gap than to open a new one. Usually, there will be a one-to-many ratio between the opening and extending gaps, which justifies a higher penalty for opening gaps and encourages gap extension rather than introduction. Finally, a completely different way of penalizing gaps consists in linearly scaling the number of gaps over all columns [22].

Sequence Weight

Moreover, an attempt to gain a higher level of biological accuracy promotes that pairwise scores of aligned sequences may be proportionally *weighted* [5, 102] according to the amount of unique information enclosed in the sequence. These weights try to decrease the influence of redundant information from strongly related sequences. A weight represents a percentage equal to a *percentage identity* (PID) calculated over each pair of aligned sequences as follows (excluding gaps):

$$PID = \frac{matches}{matches + mismatches} \quad (2.7)$$

Protein Scoring Matrices

Next, the concept of a *scoring matrix* is introduced. A scoring matrix is a fixed table filled with pre-calculated values designed to reflect affinities between known acids (amino and nucleic). Typically, protein sequence alignments benefit from scoring (or *substitution*) matrices because they reflect the relative likelihood of all possible amino acid substitutions, which amounts to 210 in total. As a result, matches and mismatches throughout protein alignments can be scored in a more biologically accurate way. When appropriate, DNA sequence alignments may also benefit from a substitution matrix called an *identity* matrix (see Table E.4 in Appendix E). For proteins, different types of matrices exist, with each one following a different evolutionary scheme. It is therefore crucial to choose carefully which matrix to use to acquire the desired alignment. Types of protein scoring matrices include:

- **PAM (point accepted mutation per 10^8 years) matrices** [29] have been designed for global protein alignments. Scores in PAM matrices have been devised

according to the study of how frequently specific amino acid substitutions occur during evolution. PAM matrices range from 1 to 500, where the number indicates the maximum divergence percentage of sequences that the scores have been calculated from. For example, the PAM250 matrix is the most widely used and is considered a good general matrix for scoring distantly related protein sequences (250% divergence assumed). See Table E.1 in Appendix E for a PAM matrix example.

- **BLOSUM (blocks substitution matrices)** [49] have scores derived from observations of the frequencies of substitutions in blocks of local alignments, which makes BLOSUM generally best suited for local sequence alignments. BLOSUM range from 1 to 100, where the number specifies, unlike PAM matrices, the minimum identity percentage of sequences the scores have been calculated from. As an example, the BLOSUM62 is best suited for aligning protein sequences for which no prior knowledge exists as well as detecting weak protein similarities (no less than 62% identity). See Figure E.2 in Appendix E for a BLOSUM example.
- The **Gonnet matrix** [46] has been developed through exhaustive pairwise alignment on protein sequence databases. It is the recommended matrix for a preliminary protein sequence alignment.

From the above description, it is clear that choosing the most appropriate scoring matrix lies in setting the best PAM or BLOSUM number according to the level of relatedness from the set of protein sequences to be aligned. A final note on the use of scoring matrices consists of the following easy guidelines:

- For closely related sequences, use a high number BLOSUM and low PAM.
- For distantly related sequences, use a high number PAM and low BLOSUM.

2.4 Use of Sequence Alignments

The process of aligning entails iterative comparisons. In everyday life, decisions are often taken upon the result of making comparisons, for example, comparing prices when shopping, comparing which tool is most suitable to dig a hole, or comparing two images

at work. The action of comparing choices only requires some memory to store previous states in order to recall these states when necessary. The output of a comparison is simple: it can be either a match or a mismatch (same or different). Then, when the result is different, it can be either better or worse. So, for purposes of comparison, two complementary actions take place: finding differences as well as similarities². Since comparing ‘things’ seems omnipresent, a wide range of applications has emerged from several disciplines. The subsequent sections emphasize the important uses of computers based on alignment.

2.4.1 Computer-Aided Molecular Biology

Computational biology is rich in applications [68]. It involves many branches of genetics including structural genomics, comparative genomics and proteomics, which rely extensively on MSA. But, what exactly is the use of MSAs? In bioinformatics, MSA is used to study properties of related biological sequences. Given two sequences, if the properties of only one of the two sequences are known, then after a successful alignment, similar properties may be *inferred* from the known sequence over the unknown sequence. More precisely, relationship properties are studied through the following categories:

- Detection of protein functions.
- Identification of conserved zones/motifs/domains within multiple related sequences to predict further functional and structural properties.
- Classification of new sequences within specific families through extrapolation from known patterns.
- Contribution to the prediction of the secondary and even tertiary structure of proteins.
- Discovery of protein behaviours with regard to hydrophobicity as well as hydrophilicity.

²The English language contains the verb ‘to differentiate’ but, interestingly, does not have the verb ‘to similarize’.

- Tool for biological database searching. Given some input sequences, \mathcal{T} , the process performs an exhaustive search on the entire database contents. The way to achieve this is to align \mathcal{T} locally with one sequence after another (i.e. pairwise alignment). Depending on the alignment scores obtained throughout the search, a similarity ranking can be established. Top rankings would then yield the closest related sequences found. Examples of database searching tools include FASTA [89, 90] and BLAST (Basic Local Alignment Search Tool) [4, 6], while PROSITE [11] or Pfam [12] are popular types of biological databases.
- Study of phylogeny or evolution of organisms. The aim here is to build phylogenetic trees based on the evolutionary relationships between species. In order to infer a particular homology (e.g. lineage, shared origins or common ancestors), MSAs are used to establish similarity scores, which in turn guide the clustering process necessary to construct the tree.
- Design of PCR (polymerase chain reaction) primers [1] can be enhanced through the localization of strongly conserved zones, which are useful to clone new members of a specific family.
- Lastly, the determination of the consensus sequence from aligned sequences, as illustrated in Figures 2.3 and 2.4.

Diverse but worthy applications, including DNA analysis which is heavily relied on in forensics and the investigation of crime scenes, form an important part of the field of bioinformatics. Positive victim identification through DNA tests have the power to validate a verdict for lawsuits. Some cases even demonstrate that wrongly accused individuals are proven innocent and are freed from prison as a result of DNA post-authentication. Exoneration through DNA tests also occurs in cases where paternity or other family relationships have required identification.

IBM and *National Geographic* are taking the MSA problem to the next level: their combined efforts led to the launch of a far-reaching worldwide project named The Geographic Project [57]. With the advent of IBM's new supercomputer Blue Gene, which achieves a monstrous theoretical peak of 360 Teraflops, IBM provides the power to analyze the largest collection of DNA samples ever assembled. The project aims at

discovering exactly how Earth was populated.

Finally, in a totally different domain, bioinformatics even intruded the minds of science fiction movie producers like Andrew Niccol with his 1997 motion picture *Gattaca* (observe how all the letters are taken from the DNA alphabet). The film depicts a far-fetched technological world in the future where what you have in your blood ultimately determines your destiny.

2.4.2 Briteny Spears

This section broadens the spectrum of computer applications that extensively use alignment techniques to compare data. These applications radiate from a common feature found in computer operating systems; that is, a search functionality. Many computer applications such as text processing also have built-in search functionalities. A search process ultimately incorporates some form of comparison mechanism. Searching retains only compared elements that match, while discarding the rest. Furthermore, when comparing data, a similarity relation between discarded elements may also be established. Command tools like **comp** (Windows) or **diff** (Linux) [56] offer to compare files rather than to output meaningful information.

A famous tool used worldwide by millions of users every day is the spelling correction system on the Google search engine. Under the umbrella of a small spelling mistake such as ‘Briteny Spears’ resides a plethora of related applications designed to compare. Typically, all queries are compared against a dictionary, after which an orthographic confirmation is returned. Queries are also checked against similar previous inputs that have returned a higher number of hits. The search engine will then suggest a better spelling if appropriate. For interest’s sake, Google has 593 registered ways of spelling ‘Britney Spears’ [47] because users have made spelling mistakes in the input query. Each variant was written by at least two unique users within a period of three months.

Lastly, miscellaneous sequence alignment theories have also been applied to the study of linguistics [21, 43] including the evolution of languages, automatic translation or even speech recognition. A plagiarism tool is another example of a direct alignment application making extensive use of comparisons [70]. The main feature of such tools is the ability to find similarities from and within texts. The following section expresses several

inherent difficulties beyond finding similarities and making comparisons when solving MSA problems.

2.5 Where Does the MSA Problem Lie?

MSAs are hard to solve, and there is never one perfect solution, making the MSA problem complicated as well as intricate. In this section, the MSA problem will be dissected in order to understand which potential obstacles stand in the way of its resolution. For clarity's sake, the generic MSA problem is expressed with the following declaration: “*Insert gaps within a given set of **sequences** in order to **maximize** a similarity **criterion***”. From the previous statement, the problem can be broken down into three distinct inherent difficulties (listed in bold). The intricacy of MSA comes from the fact that all three difficulties addressed in the following paragraphs must be overcome simultaneously.

2.5.1 Choice of Sequence

It is evident that an amino acid sequence cannot be aligned with a nucleotide sequence. Less evident is that having sequences of the same kind is still not sufficient. In order to produce a biologically meaningful alignment, aligned sequences *must* also share homologous roots.

The problem with totally unknown sequences is that sequences need to be aligned to discover how related they are but, at the same time, sequences are required to have some relatedness prior to obtaining a meaningful alignment. Therefore, a way of ‘learning’ about sequences is to align them in an empirical manner. Such learning begins with zero knowledge, then progressively adjusts criteria according to obtained results. One-shot perfect alignment simply does not exist.

2.5.2 Is This Alignment Any Good?

The alignment process is driven by the maximization of a similarity criterion that evaluates the quality of a sequence alignment. This similarity criterion, in the context of sequence alignment by *optimization*, is referred to as an *objective function* (OF). Hence, the ‘best possible’ alignment is reached when the maximization of the OF is terminated.

Virtually any scoring function can be custom-tailored to suit the needs of a particular sequence alignment. One of the most widely used functions is the simple SoP function as described in Section 2.3.2. The ultimate goal for bioinformaticists is to find an objective function which, when optimized, always results in the best possible sequence alignment. Finding such an objective function can be a daunting task for a number of reasons, two of which are listed below.

The first difficulty resides in the lack of biological information incorporated into the objective function. Such additions would increase the mathematical complexity of the OF, therefore increasing the time complexity of each comparison. Although the OF should ideally take into account as many biological properties as possible, such information is often not available.

The second problem takes the form of gap penalties: even where the affine gap penalty model is usually used (as seen in Section 2.3.3), it is still hard to reproduce accurately what actually happened from an evolutionary perspective. Actually, finding realistic values for both the gap opening and extension penalty is uncertain. The ratio between the opening and extension gap penalties may also be adjusted proportionally (or not) to the length of the sequences concerned. The best penalty values remain largely dependent on a specific alignment and therefore need to be set empirically. Each single set of sequences to be aligned would require custom values obtained through methods of trial and error. Generic optimal penalty values for any set of sequences do not exist.

2.5.3 Computer Complexity

Doing a sequence alignment by hand is possible, although only for very short pairwise alignments (as illustrated in Figure 2.3). This is due to the fact that the number of comparisons to be executed on such small alignments is kept *reasonable*. For two sequences of length n , let β represent the total number of all different alignment possibilities, which is calculated as

$$\beta = \sum_{k=0}^n \binom{n+k}{k} \binom{n}{k} \quad (2.8)$$

A brute force pairwise alignment ultimately needs to evaluate all possibilities of gap insertions in two sequences. As an example, a pair of sequences each containing 500 residues makes the count of all possible alignments rocket to approximately 10^{400} .

PSAs are actually quite computationally simple compared to MSAs, as MSAs have a minimum of three sequences to be aligned. Having more than two sequences causes the algorithmic complexity to grow exponentially because of the considerable increase in comparisons. MSAs require a k -wise alignment for k sequences. Therefore searching for the best MSA using an exhaustive enumeration of all possible alignments becomes strictly proscribed and *not* reasonable. A microcomputer would quickly give up any brute force processing of a medium MSA because of an explosion in the memory size required. Even with the right amount of physical and virtual memory, the computation would take centuries to complete. Clearly, this problem is of a combinatorial nature (NP-complete, described in detail in the next chapter) and needs to be optimized accordingly.

The focus lies in circumventing the MSA's high computational complexity. Consequently, sophisticated and efficient strategies have been proposed in order to deal with the MSA problem in a practical way. A powerful strategy to narrow the search space is to make use of *heuristics* [27, 72, 108, 110, 113]. The idea is to take the initial sequences and refine them progressively or iteratively to maximize the 'goodness' of an alignment. Using heuristics instead of brute force methods has the negative effect of finding only approximate, sub-optimal solutions. The loss of optimality is created by the proportional trade-off made between complexity and time. As a result, for one specific set of sequences there can be many solutions that are equivalently good and acceptable.

There exists a myriad of different techniques to solve the MSA problem efficiently. None of them is superior in all cases and none of them can claim to be the best way of aligning more than two sequences. The MSA problem remains largely open and could benefit from various improvements. Section 2.6 below reviews practical methods and current techniques used to solve the MSA problem.

2.6 State of the Art

Since no single MSA solving method performs best in all cases, niche applications [81] emerged in bioinformatics practice because certain algorithms would be better suited for specific alignment conditions. Twenty-five years of continuous attempts to solve MSAs more accurately and efficiently have led to the development of numerous techniques [81]. Bioinformatics practitioners have therefore always had a toolbox containing consciously

selected tools, with each having one specific purpose.

An exhaustive enumeration of all available MSA techniques lies beyond the scope of this thesis. The goal here is to give a general taxonomy of existing MSA techniques, with special emphasis on techniques which are the most relevant within the frame of this work. For an extensive review, refer to the excellent survey by Notredame [81].

Most practical and commonly used methods for MSAs are logical extensions of PSAs. The rationale is that multiple alignments are achieved by the successive application of pairwise methods. Methods to solve MSAs are divided into four distinct categories, each of which is described below.

2.6.1 The *Exact* Approach

Applied to MSAs, *dynamic programming* (DP) [80] was initially used for the alignment of two sequences. DP first converts the sequence alignment problem into the problem of finding the shortest path in a weighted direct acyclic graph. DP's objective is to build up the best possible alignment using previous solutions of optimal alignment from smaller sub-sequences. Three steps occur in DP, namely

1. creation of a two-dimensional alignment path matrix (default for pairwise alignment);
2. stepwise calculation of score values; and
3. backtracking path (i.e. evaluation of the optimal path).

Due to the exhaustive nature of the method, DP always returns a mathematically optimum solution given its objective function. With an $O(n^k)$ computational complexity for k sequences of mean length n , DP is well suited for solving PSAs [19]. The DP method has also been extended directly to MSAs, involving comparisons of more than two sequences. Regrettably, DP fails quickly as the length and number of sequences increase (hence, DP is not feasible when $k \geq 4$). Lipman, Altschul and Kececioglu [72] pioneered an algorithm called *MSA* which extends the ability of DP to align up to 10 medium-sized sequences (of a high degree of similarity) simultaneously. *MSA* discards 'fruitless' areas within the k -dimensional hypercube volume in order to shrink the search space. In other words, *MSA* applies tight lower and upper search bounds, thereby considerably

decreasing the computational complexity. Within this exact approach, Stoye [104] and Stoye, Perry and Dress [105] also proposed a *divide-and-conquer algorithm* (DCA) which significantly extends the potential of the original MSA. Stoye's DCA has been built around the following steps:

- Firstly, split up the problem of aligning long sequences into two or more problems of aligning shorter sequences. Original sequences are cut at suitable *cut positions*.
- Secondly, align each chunk of smaller sequences optimally using MSA.
- Finally, concatenate all resulting chunk alignments into one alignment, recreating the final alignment of the original sequences.

2.6.2 The *Progressive* Approach

Progressive alignment algorithms as described by Taylor [109], Feng and Doolittle [41] perform quickly (linear complexity) and use only a small amount of memory. Typically, a progressive alignment is constructed by starting with the most similar sequences and then incrementally aligning more distant sequences or groups of sequences to the initial alignment.

The standard representative of progressive methods is *CLUSTAL W* [113]. *CLUSTAL W* can create multiple alignments, manipulate existing alignments, do profile analysis and create phylogenetic trees. *CLUSTAL W* is an updated version of *CLUSTAL* [51] where *W* stands for 'weighting', which represents the program's ability to apply weights to both sequences and program parameters. The deterministic heuristics optimized by *CLUSTAL W* is based on the SoP described in Section 2.3.2 using the affine gap penalty model. The order in which sequences are successively added is guided by a constructed phylogenetic tree (magnitude of similarity) and pairwise DP. Advantages from *CLUSTAL W* include the ability to automatically select the best suited substitution matrix and gap penalties. As a replacement of the command line program, *CLUSTAL X* [112] provides a graphical user interface for *CLUSTAL W*.

The main problem with progressive algorithms is twofold:

- Dependence upon initial pairwise alignments:

1. The very first sequences to be aligned are the most closely related on the sequence tree. If alignment is good, there will be few errors in the initial alignment.
 2. The propagation of errors from the root alignment throughout the entire alignment.
 3. The more distantly related these sequences will be, the more errors there will be.
- The choice of a suitable alignment scoring scheme (substitution matrices and gap penalties) that applies to all sequences simultaneously.

Other progressive algorithms include *MUSCLE* [37], *MATCH-BOX* [31], *MultAlign* [27] and *PileUp* [32].

2.6.3 The *Iterative* Approach

With iterative methods, the result does not depend on the initial pairwise alignment. Instead, an iterative algorithm starts with a generated initial alignment, then repeatedly refines it until no more improvement can be obtained. The main objective of the iterative approach is to *globally* enhance the quality of a sequence alignment.

A particularity and advantage of iterative algorithms is the clear decoupling between the optimization *per se* and the objective function itself. The successive alignment refinements are governed by a so-called *alignment improver*. The nature of such an improver further divides the iterative methods into *deterministic* and *stochastic* categories. On the one hand, modifications ruled by DP qualify as deterministic. On the other hand, stochastic iterative methods rely on a heuristic improver. However paradoxical it may seem, many techniques making use of the stochastic principle have left their mark as being robust MSA solvers [22]. Those techniques include *hidden Markov model* (HMM) training [36, 73, 93], *simulated annealing* (SA) [67], *Gibbs sampling* (local alignment only) [71], *ant colony optimization* (ACO) [23, 79], and *particle swarm optimization* (PSO) [55].

To date, it appears that PSO has been applied to MSA in one publication by Hsiao and Chuang [55]. Unfortunately, little detail is provided on how this was done.

A number of *evolutionary algorithm* (EA) approaches have also been developed [54] for solving MSAs. One of the first approaches used simple *genetic algorithms* (GA) [44], where a bit-matrix representation was applied. Zhang and Wong [121] also successfully combined GA techniques with pairwise DP. Other GAs include the well-known *sequence alignment by genetic algorithm* (SAGA) [82] boasting 22 operators. As GAs have a substantial role within this work, a treatment of GAs is provided in Section 3.3 of the next chapter.

Unfortunately, iterative algorithms still lack speed and reliability. Firstly, convergence to a global optimum may take a relatively long time because of the iterative nature of the method (improved in [94]). Moreover, computational time becomes drastically worse by the use of stochastic methods. To alleviate the need for speed, GAs have also been implemented on parallel computers [7]. Secondly, iterative methods are mostly considered unreliable because of the fact that one specific set of sequences would almost always yield a different final alignment due to the randomness factor. Furthermore, GAs and HMMs have shown weaknesses when dealing with *ab initio*³ alignments, but, rather, excel as optimizers of existing MSA solutions. Gotoh [48] made significant efforts to improve accuracy in multiple protein sequence alignments using iterative refinement.

2.6.4 The *Consistency-based* Approach

Consistency-based approaches consider an MSA as superior when a maximal consensus of optimal pairwise alignments has been reached. First introduced in 1993, Kececioglu formulated the MSA problem as a *maximum weight trace problem* (MWTP) [60]. In 2000, Notredame, Higgins and Heringa proposed a novel method called *T-COFFEE* (tree-based consistency objective function for alignment evaluation) [83, 84] which constitutes the most recent objective function designed with consistency in mind. When T-COFFEE is optimized, mainly by SAGA, MSAs are solved by emulating the MWTP. Essentially, T-COFFEE exposes the ability to combine results from a collection of alignments from heterogeneous sources into one final alignment. Starting by computing a library of pairwise alignments from all input sequences, the program then finds the two best global and local alignments. Secondly, T-COFFEE progressively assembles a final alignment

³Latin term meaning ‘from the beginning’.

that has the highest level of consistency within the library. This consistency method has been shown to outperform most current MSA packages with regard to accuracy. The price for accuracy is a high time complexity in the order of $O(k^3n^2)$. *DIALIGN* [2, 75, 76, 78, 106] employs a segment-to-segment comparison scheme (in opposition to character-wise) and no gap penalties. *DIALIGN* works best for finding local relations between sequences.

2.7 Summary

This chapter examined various aspects relating to sequence alignments. It focused on the multiple sequence alignment (MSA) problem on account of its high complexity. Section 2.2 began at the root of sequences by suggesting a succinct introduction to molecular biology. DNA, RNA and proteins were covered from a scientific point of view, followed by a short history of life. Section 2.3 laid out the foundation of both the manipulation and creation of alignment by defining its conceptual building blocks. A part of this chapter was then devoted to outlining the usage significance of the sequence aligning discipline. The wide array of uses in Section 2.4 ranged from the fundamental bioinformatics tool to science fiction movies. Next, Section 2.5 analyzed the multi-faceted MSA problem. Furthermore, explanations were provided on why there is no single way that leads to the desired solution. Section 2.6 looked at typical techniques employed to solve the MSA problem. Many common approaches were described, with attention being given to the most relevant ones.

Chapter 3

Computational Intelligence Paradigms

“Pure mathematics is, in its way, the poetry of logical ideas.”

- Albert Einstein

The previous chapter provided relevant background on MSA. Obtaining reasonably optimal alignments requires the optimization of an objective function. This chapter aims at providing background on optimization techniques within the computational intelligence (CI) field. Initially, a brief formalism on classical optimization is presented. The remainder of the chapter then expands on two CI paradigms. Further attention is paid to key algorithms which are relevant to experiments conducted in this work, namely genetic algorithms and particle swarm optimizers. Lastly, both paradigms are contrasted in a final discussion.

3.1 Introduction

A toothbrush and an ultra sophisticated nuclear reactor only have one thing in common: *optimization*. Consider the following fictitious example. The brand new ‘XYZ’ ambidextrous toothbrush fits nicely in the hand and allows for effective tooth cleaning and whitening, as well as offering the ultimate in brushing comfort. A cutting-edge nuclear reactor is designed by teams of chemical and structural engineers, where absolutely

no room for the slightest mistake is permitted. Ultimately, a finalized, secured reactor must also imperatively comply with specific requirements such as radioactive γ -rays exposure, massive heat dissipation, cost effectiveness, throughput efficiency and environmental friendliness, amongst others. Innovation, design and growing development are the spearheads of contemporary industry. As the toothbrush and nuclear reactor comparison shows, this constant need for performance can easily be found in all aspects in life; optimization is omnipresent.

The focal point of this chapter concerns optimization techniques and, more precisely, viable techniques that belong to the realm of computational intelligence (CI). CI refers to the study of intelligent behaviour from self-adjusting agents in complex systems within changing environments [38]. Furthermore, CI paradigms can be thought of as a collection of algorithms modelled and inspired by behaviours that originate from the elegance of Mother Nature. Such systems include the human brain with its network made up of billions of neurons, the highly organized society of ants or even a flock of lapwings heading together towards their dormitory at dusk.

The chapter is articulated in six parts. Section 3.2 defines optimization problems, along with basic optimization notions. Section 3.3 begins the exploration of CI with Darwinian-based algorithms such as the genetic algorithm (GA). Swarm intelligence (SI) is thereafter introduced in Section 3.4 with an in-depth study of particle swarm optimization (PSO). The subsequent discussion in Section 3.5 puts both PSO and GAs into perspective to highlight the strong and weak points observed in both techniques. The chapter is concluded with Section 3.6, where the aforementioned material is summarized.

3.2 Classical Optimization

This section defines the basics of optimization and the intrinsic nature of nondeterministic polynomial complete (NP-complete) problems.

3.2.1 Minimization Definition

The process of optimization is defined as finding the best values for variables that will lead to either the maximization or minimization of an objective function, which reflects a given

problem. Note that minimizing the mathematical function $f(\mathbf{x})$ equally corresponds to maximizing $-f(\mathbf{x})$. In this work, an objective function or *evaluation function* (EF), namely the *MSAfunc*, which quantifies the quality of solutions for the MSA problem is optimized over an m -dimensional search space Ω . Firstly, let \mathbf{x} represent a vector of values from Ω . That is,

$$\mathbf{x} = [x_1, x_2, \dots, x_m]^T, \quad \forall \mathbf{x} \in \Omega$$

Additionally, Ω may either symbolize a real-valued space, \mathbb{R}^m , or a discrete space where values are taken from a predefined set of values. An unconstrained global minimization of *MSAfunc*(\mathbf{x}) is formally defined as

$$\text{Find } \mathbf{x}^* \text{ for which } MSAfunc(\mathbf{x}^*) \leq MSAfunc(\mathbf{x}), \quad \forall \mathbf{x} \in \Omega \quad (3.1)$$

In order to accommodate problems exposing more than one objective, an objective function f can be extended to represent a vector of p objectives so that

$$\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_p(\mathbf{x})]^T$$

3.2.2 NP-Complete Problems

The ‘NP’ tag attached to a problem originates from its *nondeterministic polynomial* execution property. The term ‘polynomial’ means that if at least one solution exists, it can be estimated and validated within an exponential time period which is usually worsened proportionally to the input length of a problem. The adjective ‘nondeterministic’ suggests that no definite solving rule exists, implying some kind of guessing and estimation.

If a problem is NP and all other NP problems are polynomial time reducible to the problem, then the problem becomes NP-complete. Moreover, these problems are said to be *intractable*, which refers to the solving incapability of deterministic algorithms. Representatives of NP-complete problems include the famous knapsack problem, the Hamiltonian cycle problem and the travelling salesman problem (TSP).

The MSA problem, as detailed in Chapter 2, also falls into the category of NP-complete problems [15, 59, 118]. Due to their nondeterministic nature, several CI algorithms have been successful at finding near optimal solutions to NP-complete problems

in a reasonable amount of time [101, 121]. Two CI paradigms that have been applied to MSA are reviewed in the Sections 3.3 and 3.4.

3.3 Genetic Algorithms

This section discusses a paradigm of CI, evolutionary computation (EC), which draws its inspiration from Darwinian theories of evolution. Evolutionary algorithms all share the same principle of ‘survival of the fittest’ (as discussed in Section 2.2.2). A vast range of evolutionary models under the EC paradigm exists [10], but the genetic algorithm [42], in particular, fits within the scope of this thesis. The first application of the EC paradigm was the GA model, which was popularized by Holland [53] in 1975. This presentation of GAs only serves as an introduction, leaving the reader to consult more literature on the topic [38, 42, 44]. A general overview of the GA is given in Section 3.3.1. An explanation of the internal GA operators follows in Section 3.3.2. Finally, Section 3.3.3 overviews a modification to the original GA called the cooperative coevolutionary genetic algorithm (CCGA).

3.3.1 Genetic Algorithm Essentials

A genetic algorithm evolves an initial *population* of random individuals, each representing a potential solution, over an arbitrary amount of time. A fitness function that reflects the optimization problem takes the role of evaluating each individual’s quality. In GA practice, a fitness function is computed for the purpose of ranking solutions within a population. At regular time intervals, called *generations*, the algorithm carries out a selection of individuals which in turn survive to the next generation.

After each generation, a portion of individuals are submitted, firstly, to recombination in an effort to produce more promising solutions, and, secondly, to mutation (discussed in Section 3.3.2) to enhance diversity. The reproductive cycle then restarts until some optimization criteria are met.

The algorithm is dubbed ‘genetic’ because it models genetic evolution through many generation cycles. Within the evolutionary scheme, sexual reproduction allows parents to pass on their chromosomal information to new *offspring*. In general, every offspring pro-

duced in this manner joins the new generation's population. Each individual comprises one *chromosome*, made up of several *genes* used to form its unique genotype inherited from the biological parents. A gene in a GA represents a parameter of the problem.

In conventional GAs, it is common to represent the genotype of an individual as a fixed-length bit string. The string length must be determined prior to optimization and reflects the dimensionality of the problem. Parameters for a specific problem can thus be easily mapped into a bit string by turning the corresponding bits 'on or off'. Other representations include variable length bit strings [45] and vectors of real-valued numbers [58].

A genetic algorithm therefore optimizes a fitness function by evolving candidate solutions over a maximum number of generations, g_{max} . The pseudocode in Algorithm 3.1 explains the steps adopted for a general GA.

```

 $g \leftarrow 0$ 
Initialize all individuals of the initial population  $C_g$ 
while not converged, or  $g \leq g_{max}$ :
    Perform fitness evaluation on each individual in  $C_g$ 
     $g \leftarrow g + 1$ 
    Select parents from  $C_{g-1}$ 
    Create offspring  $O_g$  by applying cross-over operators to parents
    Apply mutation to  $O_g$ 
    Form the new generation population  $C_g$  by mixing a selection of
    individuals from  $C_{g-1}$  and  $O_g$ 

```

Algorithm 3.1: Pseudocode outline of the genetic algorithm

A population is declared to have converged when individuals have reached an optimum. Alternatively, convergence in a population may also be considered when there exists at least one individual whose fitness $f(\mathbf{x})$ satisfies a minimum threshold value f_ϕ or when there is no change in average fitness $f(\mathbf{x})$ over a number of generations.

3.3.2 Evolutionary Operators

Operators are applied on individuals to mimic evolution processes found in nature. These GA operators, namely *cross-over*, *mutation* and *selection*, are briefly examined below.

Cross-over

The cross-over operator models the creation of offspring, which simulates natural reproduction through a recombination of the genetic material from two parents. The process involves taking genes from both parents to produce offspring.

Assuming a bit-level data representation, a binary bit mask dictates the selection of bits from both parents in order to recreate two new bit strings. The extraction of bit string pieces from the parents may be conducted by using any of the three different masks as illustrated in Figure 3.1 and explained below. Cross-over occurs at a particular rate, called the *cross-over probability*.

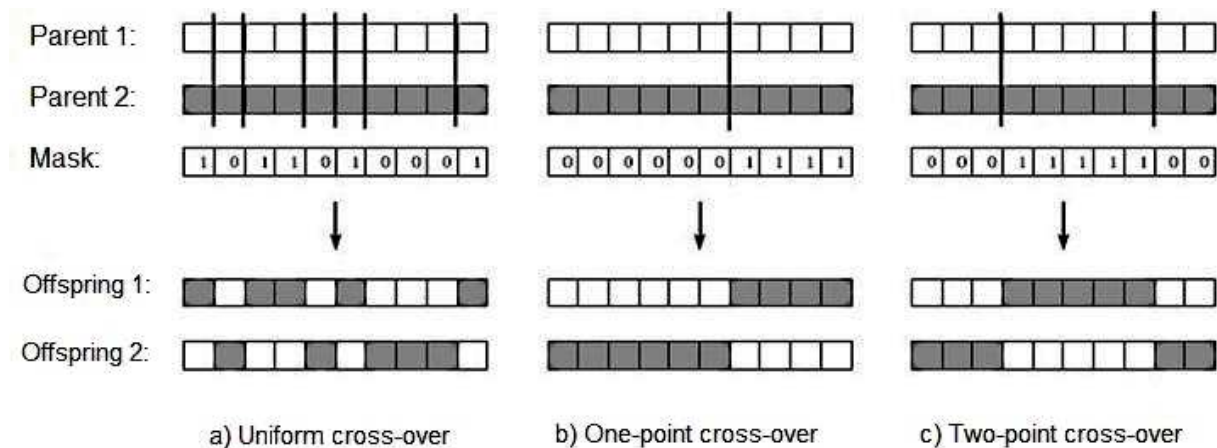


Figure 3.1: Three cross-over operators for bit representations

Uniform cross-over: A mask is randomly generated by assigning a bit value to a mask element with equal probability. Uniform cross-over produces offspring through a random composition of the two parents. See Figure 3.1(a).

One-point cross-over: An offset is chosen at random, marking a pivot point. Bits from both parents are taken in their original order until the offset is reached, after which all positions are swapped. See Figure 3.1(b).

Two-point cross-over: The same principle as for one-point cross-over is followed, where two different offsets are randomly selected. All bits from parent one are taken from the positions enclosed by the two offset delimiters, the rest is obtained from the other parent. See Figure 3.1(c).

Moreover, one-point and two-point cross-overs can also be applied to chromosomes with a real-valued representation. Considering real-valued genes only, arithmetic cross-over can be used. Two offspring, $O_{g,1}$ and $O_{g,2}$, are generated using

$$O_{g,1} = r_1 C_{g-1,1} + (1.0 - r_1) C_{g-1,2}$$

$$O_{g,2} = (1.0 - r_1) C_{g-1,1} + r_1 C_{g-1,2}$$

where $r_1 \sim U(0, 1)$ and the two parents are $C_{g-1,1}$ and $C_{g-1,2}$.

Mutation

The objective of mutation is to introduce new genetic material into existing individuals to reduce the chances that individuals get trapped into local optima. Therefore, mutation schemes facilitate diversification (i.e. adds ‘natural noise’) in a population. For a binary representation, mutation is applied by randomly alternating bit values. For a real-valued representation, mutation adds a random number to the original values, usually sampled from a zero-mean Gaussian distribution. Mutation occurs at a certain probability referred to as the *mutation rate*. Normally, a large mutation rate is initially used to encourage global exploration of the search space. The mutation rate is then decreased progressively as solutions are starting to converge to the optimum.

Selection

Selection is used to select parents for cross-over or parents to survive to the next generation. Ideally, fit individuals must be emphasized in order to promote good genes through generations. This is not the case when *random selection* is applied, because

the current fitness level is simply ignored. Random selection may incur negative repercussions; for example, good potential solutions may be discarded while weak ones are promoted. However, random selection reduces the effect of selection pressure.

A selection scheme that reflects the ‘survival of the fittest’ idea is called *elitist selection*, which is applied to select parents to survive to the next generation. In elitist selection, all individuals within a population are first ranked according to their respective fitnesses. Then highly fit individuals are marked to form part of the next generation.

Another popular scheme is *tournament selection*. For each parent to be selected, a tournament of a limited number of individuals is randomly selected. The best individual in the tournament is the parent. One advantage of tournament selection is that weak individuals are discarded while good individuals are retained to construct the next generation.

3.3.3 Cooperative Coevolutionary Genetic Algorithm

In 1997, Potter developed the cooperative coevolutionary genetic algorithm (CCGA) [92]. The main objective of CCGA was to address scalability issues observed in traditional GAs. In the CCGA, an individual’s genes are distributed over a set of K independent sub-populations which are each responsible for evolving a limited set of genes. As a consequence, no single sub-population has the necessary information to solve the problem itself, which implies that the fitness evaluation requires a special mechanism to handle a split chromosome in sub-populations.

The construction of a solution is done by adding together the best individuals from each sub-population. The difficulty is how to select the best individual from a sub-population since the individuals in question do not represent complete solutions. The issue is resolved by maintaining the complete m -dimensional solution as a *context vector*.

The basic way of constructing the context vector is by concatenating the best individuals from all sub-populations. All components within the context vector remain fixed except for the components that correspond to the current sub-population under evaluation. The value of the corresponding components in the context vector is then replaced with the selected individual from the sub-population. After the individual from the sub-population is replaced within the context vector, the fitness of the context

vector is evaluated, and that fitness is assigned as the fitness of the individual in the sub-population.

3.4 Particle Swarm Optimization

From an ornithological perspective, Mother Nature's quiet sophistication inspires when a harmonious flight of Canada geese (*Branta canadensis*) heads flawlessly as an arrow-shaped flock, which is an illustration of an interesting phenomenon: group behaviour. Likewise, schools of fish and ant colonies have also displayed astonishing social behaviour. Many disciplines, including physics and computer science, have been intrigued by the implication of such simple, yet powerful, organisms. The key to such self-organizing formations is the local interaction constantly happening between individuals. In 1995, Kennedy and Eberhart [62] derived particle swarm optimization (PSO) directly from the choreography of bird flocks. The PSO is a population-based stochastic search algorithm for global optimization. PSO has been found to be successful in a wide variety of optimization problems [38, 39, 115]. Since the introduction of PSO, there has been a considerable amount of work done in developing new PSO algorithms with improved convergence and diversity. The PSO has been in the spotlight of CI research for several years, resulting in a wealth of literature available on the topic [8, 24, 86, 87, 88, 107].

Section 3.4.1 presents the general PSO algorithm. Different PSO neighbourhood topologies are overviewed in Section 3.4.2. Section 3.4.3 discusses important PSO characteristics, emphasizing their respective influence on performance. Finally, three versions of PSO are reviewed in Section 3.4.4.

3.4.1 Algorithm Essentials

In analogy with GAs, individuals within a population in the context of PSO become *particles* within a *swarm*. Each particle in a swarm, Φ , is represented by an m -dimensional position vector, \mathbf{x}_i , that represents a potential solution to the optimization problem. Each particle is treated as one point in the m -dimensional problem space. At first, all particles are uniformly scattered throughout the search space. Particles are then gradually 'flown' through a multi-dimensional search space which marks the start of

the optimization process. The motion of a particle is governed according to its own experience as well as the experience of its neighbours.

Furthermore, each particle keeps track of the best position it came across thus far, denoted by \mathbf{y}_i . The worth of a particle at a position, \mathbf{x}_i , is measured using a predefined fitness function, $f : \mathbb{R}^m \rightarrow \mathbb{R}$, that represents the optimization problem. Particles within the swarm are organized based on a neighbourhood topology, which defines the mode of information sharing among particles. This topology allows particles to communicate with each other. The behaviour that emerges is that particles are attracted towards better particles, similar to the flocking behaviour. Three characteristics are maintained by each particle:

1. The *current position*, \mathbf{x}_i , in search space. The position at time step t of a particle is updated by adding a velocity to its current position as follows:

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1) \quad (3.2)$$

2. A *personal best position*, \mathbf{y}_i , is the best position that the particle has recorded since the first time step. When dealing with minimization, \mathbf{y}_i at next time step $t+1$ is computed as

$$\mathbf{y}_i(t+1) = \begin{cases} \mathbf{y}_i(t) & \text{if } f(\mathbf{x}_i(t+1)) \geq f(\mathbf{y}_i(t)) \\ \mathbf{x}_i(t+1) & \text{if } f(\mathbf{x}_i(t+1)) < f(\mathbf{y}_i(t)) \end{cases} \quad (3.3)$$

3. Its *current velocity*, \mathbf{v}_i . At time step t , the velocity $v_{i,j}(t)$ in dimension j is updated using

$$v_{i,j}(t+1) = v_{i,j}(t) + c_1 r_{1,j}(t)[y_{i,j}(t) - x_{i,j}(t)] + c_2 r_{2,j}(t)[\hat{y}_{i,j}(t) - x_{i,j}(t)] \quad (3.4)$$

where c_1 and c_2 are positive *acceleration coefficients* of the cognitive and social components respectively (discussed below), while $r_{1,j}(t), r_{2,j}(t) \sim U(0,1)$ are uniform random numbers in the range $[0,1]$. The symbol $\hat{\mathbf{y}}_i(t)$ refers to the neighbourhood's best particle position vector so far, as demonstrated in the next section.

Depending on the nature of the problem being optimized, different stopping criteria may be applied to declare that convergence on to an optimum solution is reached. Usually, a PSO algorithm executes for an arbitrary number of iterations or fitness function

evaluations. A PSO may also be terminated when the average velocity update over all particles approximates zero, indicating that particles are no longer moving. Alternatively, the search can be terminated when an acceptable solution is found (e.g. the error bound becomes small enough).

```

Create a swarm  $\Phi$  of size  $s$ 
Initialize the position vectors of all particles,  $\mathbf{x}_i(0) \sim U(\mathbf{x}_{min}, \mathbf{x}_{max})$ 
 $\mathbf{y}_i(0) \leftarrow \mathbf{x}_i(0), \quad \forall i \in \{1, \dots, s\}$ 
 $\mathbf{v}_i(0) \leftarrow 0, \quad \forall i \in \{1, \dots, s\}$ 
 $t \leftarrow 0$ 
repeat:
    for all particles  $i \in \{1, \dots, s\}$  do:
        if  $f(\mathbf{x}_i(t)) < f(\mathbf{y}_i(t))$  then
             $\mathbf{y}_i(t) \leftarrow \mathbf{x}_i(t)$ 
        end if
        if  $f(\mathbf{y}_i(t)) < f(\hat{\mathbf{y}}_i(t))$  then
             $\hat{\mathbf{y}}_i(t) \leftarrow \mathbf{y}_i(t)$ 
        end if
        Update velocity  $\mathbf{v}_i$  according to Equation (3.4)
        Update position  $\mathbf{x}_i$  according to Equation (3.2)
    end for
     $t \leftarrow t + 1$ 
until stopping condition is true

```

Algorithm 3.2: The standard PSO algorithm

The standard PSO algorithm is summarized in Algorithm 3.2.

3.4.2 Neighbourhood Topologies

The social interaction established in the PSO algorithm is driven by the structure through which particles inter-communicate. Only particles that belong to the same neighbourhood can exchange information among each other. Therefore, a swarm may contain many

networks of particles of different shapes and connections, called *topologies*. Neighbourhood topologies are also known as *information sharing structures*. Two early versions of the original PSO are the Global Best (*GBest*) and Local Best (*LBest*) [34]. The *GBest* PSO and the *LBest* PSO respectively reflect the *star* and the *ring* topologies. Additionally, a more recent topology called *Von Neumann* has been proposed [65]. Figure 3.2 depicts all three topologies. The rest of the section reviews these information sharing structures.

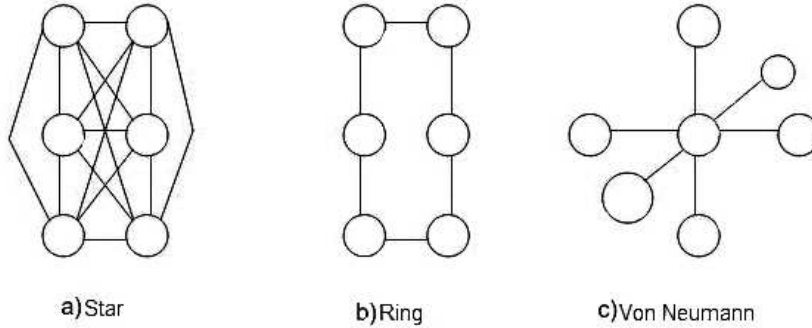


Figure 3.2: Common PSO topologies

Star Topology

The star neighbourhood structure offers an entirely interconnected network where all particles within the swarm are able to share information with one another. In this case, the swarm is comprised of only one neighbourhood englobing every particle, as pictured in Figure 3.2(a). Advantages of the *GBest* PSO, using the star topology, reside in its ease of implementation and its rapid convergence. However, a major drawback emanating from this structure is poor performance in multimodal search spaces. In effect, the *GBest* PSO may prematurely converge on a false optimum when dealing with convoluted search spaces.

With reference to velocity update equation (3.4), the neighbourhood best, $\hat{\mathbf{y}}_i$, is selected as the best solution found by the entire swarm, i.e.

$$\hat{\mathbf{y}}(t) \in \{\mathbf{y}_0(t), \mathbf{y}_1(t), \dots, \mathbf{y}_s(t)\} \mid f(\hat{\mathbf{y}}(t)) = \min\{f(\mathbf{y}_0(t)), f(\mathbf{y}_1(t)), \dots, f(\mathbf{y}_s(t))\} \quad (3.5)$$

Ring Topology

The ring neighbourhood structure connects several particles in a ring structure based on particle indices, as shown in Figure 3.2(b). Particles have several channels of communication, which depend on the size of the neighbourhood. As a result, a swarm ends up being organized into many overlapping neighbourhoods, \aleph_i , with each containing one neighbourhood best particle position, $\hat{\mathbf{y}}_i$, defined as

$$\hat{\mathbf{y}}_i(t+1) \in \aleph_i \mid f(\hat{\mathbf{y}}_i(t+1)) = \min\{f(\mathbf{y}_i)\}, \quad \forall \mathbf{y}_i \in \aleph_i \quad (3.6)$$

where a neighbourhood \aleph_i is defined as

$$\aleph_i = \{\mathbf{y}_{i-s_{\aleph_i}}(t), \mathbf{y}_{i-s_{\aleph_i}+1}(t), \dots, \mathbf{y}_{i-1}(t), \mathbf{y}_i(t), \mathbf{y}_{i+1}(t), \dots, \mathbf{y}_{i+s_{\aleph_i}-1}(t), \mathbf{y}_{i+s_{\aleph_i}}(t)\} \quad (3.7)$$

with neighbourhoods of size s_{\aleph_i} . Note that the GBest PSO is a special case of the LBest PSO where $s_{\aleph_i} = s$.

It has been shown [34, 100] that a PSO based on the LBest model converges more slowly than its GBest counterpart because of its larger search space exploration, but at the advantage of ultimately finding better optimal solutions.

Von Neumann (VN) Topology

The Von Neumann structure organizes the particles into a three-dimensional lattice, like a grid, as illustrated in Figure 3.2(c). Introduced by Kennedy and Mendes [65], this recent information sharing structure has shown significant improvements over the LBest and GBest models, compared to the original PSO.

3.4.3 PSO Parameters

This section discusses parameters which have an impact on the behaviour of PSO. By adding or modifying certain parameters that have an influence on diversity, a PSO may demonstrate various degrees of success with reference to a specific problem. Empirical research aims at adjusting PSO parameters to achieve better solutions while improving convergence speed. A discussion of the influence of these parameters is conducted in the next six subsections.

Acceleration Coefficients

As mentioned earlier, a particle does not rely solely on its previous velocity to calculate its next step. The velocity update equation, reprinted below in equation (3.8), incorporates:

- a *cognitive* component, which expresses the confidence in the particle's own experience (includes the particle's best position found so far); and
- a *social* component, which expresses the confidence in the neighbourhood's experience (includes the best position obtained thus far within the particle's neighbourhood).

$$v_{i,j}(t+1) = v_{i,j}(t) + \underbrace{\mathbf{c}_1 r_{1,j}(t)[y_{i,j}(t) - x_{i,j}(t)]}_{\text{Cognitive}} + \underbrace{\mathbf{c}_2 r_{2,j}(t)[\hat{y}_{i,j}(t) - x_{i,j}(t)]}_{\text{Social}} \quad (3.8)$$

Both cognitive and social components are independently weighted by acceleration coefficients c_1 and c_2 respectively (displayed in bold in the equation). With $c_1 > c_2$, each particle is biased towards its own best position, therefore exploring more. However, $c_2 > c_1$ facilitates exploitation by making every particle more 'attracted' to the best position of the neighbourhood. Assuming no inertia, smooth particle trajectories may be obtained by setting a low value for both c_1 and c_2 . Conversely, higher values propel particles in irregular and oscillatory trajectories. Usually, c_1 and c_2 are static values set to be roughly equal to ensure an equilibrium of knowledge for each particle. However, their optimal values may vary and are therefore problem dependent. Kennedy [61] suggested that:

$$c_1 + c_2 \leq 4.0 \quad (3.9)$$

Inertia Weight

The application of an additional weight variable, ω , to the previous velocity in velocity update equation (3.4) has been shown to notably improve performance in terms of convergence speed. Known as the *inertia weight*, this factor, introduced by Shi and Eberhart [98], serves as an extension to the original PSO (except when ω is set to 1). The velocity update changes to

$$v_{i,j}(t+1) = \omega v_{i,j}(t) + c_1 r_{1,j}(t)[y_{i,j}(t) - x_{i,j}(t)] + c_2 r_{2,j}(t)[\hat{y}_{i,j}(t) - x_{i,j}(t)] \quad (3.10)$$

The inertia weight acts as a scaling factor over the previous velocity. As a result, the calculation of the new velocity will include some proportionality of its anterior velocity. The impact of this mechanism on the trajectory of a particle is either a deceleration or an acceleration. It is important to note that if a particle accelerates due to larger step sizes (when $\omega > 1.0$), then its trajectories would immediately cause larger coverage of the search space, which is preferred for exploration. The deceleration of a particle (with $\omega < 1.0$) creates an effect of a fine-grained search that may be desired for the exploitation of a particular area in the search space. The key to setting the inertia weight adequately is to find an efficient trade-off between exploration and exploitation to ensure optimal convergence of the particles. Work on PSO convergence behaviour by Van den Bergh [115] has led to the theoretical derivation of a constraint that represents the relationship between parameters. This constraint must be satisfied to ensure convergence, stated formally as:

$$1.0 > \omega > \frac{(c_1 + c_2)}{2} - 1 \quad (3.11)$$

Instead of applying a static inertia throughout the entire course of a PSO algorithm, inertia values may be changed dynamically over time. By doing so, the early stages of the algorithm are favoured towards exploration, while exploitation of local areas occurs at later stages. An inertia weight that is linearly decreased over time has been introduced and studied by Shi and Eberhart [99].

Velocity Clamping

Monitoring the velocity of a particle is recommended in order to reduce potential divergent behaviour. In effect, when the velocity explodes to large values, a particle is more likely to leave the search space or never settle in a specific area because of giant steps. In general, the velocity is clamped over a range with lower and upper bounds $[-V_{max}, V_{max}]$ in each dimension. The value of V_{max} is usually set to a value proportional to the domain of the variables being optimized.

Swarm Size

Indicated by s , the swarm size denotes the number of particles present in the swarm. The number of particles does have an impact on the way the search space is covered. On

the one hand, large swarms have the advantage of spreading faster, thus exploring and attaining optimal solutions in fewer iterations than smaller swarms. On the other hand, small swarms cost computationally less per iteration and have also proven to find optimal solutions successfully [115]. Again, the most adequate number of particles depends on the problem under consideration. Empirical methods need to be applied in order to find an ideal swarm size.

Number of Iterations

Iterations in the context of PSO correspond to generations in GAs. An iteration represents an atomic execution (one time step t) of the PSO algorithm. When utilized as a stopping criterion, a PSO will then terminate when an arbitrary maximum number of iterations, T , has been reached. If the iteration limit has been set too low, a risk exists that acceptable solutions may not yet have been found. Conversely, too many iterations may result in a waste of computational time, as an optimal solution will already have been discovered well before the end of the computation time. Relative to the number of iterations, the number of objective function evaluations can also be used as a stopping criterion. Each particle's fitness over all iterations represents the number of objective function evaluations.

Synchronous versus Asynchronous Updates

In a synchronous update, the personal best and global (or neighbourhood) best positions are updated after all the particle position updates have been computed and only once per iteration. Alternatively, asynchronous updates calculate the new best positions after each particle position update. This work adheres to the study by Carlisle and Dozier [20] which states that a synchronous update is more appropriate for the GBest PSO and an asynchronous update is more suitable for the Lbest PSO. Furthermore, in this work, asynchronous updates are applied to PSOs that rely on the Von Neumann topology (Von Neumann topology is not global, whereas the star topology in Gbest PSO) is.

3.4.4 Modifications to PSO

After the examination of crucial internal parameters of the particle swarm optimizer, three relevant variations of the original PSO, namely binary PSO, mutating PSO and cooperative split PSO, are now discussed. These modifications are discussed due their direct use in the work presented in this thesis.

Binary PSO

While the basic PSO has been designed to work with real-valued, continuous domains, problems such as the n -queens require optimization of discrete-valued variables. The binary PSO (B-PSO), introduced by Kennedy and Eberhart [63] in 1997, has been developed to allow optimization over binary search spaces. The B-PSO may also be applied to real-valued optimization problems after their domain has been converted to a binary-valued domain.


An outline of the B-PSO is provided in Algorithm 3.3. In the binary version of the PSO, values for the position vectors are restricted to the set $\{0, 1\}$; that is, $\mathbf{x}_i, \mathbf{y}_i, \hat{\mathbf{y}}_i \in \mathbb{B}^m$. However, no restriction exists on the velocity values. Consequently, the velocity update used for the B-PSO remains identical to the S-PSO. The velocity in the B-PSO is used to determine the probability of flipping a bit. Transformation of the S-PSO into a B-PSO lies in a probabilistic position update equation, given by

$$x_{i,j}(t+1) = \begin{cases} 0 & \text{if } r_{3,j}(t) \geq \text{sig}(v_{i,j}(t)) \\ 1 & \text{if } r_{3,j}(t) < \text{sig}(v_{i,j}(t)) \end{cases} \quad (3.12)$$

where $r_{3,j}(t) \sim U(0, 1)$ and $\text{sig}(v_{i,j}(t))$, is the sigmoid function,

$$\text{sig}(v_{i,j}(t)) = \frac{1}{1 + e^{-v_{i,j}(t)}} \quad (3.13)$$

Since velocity is interpreted as a probability, and $\mathbf{v}_i \in \mathbb{R}^m$, the sigmoid function is used to scale velocities to the range $[0, 1]$. Due to asymptotic characteristics of the sigmoid function, and to ensure that probabilities do not converge too quickly towards 0 or 1, velocities are usually clamped within the range $[-4, +4]$ [35]. Performance-wise, Kennedy and Spears [66] observed that the B-PSO was able to find solutions to problems of high dimensionality and outperformed GAs in terms of speed.



```

Create a swarm  $\Phi$  of size  $s$ 
Initialize the position vectors of all particles,  $\mathbf{x}_i(0) \sim U(\mathbf{x}_{min}, \mathbf{x}_{max})$ 
 $\mathbf{y}_i(0) \leftarrow \mathbf{x}_i(0), \quad \forall i \in \{1, \dots, s\}$ 
 $\mathbf{v}_i(0) \leftarrow 0, \quad \forall i \in \{1, \dots, s\}$ 
 $t \leftarrow 0$ 
repeat:
    for all particles  $i \in \{1, \dots, s\}$  do:
        if  $f(\mathbf{x}_i(t)) < f(\mathbf{y}_i(t))$  then
             $\mathbf{y}_i(t) \leftarrow \mathbf{x}_i(t)$ 
        end if
        if  $f(\mathbf{y}_i(t)) < f(\hat{\mathbf{y}}_i(t))$  then
             $\hat{\mathbf{y}}_i(t) \leftarrow \mathbf{y}_i(t)$ 
        end if
        Update velocity  $\mathbf{v}_i$  according to Equation (3.4)
        Update position  $\mathbf{x}_i$  according to Equation (3.12)
    end for
     $t \leftarrow t + 1$ 
until stopping condition is true

```

Algorithm 3.3: The binary PSO algorithm

Mutating PSO

Premature convergence is a tendency that seems to be recurrent for the basic PSO [50, 96, 103]. However, this tendency is largely problem-dependent as highly multimodal functions such as the complex Schwefel function falsely attract particles to local optima.

Early particle convergence and stagnation are a direct consequence of a lack of diversity [40, 120]. Mutation mechanisms, as utilized by evolutionary programming (EP), have successfully been applied to PSO [74, 96, 122] to maintain diversity throughout the optimization process. When applied to either the velocity or position vectors of particles, mutation enhances the exploration capability of the PSO. During the execution of a typical mutating PSO, a high mutation probability is used at the beginning, which is then steadily decreased as the number of iterations increases. The rationale behind the

gradual decrease in the mutation probability is to promote diversity during the initial steps of the search for greater exploration and later to improve exploitation of superior solutions.

```

Create a swarm  $\Phi$  of size  $s$ 
Initialize the position vectors of all particles,  $\mathbf{x}_i(0) \sim U(0, 1)$ 
 $\mathbf{y}_i(0) \leftarrow \mathbf{x}_i(0), \quad \forall i \in \{1, \dots, s\}$ 
 $\mathbf{v}_i(0) \leftarrow 0, \quad \forall i \in \{1, \dots, s\}$ 
 $t \leftarrow 0$ 
repeat:
    for all particles  $i \in \{1, \dots, s\}$  do:
        if  $f(\mathbf{x}_i(t)) < f(\mathbf{y}_i(t))$  then
             $\mathbf{y}_i(t) \leftarrow \mathbf{x}_i(t)$ 
        end if
        if  $f(\mathbf{y}_i(t)) < f(\hat{\mathbf{y}}_i(t))$  then
             $\hat{\mathbf{y}}_i(t) \leftarrow \mathbf{y}_i(t)$ 
        end if
        Update velocity  $\mathbf{v}_i$  according to Equation (3.4)
        Update position  $\mathbf{x}_i$  according to Equation (3.2)
        for all dimensions  $j \in \Omega$  do:
            if  $U(0, 1) < P_m(t)$  then
                Mutate position  $x_{i,j}$  according to Equation (3.14)
            end if
        end for
    end for
     $t \leftarrow t + 1$ 
    Decrease  $P_m(t)$ 
until stopping condition is true

```

Algorithm 3.4: The mutating PSO algorithm

The application of *Gaussian* mutation predominantly aims at adjusting position vectors after the completion of both velocity and position update equations. Position vectors



are then perturbed by adding a random value sampled from a Gaussian distribution of zero-mean to each component of the particle position vectors. Higashi and Iba [50] mutate all j components of particle position vectors, $\mathbf{x}_i(t+1)$, to obtain a new position, $\mathbf{x}'_i(t+1)$. Mutation occurs at a predefined probability, $P_m(t)$, which is linearly decreased over time (usually from 0.9 down to 0.1). Each component, $x_{i,j}(t+1)$, is mutated using

$$x'_{i,j}(t+1) = x_{i,j}(t+1) + N(0, \sigma)x_{i,j}(t+1) \quad (3.14)$$

where the standard deviation, σ , is defined as one tenth of the length of the search space [50], i.e.

$$\sigma = 0.1(\mathbf{x}_{max} - \mathbf{x}_{min}) \quad (3.15)$$

Unlike the use of a static σ by Higashi and Iba, σ can also dynamically decrease. Experiments conducted by Higashi and Iba [50] revealed that the PSO combined with Gaussian mutation was always able to outperform the standard PSO when minimizing typical test functions, all characterized by either unimodality or multimodality. The pseudocode for the mutating PSO is outlined in Algorithm 3.4.

Cooperative Split PSO

Based on Potter's CCGA, Van den Bergh and Engelbrecht [116, 117] introduced the cooperative split PSO algorithm (CPSO- S_K). The prime motivation for using a cooperative PSO was to improve the performance of PSO in optimizing high dimensional functions. The CPSO- S_K algorithm functions in a similar fashion to Potter's CCGA (refer to Section 3.3.3). The crucial difference resides in its participant algorithm, with the particles representing parts of the solution as opposed to genes.

The CPSO- S_K has K (referred to as the *split factor*) independent *sub-swarms*, S_k , which encompass all the split particles and are each responsible for optimizing one part of the divided particle vector. The construction of a solution is achieved by recreating the original full-sized particles from all sub-swarms. Several approaches can be used to calculate K . The approach called *perfect split* splits the problem into parts of equal dimension to be assigned to each participant. Using integer values, excluding 1, the smallest divisibility factor of the total number of dimensions that is returned is used as K .

The CPSO- S_K offers the advantage that a finer-grained search is performed. As a result, the CPSO- S_K can be perceived as ‘virtual parallel PSOs’ and as a powerful local search mechanism. However, the CPSO- S_K raises a problem in terms of swarm credit assignment. Each sub-swarm needs to be assigned credit for the quality of its participation in the overall solution. The danger is that the algorithm could spend too much time optimizing variables that have little effect on the overall solution.

```

Create and initialize  $K$  ( $m/K$ )-dimensional swarms
Initialize the position vectors of all particles,  $S_k.\mathbf{x}_i(0) \sim U(\mathbf{x}_{min}, \mathbf{x}_{max})$ 
 $S_k.\mathbf{y}_i(0) \leftarrow S_k.\mathbf{x}_i(0), \quad \forall i \in \{1, \dots, S_k.s\}, \quad \forall k \in \{1, \dots, K\}$ 
 $S_k.\mathbf{v}_i(0) \leftarrow 0, \quad \forall i \in \{1, \dots, S_k.s\}, \quad \forall k \in \{1, \dots, K\}$ 
 $t \leftarrow 0$ 
repeat:
  for each sub-swarm  $S_k, k = 1, \dots, K$  do:
    for all particles  $i \in \{1, \dots, S_k.s\}$  do:
      if  $f(\mathbf{b}(k, S_k.\mathbf{x}_i(t))) < f(\mathbf{b}(k, S_k.\mathbf{y}_i(t)))$  then
         $S_k.\mathbf{y}_i(t) \leftarrow S_k.\mathbf{x}_i(t)$ 
      end if
      if  $f(\mathbf{b}(k, S_k.\mathbf{y}_i(t))) < f(\mathbf{b}(k, S_k.\hat{\mathbf{y}}_i(t)))$  then
         $S_k.\hat{\mathbf{y}}_i(t) \leftarrow S_k.\mathbf{y}_i(t)$ 
      end if
      Update velocity  $S_k.\mathbf{v}_i$  according to Equation (3.4)
      Update position  $S_k.\mathbf{x}_i$  according to Equation (3.2)
    end for
  end for
until stopping condition is true

```

Algorithm 3.5: The cooperative split PSO algorithm

The algorithm for CPSO- S_K is outlined in Algorithm 3.5. The context vector, $\mathbf{b}(k, \mathbf{z})$, represents the necessary mechanism that provides the glue for the cooperative approach. In effect, $\mathbf{b}(k, \mathbf{z})$ returns an m -dimensional vector which is formed by the concatenation of the global best positions from all sub-swarms, but excluding the k^{th} component which

is replaced by \mathbf{z} . The position vector of a particle from the S_k^{th} sub-swarm is denoted by \mathbf{z} . The context vector is thus defined as

$$\mathbf{b}(k, \mathbf{z}) = (S_1 \cdot \hat{\mathbf{y}}, \dots, S_{k-1} \cdot \hat{\mathbf{y}}, \mathbf{z}, S_{k+1} \cdot \hat{\mathbf{y}}, \dots, S_K \cdot \hat{\mathbf{y}}) \quad (3.16)$$

3.5 Swarm Intelligence versus Evolutionary Computation

This section compares the characteristics of PSO (as introduced in Section 3.4) with those of EAs (as seen in Section 3.3).

Population versus Swarm: A population of individuals merely implies that a ‘group’ of individuals exists. In the case of PSO, the term ‘swarm’ is used to refer to a group of individuals. Individuals are referred to as particles. Particles are bound together by social interactions. The fact that particles act as an ensemble strongly differentiates PSO, since individuals in EC are independent and totally unaware of the existence of their neighbours. Fundamentally, particles in a swarm explore the search space exhibiting collective behaviour, while EC individuals do not.

Generation versus Iterations: EC and PSO both keep track of the algorithm progress. In the case of EC, individuals can die and reproduce through generations. In PSO, particles never die, they live throughout all iterations of the algorithm.

Memory: Since PSO relies on previously obtained best solutions, PSO algorithms need memory to retain and compare positions. Unless elitism is used, EC does not make use of memory because individuals explore the search space on their own without knowing what other individuals, including themselves, have found.

Randomness factor: For both paradigms, movements are based on stochastic variation in current positions. The shared stochastic nature affects the search for a solution in PSO and EC differently. In PSO, the r_1 and r_2 factors in the velocity update equation (refer to equation (3.4)) act as the random components. The stochastic element in PSO affects the direction to find a solution. In EC,

the stochastic element affects its mutation and selection schemes (refer to Section 3.3.2). Individuals in EC are randomly mutated in the hope of introducing better solutions, which is not the case in PSO.

Conclusively, while both PSO and EC are population-based, stochastic optimization algorithms, PSO does not fall under EC. The fact that PSO relies on collective, social interaction to enable the optimization process is enough to acknowledge its identity as a distinct member of SI.

3.6 Summary

This chapter dealt mainly with evolutionary computation (EC) and swarm intelligence (SI) paradigms within computational intelligence (CI). Section 3.2 covered a short introduction on optimization theory and NP-complete problems. Next, the genetic algorithm (GA), a common representative of the EC paradigm, was overviewed in Section 3.3. An outline of the GA pseudocode was given, followed by the exploration of various GA evolutionary operators. The core of this chapter was provided in Section 3.4 where the particle swarm optimization (PSO) was described. Several aspects of SI were examined, including the PSO algorithm, topologies, parameters and three modifications (binary PSO, mutating PSO and cooperative split PSO). Lastly, Section 3.5 debated upon the identity of the PSO in a discussion opposing EC and SI paradigms in four different aspects. The material covered in this chapter marks the end of the background material necessary to introduce the work established in this thesis.

Chapter 4

Particle Swarm Optimization for Multiple Sequence Alignment

“You do not really understand something unless you can explain it to your grandmother.”

- Albert Einstein

This short chapter describes the application of particle swarm optimization to the multiple sequence alignment problem. More precisely, data representation schemes and alignment evaluation mechanisms are described in detail.

4.1 Introduction

This chapter shows how the background given in the previous chapters is applied to the MSA. The focus of the chapter is on how PSO can be applied to solve MSA problems.

Section 4.2 describes data representation schemes while Section 4.3 presents methods for alignment evaluations. This chapter ends at Section 4.4 with a short summary.

4.2 Representation Schemes

In order to apply PSO to an MSA, it is necessary to define suitable representations. This section presents two representation schemes that permit the application of PSO to an

MSA.

Throughout this study, all particles from the swarm held a well defined representation. Each element of a particle represents the relative position where one gap is to be inserted. Hence, gap positions are the variables that are optimized. The position index within input sequences always starts at 0.

Each particle has a dimension of *totalGaps*, where

$$totalGaps = k(gapsAllowed) + gapFiller \quad (4.1)$$

where *gapsAllowed* is a fixed parameter that is set to specifically determine, per sequence, the maximum number of gaps that can be inserted. Chellapilla and Fogel [22] observed that common alignment rarely contained a gaps/characters ratio of more than 20%. A pre-processing step is necessary to adjust input sequences since they are commonly of different lengths and valid alignments require that all aligned sequences are of the same length (refer to Section 2.3.1). *gapFiller* represents the number of padding gaps that are needed for each sequence to ensure that all sequences are of the same length.

The choice of this representation offers a dimensionality that is equal to the total number of gaps to be inserted. This method has the primary advantage of being totally independent of the number of sequences to align. If, for example, \mathcal{S}_1 , \mathcal{S}_2 and \mathcal{S}_3 have 2, 3, 2 gaps to be inserted respectively, then the representation is given as:

$$\mathbf{x} = \overbrace{[x_1, x_2, x_3, x_4, x_5, x_6, x_7]}^{totalGaps}$$

$\underbrace{\hspace{1.5cm}}_{\mathcal{S}_1}$

$\underbrace{\hspace{1.5cm}}_{\mathcal{S}_2}$

$\underbrace{\hspace{1.5cm}}_{\mathcal{S}_3}$

where x_i represents the i^{th} component of the position vector \mathbf{x} of a particle. The following example illustrates how the position vector is used to convert an original sequence into the corresponding aligned sequence. Let $\mathcal{S}_1 = \text{"ACTG"}$ be the original sequence ($n = 4$) with given gap positions $\{4, 0, 3\}$. Each gap position represents the index at which one gap must be inserted within the *original* sequence. Positions are sorted to give $\{0, 3, 4\}$. Gaps are then inserted accordingly in the original sequence (after padding as required), as illustrated below:

1. Position at index 0, "ACTG" \rightarrow "-ACTG".
2. Position at index 3, "-ACTG" \rightarrow "-ACT-G".

3. Position at index 4, “-ACT-G” \rightarrow “-ACT-G-” which becomes $\widehat{\mathcal{S}}_1$ with $\widehat{n} = 7$.

This study investigates two different search spaces, namely integer-valued and bit-valued search spaces.

4.2.1 Integer Search Space

The default search space used in this study is defined using integer-valued numbers. The domain of each dimension of the search space is $[0, \text{longestLength}]$, where *longestLength* is the length of the longest sequence in the input set. The rationale of this choice of range is that gaps are allowed to be inserted at any given position limited by the span of the original – potentially padded – sequence.

It is possible for particles to move beyond the boundaries of the search space. All negative values are clipped to 0. However, no boundary enforcement was applied for values higher than *longestLength* because positions that are equal to or greater than *longestLength* automatically append a gap at the end of a sequence, as explained in the next paragraph.

Elements of particle position vectors are floating-point numbers. It is therefore necessary to discretize particle positions, which is done by rounding off their real-valued elements. When a gap is designated to be inserted at index position $\geq \text{longestLength}$, the gap is appended at the end of the sequence. Conversely, a gap at position 0 will be inserted in the sequence as the first (leading) character.

4.2.2 Binary Search Space

As described in Section 3.4.4, the B-PSO requires a bit-valued search space. The binary representation is different from the integer-valued representation with regards to the following points. In the B-PSO, the components of the position vector hold bits. Therefore, the bit vector must be translated into integer-valued gap positions. Since gap positions range within $[0, \text{longestLength}]$, several bits are needed to represent integer values within this range. As an example, if *longestLength* = 15, then 4 bits are needed to code values ranging from 0 (0000) to 15 (1111). Thus, the number of bits needed equals $\lceil \log_2 \text{longestLength} \rceil$. Each group of 4 bits in the particle position vector therefore represents one gap position.

This representation has the drawback of growing in dimensionality as the size of the domain increases.

4.3 Fitness Evaluation

Prior to the evaluation of candidate alignments, columns containing only gaps are removed (according to Definition 2.4). An objective function for MSA is a function that maps an aligned set of sequences to a real number, which is designed to produce larger values for better alignments. The objective is then to maximize the MSA objective function, defined as

$$Fitness = \xi_1(SymbolScore) - \xi_2(GapScore)$$

where *SymbolScore* can be one of the methods previously described in Section 2.3.2, developed to reward character matches, namely the similarity or matches approaches. Additionally, a *PID* weight, as specified in Section 2.3.3, can be applied to the *SymbolScore* in an effort to eliminate potential bias caused by the relatedness of sequences. *GapScore* is obtained by applying one of the two gap penalizing methods described in Section 2.3.3, namely the *aGP* (affine gap penalty) and the *lGP* (linear gap penalty) approaches. Note that the *lGP* can be forced either by setting a penalty of 0 for gaps when using the similarity method or by setting a value of 0 for g_{ext} when using the *aGP* approach. A major problem that emerges from the use of the *aGP* approach has to do with finding the best values for its gap opening and extension penalty parameters. The best values for these parameters remain largely problem-dependent. Also, two proportionality variables, ξ_1 and ξ_2 , act as independent weights to allow for flexibility in terms of favouring the optimization of either one of the two scores. The objective is to simultaneously maximize the *SymbolScore* (i.e. the overall score for the number of matched symbols over all columns) and to minimize the *GapScore* (i.e. the number of all gaps over all columns).

The presented fitness evaluation aggregates two conflicting objectives as one objective. However, a multiple objective optimization (MOO) algorithm [26] would optimize each of the two conflicting objectives simultaneously to yield multiple non-dominating solutions. Since an aggregation-based approach showed to be sufficient for the MSA problem (refer to Chapter 2), an investigation into Pareto-based MOO methods is left

for future work (refer to Section 6.2).

To find the best biologically accurate objective function lies beyond the scope of this research. In this work, an alignment that attained the best possible evaluation score is considered to be optimal, whether it is biologically accurate or not. Recall that biological accuracy means that biochemical properties between sequences are taken into account. In general, the production of an optimal sequence alignment embodies a trade-off between computational efficiency, biological accuracy and score optimality.

For any MSA problem, an optimal alignment (i.e. global optimum solution) *always* exists. In addition to at least one global optimum, multiple local optima or *sub-optimal* solutions that are all considered equally good can potentially be produced from the same set of input sequences.

To complete the topic of fitness evaluation, it is worth noting that the order of the input sequences poses no influence on the resulting alignment. Provided that the same initial conditions are used, the alignment will always yield the same score and matches. The reason for this insensitivity to order resides in the way alignments are scored. Since the aligned sequences are evaluated in a pairwise fashion, two sequences at a time, the pairwise scores will simply remain the same but might be added in a permuted order which does not affect the results in a mathematical addition.

4.4 Summary

This chapter showed how PSO can be used to solve the MSA problem. Two representation schemes were given in Section 4.2, while Section 4.3 presented the corresponding fitness functions. The next chapter discusses the experimental work.

Chapter 5

Empirical Analysis

“I have not failed. I’ve just found 10,000 ways that won’t work.”

- Thomas Alva Edison

The core of all experimental work conducted for this thesis is presented in this chapter. Tests start with a comparatively small alignment (of low complexity) to test the viability of PSO as a valid aligner. Further experiments increase the complexity of the alignments, to identify strengths and pitfalls of PSOs occurring in larger alignments. The analysis of results and solutions to potential weaknesses in alignment strategies are investigated.

5.1 Introduction

Sets of experiments can now be conducted to determine the viability of using PSOs as an efficient and robust approach for solving MSAs.

Thus far, the closest PSOs have come to solving MSAs is in Hsiao and Chuang [55], where an unknown version of PSO was used to solve MSAs. Unfortunately, little detail is provided on how this was done. Considering the lack of information, no comparison is done between Hsiao and Chuang’s work and the empirical work enclosed in this thesis.

As already mentioned, the complexity of the MSA problem originates from the combination of the following criteria:

1. The relatedness and nature of the sequences to be aligned.
2. The length and number of sequences to be aligned.
3. The choice of the objective function (OF).

Since this work has no prior reference in terms of PSOs solving MSAs, this research started with zero knowledge. Considering the knowledge at hand on PSO and MSA, empirical analysis has been applied in an effort to derive results in a cognitive fashion. The experimental procedure starts with problems of low complexity and incrementally increases the complexity as the experiments progress. To achieve complexity increments, the criteria enumerated above will be adjusted individually to obtain the desired complexity. Each experimental step will represent the concretization of a ‘mini’ trial experiment (upon success or failure of the alignment optimization), thereby also creating a new starting point for the next experiment based on previous and current observations. Experiments in this work are then derived empirically from the step before and will determine the step thereafter. Conclusions are subsequently drawn for every experimented objective. Also, potential PSO limitations and pitfalls are identified, along with a discussion to understand and possibly overcome them. Lastly, PSO-generated MSA solutions are systematically submitted for comparison (benchmarking) with other third-party MSA programs.

Section 5.2 conducts the first set of experiments that intends to evaluate the influence of individual S-PSO and B-PSO characteristics on a low complexity MSA data set. The second part of experiments conducts a scalability analysis in Section 5.3, where high complexity data sets and various kinds of PSOs are tested. Section 5.4 closes the chapter by providing a short summary.

5.2 S8 Analysis

This section is dedicated to experimenting exclusively on S8. Section 5.2.1 gives details on data set S8. Characteristics for both PSO and MSA are listed and motivated in Section 5.2.2. Next, the experimental procedure is provided in Section 5.2.3. The first set of experiments produced results from the similarity and matches methods using the S-PSO (see Section 5.2.4). The B-PSO is applied in Section 5.2.5, also using the similarity

and matches methods. Concluding on S8 experiments, Section 5.2.6 provides a brief summary and sheds light upon the future direction of experiments.

5.2.1 DNA Data Set S8

As priorly mentioned, the major aim of the preliminary experiments in this chapter is to validate the application of PSO as a viable MSA solver. The first candidate data set is the computationally low DNA set, S8, as displayed in Figure 5.1. Data set S8 has no reference because it was not extracted from a biological source. S8 was artificially generated for the purpose of this thesis. The legend above the sequences indicates the frequency of matches for each character in the corresponding column. Characters are scaled larger for more matches. The number at the end of each sequence represents the index, which stops at the last character.

Table 5.1: Properties of DNA data set S8

<i>ID</i>	<i>Type</i>	<i>k</i>	<i>Length_{mean}(min, max)</i>	<i>Similarity %</i>
S8	DNA	5	8 (7,10)	52.5 (medium)

Data set S8 contains 5 very short DNA sequences of varying lengths. S8 totals 48 characters and has a medium similarity of 52.5%.

Characteristics of S8 are summarized in Table 5.1. Note that properties of input data sets are always summed up using the same table headings. These headings have the following meaning:

ID – The name of the data set.

Type – The biological nature of the sequences, i.e. DNA.

k – The number of sequences in the set.

Length_{mean}(min, max) – The computed mean length of the input sequences followed by an interval stating minimal and maximal lengths within the set.

Similarity % – An indicator of the global similarity level among all input sequences. The scale is as follows: (1) very low (< 25 %); (2) low (from 25% to 44%); (3)

medium (from 45% to 64%); (4) high (from 65% to 84%); and (5) very high ($\geq 85\%$).

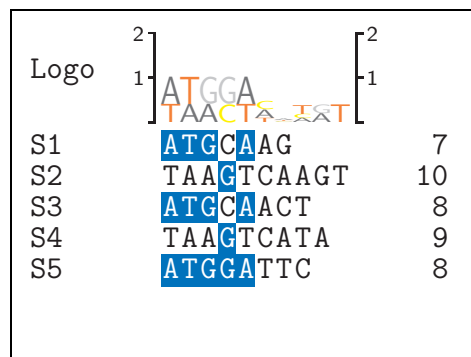


Figure 5.1: Visualization of DNA data set S8

With reference to the complexity criteria enumerated in Section 5.1, S8 exhibits a low complexity alignment problem due to (1) its small number of sequences, (2) its short sequence length and (3) a medium similarity percentage. Computationally, the objective function has also been kept to a bare minimum by means of excluding sequence weighting when calculating the *SymbolScore*.

5.2.2 Characteristic Selection

The experimental work presented in this study is based on the analysis of the influence of selected characteristics for both PSOs and MSAs. All PSO aspects and parameters discussed in Sections 3.4.2 and 3.4.3 are separately investigated. Two MSA characteristics were investigated, namely the *SymbolScore* method and the two ξ weights. For this purpose, a ‘reference configuration’ has been used to serve as a reference point against which the effect of different characteristics are compared.

PSO Characteristics

Characteristics of the PSO reference configuration have been assigned default starting values to be further optimized empirically:

- PSO algorithm: Standard PSO (S-PSO)

- Topologies: LBest topology (asynchronous, neighbourhood size = 3)
- Swarm size: 25
- Number of iterations: 1,000 (i.e. 25,000 objective function evaluations)
- V_{max} : No clamping applied
- Velocity vector initialization: 0.0
- Inertia weight: Constant of 0.729844
- Cognitive coefficient c_1 : Constant of 1.496180
- Social coefficient c_2 : Constant of 1.496180

MSA Characteristics

The following default starting values were applied to MSA characteristics:

- Sequence weight: None
- ξ_1 and ξ_2 : 1.0
- *SymbolScore*: Similarity with rewards and penalties in accordance with Section 2.3.2 where a match is rewarded +2, a mismatch penalty of 0 is used and a gap penalty of -1 is used.
- *GapScore*: aGP with $g_{op} = +2$ and $g_{ext} = +1$
- *#gapsAllowed*: 3, which amounts to 30% of the longest sequence length, which is 10.

5.2.3 Experimental Procedure

All algorithms, problems and measurements utilized throughout this research were performed using the computational intelligence library (Cilib) [91]. Cilib represents a collective project aimed at building a generic framework designed to accommodate the development of CI algorithms. Written in the Java 1.5 programming language, Cilib

further allows CI simulations to be performed on various problems. This work made use of Cilib version 0.6.5, which is freely available at <http://cilib.sourceforge.net>. Refer to Appendix B for an example of a typical Cilib simulation file.

Experiments in this research took the form of running simulations of CI algorithms that optimize several MSA problems. Each simulation was run, as a standard, 30 times to ensure statistical soundness and meaningful results. Results obtained for each experiment were as follows:

Best: the maximum score achieved during the course of 30 simulations.

Mean: the average score computed over 30 simulations.

CI: the confidence interval calculated with $\alpha = 0.95$ using the t-test (confidence of 95%).

All simulations are defined as a variation of the reference configuration. A variation means that one or more characteristics found in the reference configuration is changed while all other characteristics remain unchanged.

In the context of this work, the term ‘univariate’ means that results are obtained by testing the influence of a single characteristic at a time. Univariate results come from testing different values of a single characteristic while keeping all other characteristics identical.

Lastly, this work uses the term ‘instable’ as a way to express the fact that results obtained over all simulations exposed a large confidence interval, which means that results vary extensively from each other. Instable results are consequently not reliable. Conversely, the term ‘stable’ means that results obtained over all simulations were similar and exposed a small confidence interval, which makes stable results reliable.

5.2.4 Experimental Results for S-PSO

This section presents the first results obtained from aligning S8 using S-PSO. The aim was to change one variable at a time to observe exactly the influence of individual characteristics. Two tables display the results. Both tables only differ by the method employed to compute the *SymbolScore* term of the objective function. Table 5.2 displays results for the similarity method, while Table 5.3 displays results for the matches method

with the *#gapsAllowed* parameter set to 3 and 8 (splitting both tables vertically). In an effort to facilitate the browsing of results, all variations that outperformed the reference configuration in terms of the *Mean* fitness were indicated in bold in the relevant rows of all tables. Note that the arrow symbols ‘ \nearrow ’ and ‘ \searrow ’ respectively mean linearly increasing and decreasing *over time*. Also, these arrow symbols are associated with a range of values which is indicated with square brackets separated by a comma, such as ‘[*from*, *to*]’. For space constraints, the terms ‘asynchronous’, ‘synchronous’ and ‘domain’ have been abbreviated ‘async’, ‘sync’ and ‘dom’, respectively.

With reference to Section 4.2, all the variations that deal with *#gapsAllowed* = 3, the search space remains of a low complexity with 23 dimensions (i.e. maximum of 23 gaps to insert). As for variations dealing with *#gapsAllowed* = 8, the dimensionality increases to 48 (roughly doubling the number of gaps that can be inserted) to observe the effect of increasing the complexity. Care should be given when setting the *#gapsAllowed* parameter because of the fact that the problem dimensionality is multiplied proportionally to the number of sequences involved. For example, to add only 2 more gaps to a current *#gapsAllowed* would result in 20 more dimensions to an MSA problem that contains a set of 10 sequences. Refer to Section 4.2 for the mathematical relationship between the number of gaps, dimensionality and the number of sequences.

***SymbolScore* with Similarity Method**

Table 5.2 provides the results of all single characteristic variations (one on each row) and the reference configuration (last row of the table). The following observations in comparison with the reference configuration can be made:

- **Topologies:** The reference topology (LBest) was found to perform the best, especially when *#gapsAllowed* = 8, but not significantly for *#gapsAllowed* = 3. The Von Neumann topology performed efficiently with 3 gaps allowed, considering that it compared slightly worse but statistically no different than the LBest reference topology. However, the Von Neumann topology showed a clear decline in performance when dealing with 5 more gaps (*#gapsAllowed* = 8). The GBest topology showed instability and remained statistically the worst performer, a fact confirmed by its drastic performance decline observed when *#gapsAllowed* was set to 8.

Table 5.2: Performance table for S-PSO optimizing the S8 alignment based on the similarity method with 3 and 8 gaps

<i>Variations</i>	#gapsAllowed = 3			#gapsAllowed = 8		
	<i>Best</i>	<i>Mean</i>	<i>Ci</i> [lo, hi]	<i>Best</i>	<i>Mean</i>	<i>Ci</i> [lo, hi]
Von Neumann (async)	55.0	51.40	[49.82062,52.97938]	55.0	50.90	[49.17873,52.62128]
GBest (sync)	55.0	47.46667	[42.85819,52.07515]	55.0	35.36666	[25.90489,44.82844]
Swarm of 50 particles	55.0	54.70	[54.12129,55.27871]	55.0	54.0	[53.06590,54.93410]
Swarm of 10 particles	55.0	51.13334	[49.35731,52.90936]	55.0	35.76667	[27.61305,43.92028]
100 iterations	55.0	51.10	[49.75222,52.44778]	55.0	48.76667	[46.95736,50.57597]
2500 iterations	55.0	54.40	[53.59582,55.20418]	55.0	54.03333	[53.06557,55.00109]
V_{max} clamped [± 4]	55.0	51.66667	[49.23130,54.10203]	55.0	36.70	[29.41082,43.98918]
V_{max} clamped [$\pm \text{dom}$]	55.0	52.90	[51.32401,54.47599]	55.0	51.16667	[49.44357,52.88976]
$V_{max} \searrow [\text{dom}, +4]$	55.0	54.50	[53.81564,55.18436]	55.0	51.26667	[49.37813,53.15521]
$V_{max} \searrow [+100, \text{dom}]$	55.0	53.30	[52.06898,54.53101]	55.0	51.93333	[50.42506,53.44160]
$\mathbf{v}_i(0) \sim U(\mathbf{v}_{min}, \mathbf{v}_{max})$	55.0	55.0	[55.0,55.0]	55.0	55.0	[55.0, 55.0]
Large inertia=0.9	55.0	52.53333	[51.11630,53.95037]	55.0	51.40	[49.89552,52.90449]
Small inertia=0.3	55.0	42.46667	[38.76393,46.16941]	19.0	-17.80	[-24.40274,-11.19725]
\nearrow inertia [0.3,0.9]	55.0	46.50	[43.21597,49.78403]	55.0	3.83333	[-4.17067,11.83733]
\searrow inertia [0.9,0.3]	55.0	51.56667	[50.02821,53.10512]	55.0	51.03333	[49.56953,52.49713]
Large cognitive=2.5	55.0	54.80	[54.41419,55.18580]	55.0	52.63334	[51.26311,54.00357]
Small cognitive=0.5	55.0	51.10	[49.57645,52.62355]	55.0	48.96667	[47.07688,50.85645]
\nearrow cognitive [0.5,2.5]	55.0	52.83333	[51.47691,54.18976]	55.0	51.66667	[50.17436,53.15897]
\searrow cognitive [2.5,0.5]	55.0	53.43333	[52.23566,54.63101]	55.0	52.90	[51.68931,54.11070]
Large social=2.5	55.0	50.60	[49.01252,52.18748]	55.0	49.70	[48.21397,51.18603]
Small social=0.5	55.0	23.70	[21.31106,26.08894]	10.0	-25.56667	[-30.82288,-20.31046]
\nearrow social [0.5,2.5]	55.0	48.83333	[46.69588,50.97079]	55.0	46.36666	[41.86642,50.86691]
\searrow social [2.5,0.5]	46.0	43.16667	[40.57023,45.76311]	46.0	34.70	[30.10973,39.29027]
reference configuration	55.0	53.80	[52.70409,54.89591]	55.0	54.16667	[53.39793,54.93541]

- Number of particles (swarm size): Small swarm sizes proved to be instable and statistically highly inferior when dealing with the 5 gaps increase from 3 to 8 but were nevertheless able to perform satisfactorily otherwise. The largest swarm was always statistically superior to the smallest swarm but equal to the reference swarm (25 particles). Even so, the reference swarm was outperformed based on the *Mean* fitness by a swarm double the size when dealing with *#gapsAllowed* equal to 3 but gave a better *Mean* fitness when increased to 8.
- Number of iterations: The general trend suggests that results obtained with runs of 100 iterations are the worst and always statistically inferior. The scores from the reference number of iterations (1000) were statistically equal to the scores of

the 2500 iterations variation. However, the 2500 iterations variation showed the best *Mean* fitness score when dealing with *#gapsAllowed* set to 3, but not for 8 where the reference configuration performed best.

- V_{max} : Four different velocity clamping strategies have been investigated. Results failed to create a definite pattern. Severe clamping restricted velocities with fixed values as in the two first rows of velocity clamping variations, yielding good, statistically identical results for 3 gaps and 8 gaps except for clamping using $V_{max} \pm 4$, which significantly degraded for 8 gaps. For the dynamic clamping strategies, the *Mean* fitnesses for 3 gaps outperformed the reference in the linear decreasing from domain length to +4 strategy. The increase to 8 gaps resulted in lower scores, where the reference clamping strategy (no clamping at all) performed better overall.
- Velocity initialization: The random velocity initialization variation consistently and statistically outperformed the reference variation. Furthermore, this variation was shown to be extremely stable since all simulations converged on the average (standard deviation of zero). In addition to always finding the best solution, this variation also performed very fast. On average, 205 iterations were required to find a solution.
- Inertia weight: For *#gapsAllowed* = 3, it appeared that small weights invariably led to instable and statistically inferior results. When the *#gapsAllowed* was increased to 8, it was clearly observed that starting with or keeping a low inertia weight led to very poor performance. Large weights for 3 allowed gaps showed performance statistically equal to the reference inertia weight (fixed at 0.729844). Also, large weights were minimally affected by the increase in gaps. However, with 8 gaps allowed, the reference inertia weight achieved statistically the best results. Note that linearly increasing or decreasing the weights over time did not show a clear advantage over fixed weights.
- Cognitive coefficient c_1 : With *#gapsAllowed* = 3, all results were statistically equal to the reference cognitive coefficient (fixed at 1.496180) except for one, where the smallest coefficient of 0.5 was applied. The best *Mean* fitness, 54.8, was achieved

by the large coefficient variation. With 8 gaps allowed, small and high c_1 's declined slightly in performance and were all inferior to the *Mean* result of the reference c_1 . Again, note that linearly increasing or decreasing the coefficient values over time did not show a clear advantage over fixed weights.

- Social coefficient c_2 : Testing the four social variations highlighted the negative impact of applying a small social coefficient value. Specifically with $\#gapsAllowed = 8$, small values obtained the worst results, while larger values still produced good results. Note that the fixed variation with $c_2 = 0.5$ produced the worst results. The reference value $c_2 = 1.496180$ was consistently statistically better for both 3 and 8 gaps allowed, followed by the large fixed value variation with $c_2 = 2.5$. Lastly, linearly increasing or decreasing the social coefficient provided mixed results, considering that a low c_2 resulted in a degradation in performance.

Interestingly, the only variation that produced better performance for an increase in the number of gaps allowed is the reference configuration. Before proceeding to the next set of experiments, it is important to gain an idea of what the best S8 alignment found so far looks like. Figure 5.2 depicts the best S8 alignment found by optimizing the objective function that used the similarity method to compute the *SymbolScore*.

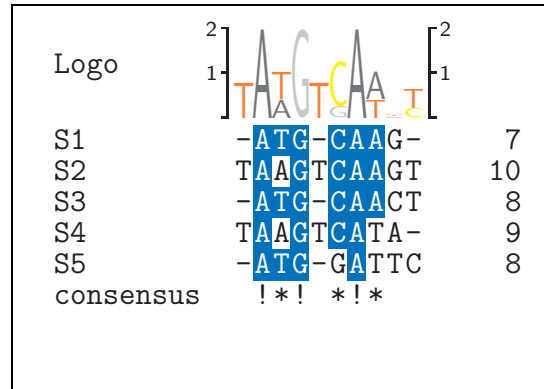


Figure 5.2: Best alignment of S8 optimized by the OF using the similarity method

In Figure 5.2 and all subsequent visualization figures, a consensus line at the bottom is used to provide a quick impression of the quality of the depicted sequence alignment. The consensus is determined column-wise, where a particular character has been assigned

to represent each column. These characters can either be an exclamation mark ‘!’, an asterisk ‘*’ or a space ‘ ’ indicating a fully matched column, a highly matched column or a slightly/not matched column respectively. This specific alignment achieved three fully aligned columns as well as three highly aligned columns, and receiving a score of 55.0.

Lastly, the influence of the weights ξ_1 and ξ_2 in the objective function is discussed. Comparisons between the reference configuration ($\xi_1, \xi_2 = 1.0$) and two variations were conducted. The first variation applied a small value of $\xi_1 = 0.25$ and a large value of $\xi_2 = 0.75$. Conversely, the second variation used $\xi_1 = 0.75$ and $\xi_2 = 0.25$. The criteria for the comparison were based on the average number of matches and gaps found in solutions over 30 simulations. Figure 5.3 offers a graphical summary of the analysis.

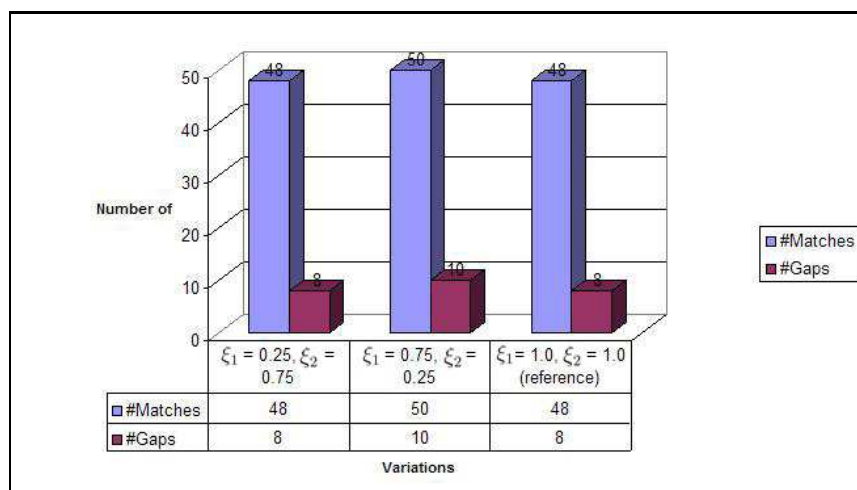


Figure 5.3: Graphical representation of the influence of weights ξ_1 and ξ_2 on S8 (similarity method)

The results indicated that a large ξ_1 and a small ξ_2 effectively biased the search process towards solutions that contained more matches, but using more gaps. The reference configuration achieved the same number of matches and gaps than the variation with a small ξ_1 and a large ξ_2 . In summary, weights can bias the search towards more matches/fewer gaps, but without a statistically significant difference.

SymbolScore with Matches Method

This new set of experiments differs from the previous set by the *SymbolScore* method used, which, in this case, is the matches method. The same variations used in the previous experiment were investigated for the matches method.

Table 5.3: Performance table for S-PSO optimizing the S8 alignment based on the matches method with 3 and 8 gaps

<i>Variations</i>	#gapsAllowed = 3			#gapsAllowed = 8		
	<i>Best</i>	<i>Mean</i>	<i>Ci</i> [<i>lo</i> , <i>hi</i>]	<i>Best</i>	<i>Mean</i>	<i>Ci</i> [<i>lo</i> , <i>hi</i>]
Von Neumann (async)	106.4	103.15331	[100.55110,105.75552]	108.0	103.83998	[101.62962,106.05034]
GBest (sync)	108.0	104.47998	[101.76031,107.19965]	108.0	89.82667	[83.16093,96.49241]
Swarm of 50 particles	108.0	106.57332	[106.40105,106.74559]	108.0	106.61331	[106.41848,106.80814]
Swarm of 10 particles	108.0	99.55332	[95.75585,103.35079]	108.0	80.61333	[73.55638,87.67027]
100 iterations	108.0	104.13998	[102.50021,105.77975]	106.4	98.02665	[95.26260,100.79069]
2500 iterations	108.0	106.7732	[106.53091,107.01573]	108.0	105.78665	[104.86827,106.70504]
V_{max} clamped [± 4]	108.0	105.77999	[104.66376,106.89622]	108.0	94.75999	[89.90499,99.61500]
V_{max} clamped [$\pm \text{dom}$]	108.0	106.00665	[104.80635,107.20694]	108.0	102.95999	[100.68066,105.23932]
$V_{max} \searrow [\text{dom}, +4]$	108.0	106.10664	[105.09622,107.11707]	108.0	100.47332	[96.61441,104.33223]
$V_{max} \searrow [+100, \text{dom}]$	108.0	106.33998	[105.58289,107.09707]	108.0	101.01331	[96.71873,105.30790]
$\mathbf{v}_i(0) \sim U(\mathbf{v}_{min}, \mathbf{v}_{max})$	106.4	106.39998	[106.39997,106.39999]	108.0	106.50664	[106.36367,106.64960]
Large inertia=0.9	108.0	106.6665	[106.45305,106.88024]	108.0	106.52664	[106.38042,106.67286]
Small inertia=0.3	108.0	93.89999	[89.14039,98.65959]	84.4	53.840	[48.47974,59.20026]
\nearrow inertia [0.3,0.9]	108.0	100.860	[97.03580,104.68420]	101.0	65.94667	[59.93676,71.95658]
\searrow inertia [0.9,0.3]	106.4	105.73997	[104.46682,107.01313]	108.0	105.95332	[105.18801,106.71862]
Large cognitive=2.5	108.0	106.51998	[106.37598,106.66398]	108.0	105.76665	[104.73009,106.80320]
Small cognitive=0.5	106.4	102.83999	[100.60975,105.07023]	108.0	99.41335	[96.63349,102.19320]
\nearrow cognitive [0.5,2.5]	108.0	105.51999	[103.99623,107.04375]	106.4	96.26667	[91.72136,100.81198]
\searrow cognitive [2.5,0.5]	108.0	106.45332	[106.35043,106.55620]	108.0	105.74665	[104.82066,106.67264]
Large social=2.5	106.4	105.97998	[105.16979,106.79017]	108.0	106.50665	[106.36368,106.64961]
Small social=0.5	108.0	80.78669	[75.54714,86.02624]	65.4	42.020	[37.46303,46.57696]
\nearrow social [0.5,2.5]	108.0	97.95999	[93.96070,101.95928]	106.4	95.17334	[91.38480,98.96188]
\searrow social [2.5,0.5]	106.4	83.360	[78.68799,88.03201]	106.4	73.280	[66.05570,80.50429]
reference configuration	108.0	106.60664	[106.41046,106.80283]	108.0	102.68666	[99.69510,105.67822]

Table 5.3 provides the results of all single characteristic variations (one on each row) and the reference configuration (last row of the table). The following observations in comparison with the reference configuration can be made:

- Topologies: With only 3 gaps allowed, the GBest variation had a slightly better *Mean* fitness than the Von Neumann one, although the GBest variation was more instable. However, with 8 gaps allowed, the Von Neumann topology showed a

minimal drop in performance, whereas the GBest topology became statistically the worst on account of a massive drop in performance. The reference topology (LBest) achieved better *Mean* fitnesses than the Von Neumann and GBest topologies when dealing with only 3 gaps, but showed a drop in performance by 4% in consequence of the additional 5 gaps.

- Number of particles (swarm size): For $\#gapsAllowed = 3$, doubling the size of the swarm did not affect the performance in comparison to the 25 particles in the reference configuration. The smaller swarm containing 10 particles scored reasonably well, but was still largely statistically outperformed by all larger swarm sizes. For $\#gapsAllowed = 8$, 10 particles produced the worst score. The clear trend here was that the more particles used, the better the scores (the 50 particles variation was significantly better than the reference configuration).
- Number of iterations: A pattern emerged that 2500 iterations yielded better *Mean* fitnesses than 100 and 1000 iterations, and proved to be proportionally less affected by the increase in complexity when using 8 gaps. However, the 2500 iterations variation was statistically equal to the reference configuration for both 3 and 8 gaps allowed.
- V_{max} : In all cases, the four variations were statistically equal to the reference configuration (using no velocity clamping), except for the ± 4 clamping variation with 8 gaps allowed. For $\#gapsAllowed = 8$, performance deteriorated a little except for the variation where V_{max} was clamped on the domain, which was not significant. With only 3 gaps, the performance of the reference configuration had the highest *Mean* fitness but was nonetheless closely followed by the other variations.
- Velocity initialization: As suspected from previous experiments using the similarity approach, the random initialization variation performed extremely well. Performing statistically better than the reference configuration with 8 gaps allowed, this random variation demonstrated an increase in performance when increasing from 3 to 8 gaps.
- Inertia weight: The variation testing the fixed inertia weight of 0.9 obtained the best overall *Mean* fitnesses. Also, the variation with $\omega = 0.9$ and the variation with

the decreasing ω were both statistically superior to the reference configuration when 8 gaps are allowed. The small inertia variation achieved statistically the lowest scores which were again worsened by an increase of gaps. Linearly increasing and decreasing the weights logically produced a mixed performance, since the large fixed ω obtained good results while the small fixed ω obtained the worst results. However, the results showed that increasing the inertia weight from 0.3 to 0.9 yielded worse results than decreasing it from 0.9 to 0.3.

- Cognitive coefficient c_1 : The results showed that higher values generally performed better than lower values. With $\#gapsAllowed = 3$, all variations were statistically identical to the reference configuration, except for $c_1 = 0.5$ which produced inferior results. All *Mean* fitnesses had minor losses of performance when the number of gaps increased to 8. With 8 gaps allowed, the best *Mean* fitnesses were achieved by variations keeping or starting with a value of 2.5. However, the reference configuration using $c_1 = 1.496180$ was only outperformed by the variation with $c_1 = 2.5$ when $\#gapsAllowed = 8$.
- Social coefficient c_2 : Small values confirmed the previous observations from the similarity method in that performance significantly dropped when small social coefficients were applied, producing, once again, statistically the worst results. However, for $c_2 = 2.5$, a good *Mean* fitness was obtained for both 3 and 8 gaps. The variation using $c_2 = 2.5$ also statistically outperformed the reference configuration using $c_2 = 1.496180$ with 8 gaps. All other variations were statistically inferior to the reference configuration. Again, linearly decreasing and increasing values provided mixed results.

Figure 5.4 visualizes the best S8 alignment found by the optimization of the OF that used the matches method. The alignment achieved three fully aligned columns as well as four highly aligned columns. A score of 108.0 was obtained. Lastly, to investigate the influence of the weights ξ_1 and ξ_2 in the objective function, the same weights value assignments as for the similarity method have been used.

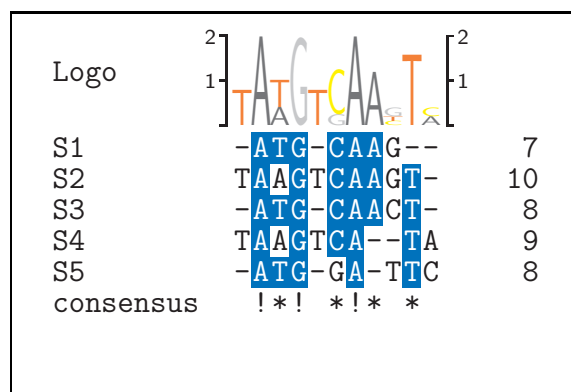


Figure 5.4: Best alignment of S8 optimized by the OF using the matches method

Figure 5.5 offers a graphical summary of the analysis. The results indicated that a small ξ_1 and a large ξ_2 effectively biased the search process towards solutions that contained fewer gaps but also fewer matches. The reference configuration achieved the same number of matches and gaps than the variation with a large ξ_1 and a small ξ_2 . In summary, weights can bias the search towards fewer gaps/more matches, but without a statistically significant difference.

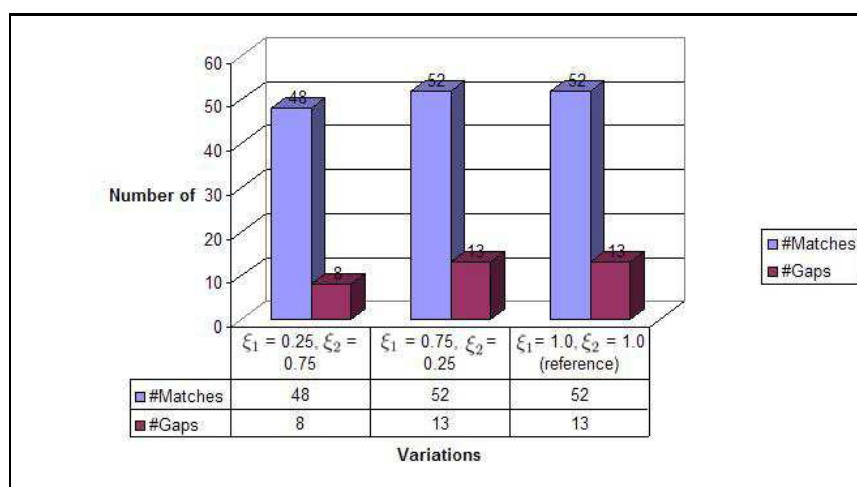


Figure 5.5: Graphical representation of the influence of weights ξ_1 and ξ_2 on S8 (matches method)

Discussion

This section provides a summary of the findings for all experiments that have been conducted to this point. Interestingly, the only similarity method variation that produced a better score as the number of gaps increased from 3 to 8 was the reference configuration. As for the matches method, many variations also produced a better score with more gaps except for the reference configuration. In general, variations performed worse as the number of gaps increased. All these facts indicated that some variations actually performed better as the gaps increased (higher complexity) which is very encouraging for further experiments on more complex MSAs. Furthermore, these facts mean that a S-PSO configuration that is highly efficient when $\#gapsAllowed = 8$ may not necessarily be as efficient for $\#gapsAllowed = 3$. However, while several variations emerged as good candidates for solving larger MSAs, the general trend thus far suggested that the S-PSO copes with MSAs exhibiting small dimensionality extremely well. Questions were therefore raised, such as “*To what extent can promising S-PSO variations solve larger MSAs?*” or “*What if the S-SPO becomes inadequate and demonstrates poor performance in larger MSAs?*” These questions are addressed in Chapter 6.

The impact of the weights, ξ_1 and ξ_2 , on both scoring methods was evaluated. It was shown that there exists a relationship between the weights and the alignment solutions in terms of matches and gaps. There exists a matches to gap ratio in each solution alignment that can be biased towards either increasing the number of matches or decreasing the number of gaps. The weight values that would yield the best matches to gap ratio need to be set empirically and therefore remain largely problem-dependent. The need to find the best weight values would not exist if an MSA problem was optimized by a multiple objective optimization approach (refer to Section 6.2). In effect, the maximization of matches and minimization of gaps would be optimized as two conflicting weightless objectives and not as one aggregated objective.

Comparative Results

The results obtained by S-PSO are compared to that of CLUSTAL X (version 1.83) and the GA-based T-COFFEE (version 4.99), described respectively in Sections 2.6.2 and 2.6.4. The output solutions from CLUSTAL X and T-COFFEE are illustrated

in Figures 5.6 and 5.7 respectively. CLUSTAL X was used locally and was run only once due to the deterministic nature of its algorithm. T-COFFEE was run online at <http://www.igs.cnrs-mrs.fr/Tcoffee/tcoffee.cgi/index.cgi>. T-COFFEE always returned the same result for repetitive runs. Therefore, no average and only one result was taken.

Both CLUSTAL X and T-COFFEE use different scoring mechanisms, which also differ from that used in S-PSO. It is therefore not possible to compare the performance of these algorithms based on their scores. Instead, to allow fair comparison, the number of matches (*#Matches*), gaps (*#Gaps*) and fully matched columns (*#Full_columns*) are used in Table 5.4. The table also provides top S-PSO scores computed with the similarity (*Sim_score*) and the matches (*Match_score*) methods for reference purposes.

Table 5.4: Comparison between alignments of S8 produced by S-PSOs, T-COFFEE and CLUSTAL X

<i>MSA_technique</i>	<i>#Matches</i>	<i>#Gaps</i>	<i>#Full_columns</i>	<i>Sim_score</i>	<i>Match_score</i>
S-PSO – Similarity	48	8	3	55.0	106.4 (M_conv)
S-PSO – Matches	52	13	3	38.0 (S_conv)	108.0
T-COFFEE	43	13	2	24.0	80.4
CLUSTAL X	31	13	1	4.0	44.4

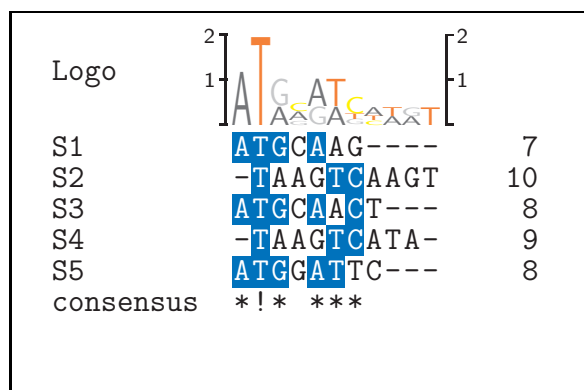


Figure 5.6: Visualization of aligned data set S8 by CLUSTAL X

Table 5.4 clearly emphasizes the dominance of both S-PSO methods. Both S-PSO methods successfully balanced the two objectives of maximizing matches and minimizing

gaps. The similarity method, however, gave a larger priority to the minimization of gaps, having achieved the lowest number of gaps, while the matches method gave a larger priority to the maximization of matches, having achieved the highest number of matches.

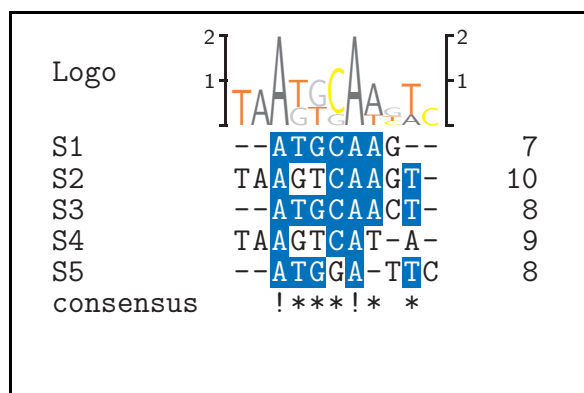


Figure 5.7: Visualization of aligned data set S8 by T-COFFEE

T-COFFEE ranked third due to having the worst number of gaps and only a satisfactory number of matches. Two fully matched columns were achieved by T-COFFEE while both S-PSOs obtained three fully matched columns. CLUSTAL X, ranking fourth, only achieved one fully matched column on account of its poor number of matches and despite having the lowest number of gaps.

It is worth noticing that a top similarity score of 55.0 was equivalent to a score of 106.4 when measured with the matches method ('M_conv' in Table 5.4). However, the opposite could not be verified: the top matches score of 108.0 only obtained an equivalent similarity score of 38 ('S_conv' in Table 5.4). This low score accentuated the fact that the matches method primarily focused on achieving more matches rather than fewer gaps. No clear winner between the S-PSO methods was declared, because their respective top solutions were both considered to be optimal.

Progression of S8 Alignment

It is instructive to follow a step-by-step alignment optimization process. Figures 5.8 to 5.14 show, in increments of 50 iterations, the progression of a S8 alignment using the random velocity initialization variation and the similarity method ($\#gapsAllowed = 8$).

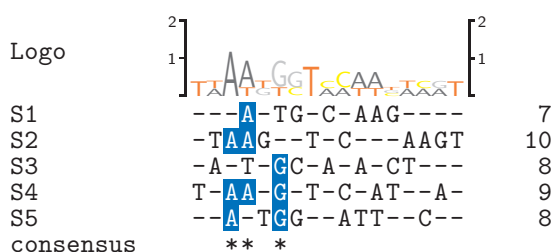


Figure 5.8: Alignment of S8 (best similarity variation) – first iteration

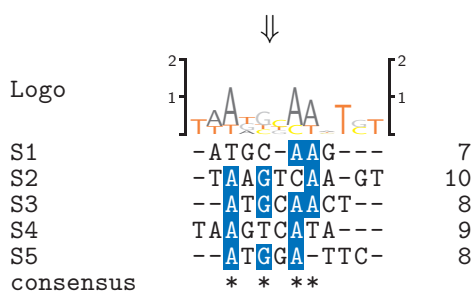


Figure 5.9: Alignment of S8 (best similarity variation) – 50th iteration

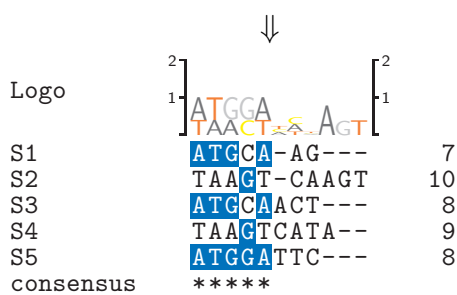


Figure 5.10: Alignment of S8 (best similarity variation) – 100th iteration

Observe how the initial gaps are randomly scattered and then progressively removed or moved to positions that developed into an improved alignment. Figure 5.15 depicts a graphical summary of the S8 alignment progression that clearly contrasts between the maximization of matches and the minimization of gaps throughout the S-PSO iterations.

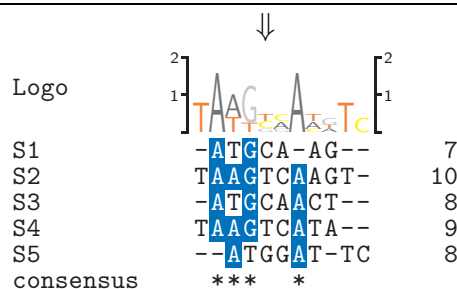


Figure 5.11: Alignment of S8 (best similarity variation) – 150th iteration

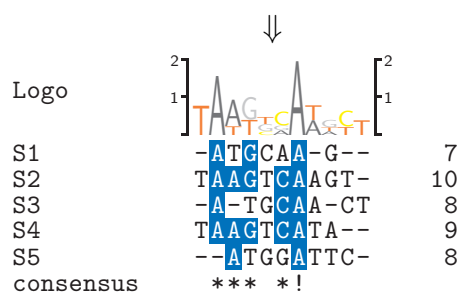


Figure 5.12: Alignment of S8 (best similarity variation) – 200th iteration

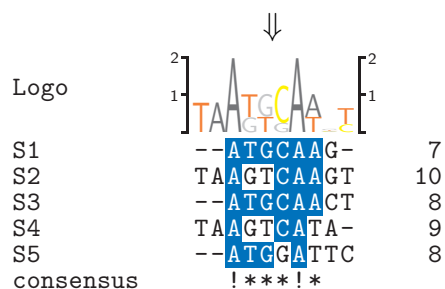


Figure 5.13: Alignment of S8 (best similarity variation) – 250th iteration

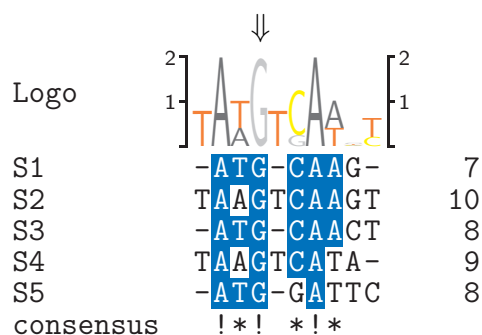


Figure 5.14: Final alignment of S8 with similarity method (55.0) – last iteration (272th)

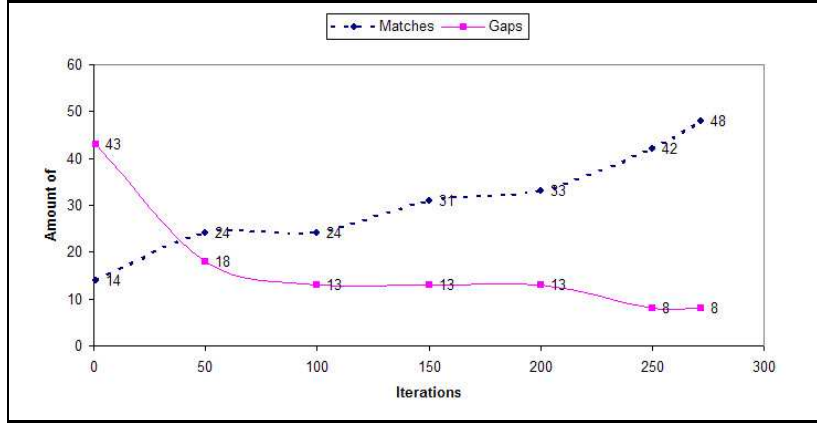


Figure 5.15: Progression in sub-objectives for S8

5.2.5 Experimental Results for B-PSO

This section applies the binary PSO (B-PSO), as detailed at Section 3.4.4, to solve the MSA problem. To the knowledge of the author, this is the first application of B-PSO to solve MSAs.

For the purpose of this section, the reference configuration as for the S-PSO has been used. Additionally, velocities were clamped on the range $[-4, +4]$.

The univariate results consisted in testing single PSO characteristics. Univariate results consisted in testing the following PSO characteristic variations: different topologies, different swarm sizes, and different velocity initialization schemes. Additionally, a newly introduced configuration called ‘explorer’ is compared.

The explorer configuration has the following characteristics:

- number of iterations is 2000;
- a Von Neumann topology;
- a linearly increasing V_{max} ranging from 0.001 to 4.0;
- a linearly increasing inertia weight ranging from 0.001 to 0.7; and
- c_1 and c_2 set to linearly increase from 0.5 to 2.0

Table 5.5 displays results for the similarity method while Table 5.6 displays results for the matches method with the *#gapsAllowed* parameter set to 3 and 8. All variations

that outperformed the reference configuration in terms of the *Mean* fitness are indicated in bold. Results are now discussed.

SymbolScore with Similarity

Table 5.5 provides results of all single characteristic variations (one on each row) and the reference configuration (last row of the table).

Table 5.5: Results for B-PSO optimizing the S8 alignment based on the similarity method with 3 and 8 gaps

<i>Variations</i>	#gapsAllowed = 3			#gapsAllowed = 8		
	<i>Best</i>	<i>Mean</i>	<i>Ci</i> [<i>lo</i> , <i>hi</i>]	<i>Best</i>	<i>Mean</i>	<i>Ci</i> [<i>lo</i> , <i>hi</i>]
Von Neumann (async)	35.0	27.60	[26.59276,28.60725]	29.0	17.93333	[16.43645,19.43022]
GBest (sync)	40.0	27.66667	[26.22688,29.10646]	30.0	16.56667	[15.06764,18.06569]
Swarm of 50 particles	38.0	27.40	[26.37591,28.42409]	27.0	17.90	[16.75466,19.04534]
$\mathbf{v}_i(0) \sim U(\mathbf{v}_{min}, \mathbf{v}_{max})$	37.0	26.93333	[25.80840,28.05826]	23.0	17.56667	[16.41265,18.72069]
Explorer	37.0	29.53333	[28.44788,30.61879]	28.0	20.46667	[19.10191,21.83142]
reference configuration	35.0	27.06667	[26.11025,28.02309]	27.0	17.73333	[16.46541,19.00126]

The following observations in comparison with the reference configuration can be made:

- Topologies: All three topologies were statistically identical with both 3 and 8 gaps allowed. The GBest topology variation obtained the best *Mean* and *Best* fitnesses with *#gapsAllowed* = 3. Also, the Von Neumann topology variation achieved better *Mean* fitnesses than the reference topology (LBest) with both 3 and 8 gaps allowed. All tested topologies experienced a large drop in performance when the number of gaps increased from 3 to 8.
- Number of particles (swarm size): Doubling the size of the reference swarm from 25 particles to 50 particles resulted in an increase in the *Mean* performance, although not a significant one. The gain in performance was too small compared to the expense of a longer execution time. Furthermore, the confidence intervals indicated that both swarm sizes had statistically equal performance.
- Velocity initialization: The high level of performance of the random velocity initialization observed in S-PSO was not observed with the B-PSO. This is explained

by the fact that random large velocities increased the probability of switching bits, which made the B-PSO search erratic. In fact, this variation had the worst *Mean* fitnesses overall. Also, all scores from this variation were approximately equal to the reference velocity initialization, where initial velocities were set to zero.

- Explorer configuration: The explorer configuration yielded a statistically superior performance than the reference configuration. Also, it can be noted that this configuration achieved the highest *Mean* fitnesses of all variations. Linearly increasing values of V_{max} , ω , c_1 and c_2 resulted in a clear advantage over fixed values (reference configuration). The superiority of this configuration can be attributed to its initial small V_{max} and inertia weight, which increased the exploration ability. A larger number of iterations also allowed the B-PSO to optimize solutions more efficiently.

Figure 5.16 visualizes the best S8 alignment produced by the similarity method.

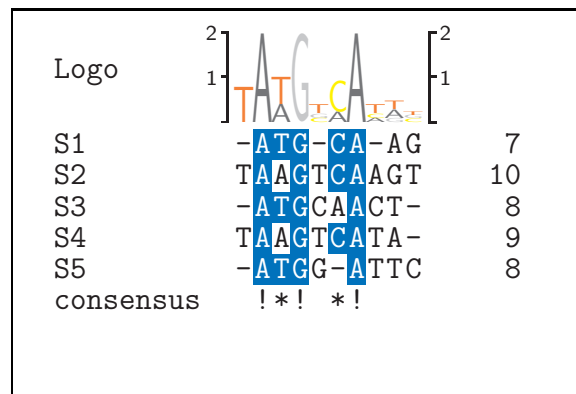


Figure 5.16: Visualization of the best S8 alignment for the similarity method

This alignment exhibited three fully matched columns as well as two highly aligned columns. A score of 40.0 (equivalent to a matches score of 92.4) was obtained. This score was the *Best* fitness from the GBest topology variation.

***SymbolScore* with Matches**

This new set of experiments differed from the previous set by the *SymbolScore* method used, which, in this case, is the matches method. The same variations as used in the

previous experiment were investigated for the matches method. Table 5.6 provides results of all single characteristic variations (one on each row) and the reference configuration (last row of the table).

Table 5.6: Results for B-PSO optimizing the S8 alignment based on the matches method with 3 and 8 gaps

<i>Variations</i>	#gapsAllowed = 3			#gapsAllowed = 8		
	<i>Best</i>	<i>Mean</i>	<i>Ci[lo, hi]</i>	<i>Best</i>	<i>Mean</i>	<i>Ci[lo, hi]</i>
Von Neumann (async)	81.20	66.55333	[64.49950,68.60715]	75.0	58.67332	[56.23402,61.11263]
GBest (sync)	77.0	65.53334	[63.67043,67.39625]	64.40	55.44667	[53.71401,57.17933]
Swarm of 50 particles	82.80	66.54668	[64.59789,68.49546]	72.40	58.32666	[56.34916,60.30416]
$\mathbf{v}_i(0) \sim U(\mathbf{v}_{min}, \mathbf{v}_{max})$	77.80	65.13999	[63.25700,67.02298]	93.60	59.62667	[56.72687,62.52646]
Explorer	83.60	68.52666	[66.63566,70.41765]	80.80	61.11333	[58.99666,63.23001]
reference configuration	82.80	65.66667	[63.66186,67.67149]	75.20	57.77333	[55.15395,60.39271]

The following observations in comparison with the reference configuration can be made:

- Topologies: The Von Neumann topology variation again proved to be stable and efficient with its high *Mean* fitnesses. Analogous to the observations made with the similarity method, all topology variations had statistically equal performance. Also, the drop in performance was persistent across all topologies when the increase of 5 gaps was applied.
- Number of particles (swarm size): The results from this variation confirmed the observations made for the similarity method. Doubling the reference swarm size produced slightly higher *Mean* fitnesses, but they remained nonetheless statistically equal to the reference configuration. Again, the increase in performance was minimal compared to the computational expense of executing a larger swarm.
- Velocity initialization: As seen with the similarity method, the random velocity initialization had the effect of slightly degrading performance. The only case where the reference velocity initialization was worse based on the *Mean* fitness occurred when #gapsAllowed increased to 8. Both velocity initialization schemes achieved statistically equal results.

- Explorer configuration: The results show that the explorer configuration had higher *Mean* fitnesses compared to the reference configuration and all other variations. However, there was no statistically significant difference in performance between the reference configuration and the explorer configuration for $\#gapsallowed = 3$ and 8.

Figure 5.17 depicts the best alignment obtained by the B-PSO for the matches method. This alignment exhibited three fully matched columns, as well as two highly aligned columns. A score of 93.6 (equivalent to a similarity score of 53.0) was obtained. This score was the *Best* fitness from the random initialization topology variation.

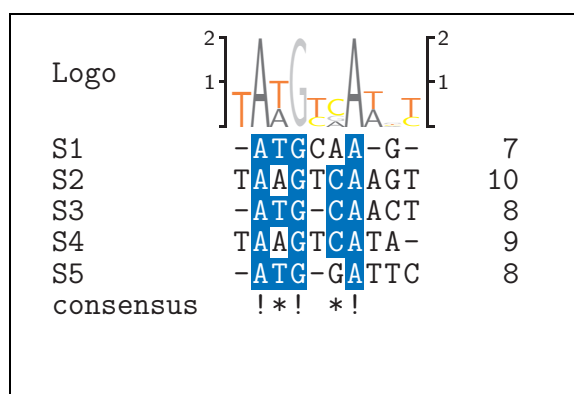


Figure 5.17: Visualization of the best S8 alignment for the matches method

Discussion

This section compares B-PSO scores with that of S-PSO, T-COFFEE and CLUSTAL X scores. The best similarity score achieved by the B-PSO was 40.0 (see *Best* column in Table 5.5), which is significantly better than the scores of T-COFFEE (24.0) and CLUSTAL X (4.0). However, the B-PSO score was below the similarity result of S-PSO (55.0). The best matches score from the B-PSO was 93.6 (see *Best* column in Table 5.6), which is significantly better than the scores of T-COFFEE (80.4) and CLUSTAL X (44.4). Also, the S-PSO remained better, with a matches score of 108.0.

The worse performance of B-PSO in comparison with S-PSO can be attributed to the increased dimensionality of solutions for the B-PSO. With reference to Section 4.2.2, data

set S8 has a maximal unaligned length of ten, requiring 4 bits. For $\#gapsAllowed = 3$, $m = 23$ (total of 23 gaps that can be inserted). Therefore the dimension of each particle for the B-PSO is $4m = 92$ compared to 23 for the S-PSO. Similarly for $\#gapsAllowed = 8$, $m = 48$ resulted in particle dimensions of 192. The increase in dimension for 8 gaps allowed compared to 3 gaps allowed also explains the significant drop in performance for more gaps. This dimensionality problem becomes more severe for longer sequences. It is for this reason that the B-PSO is not further investigated in subsequent experiments.

5.2.6 Conclusion

This section concludes on all experiments conducted on data set S8. Conclusions are drawn and pointers for further experimentation are established. The goal of the experiments on S8 was to test the S-PSO extensively to evaluate its viability to solve MSAs. A comprehensive selection of characteristics was used to determine the effectiveness of the S-PSO. The B-PSO was also tested but was discarded on account of its recurring bad performance.

The main outcome of this section was the identification of the best configuration for each scoring method:

- For the similarity scoring method, the best configuration used a LBest topology using asynchronous iterations with 3 neighbours; a swarm containing 25 particles; 1000 iterations for a complete run; no restriction on V_{max} ; a random velocity initialization; an inertia weight kept constant at 0.729844; and $c_1 = c_2 = 1.496180$ throughout.
- For the matches scoring method, the best configuration used a LBest topology using asynchronous iterations with 6 neighbours; a swarm containing 50 particles; 1000 iterations for a complete run; no restriction on V_{max} ; a zero velocity initialization; an inertia weight kept constant at 0.729844; and $c_1 = c_2 = 1.496180$ throughout.

These configurations will be used as the reference configurations in the next section where more sequences of differing complexity are used.

5.3 Scalability Analysis

The previous section explored the ability of S-PSO to align multiple sequences. This section extends the empirical analysis to more complex MSAs. Additionally, the section also presents a comparative analysis of the performance of different PSO algorithms.

This section is organized as follows. Section 5.3.1 describes the MSA data sets used. Configurations of the three newly participating PSO algorithms are given in Section 5.3.2. The experimental procedure is described in Section 5.3.3. Experimental results are presented and discussed in Section 5.3.4. Before the conclusion, a comparison between solutions from commonly used MSA programs and top PSOs is given in Section 5.3.5.

5.3.1 MSA Data Sets

As discussed in Section 5.1, the difficulty of an MSA problem is influenced by the following factors:

1. The relatedness and nature of the sequences to be aligned.
2. The length and number of sequences to be aligned.
3. The choice of the OF.

Based on these factors, a suite of MSA problems of varying complexity is defined in Table 5.7.

Table 5.7: Properties of the seven new data sets (S1 through S7)

<i>ID</i>	<i>Type</i>	<i>Name</i>	<i>k</i>	<i>Length_{mean}(min, max)</i>	<i>Similarity %</i>	<i>#gaps</i>	<i>m</i>
S1	DNA	Hepatitis C virus	10	212 (211, 212)	92.2 (very high)	3	31
S2	DNA	CGM13 gene	5	1,780 (1,775, 1,782)	63.0 (medium)	132	668
S3	DNA	Histone H3	21	122 (122, 122)	95.6 (very high)	1	21
S4	DNA	Partial HIV	8	1,437 (1,356, 1,485)	44.4 (low)	165	1,701
S5	DNA	Partial HIV	8	1,680 (1,680, 1,680)	95.4 (very high)	15	120
S6	mRNA	Pithecia	6	1,456 (1,430, 1,463)	77.7 (high)	65	427
S7	rRNA	16S Microcystis	8	457 (457, 457)	99.3 (very high)	1	8

These MSA problems are used in the rest of this chapter to further investigate the behaviour of PSOs solving more difficult MSAs. Criteria for the selection of the seven

data sets are divided into four distinct factors. Indeed, the new data sets differ with respect to their (1) number of sequences within the set in column four of Table 5.7, (2) sequence lengths in column five, (3) global similarity among the sequences in the set in column six, and (4) number of gaps allowed for alignment in column seven. The last column lists the calculated problem dimensionality for each set (refer to Section 4.2 for full details on how this is calculated). Complementary details such as accession IDs and data set sources are listed in Appendix A.

The four aforementioned factors have been chosen carefully to provide a diverse range of problems. The minimum average length was 122, for S3, while the maximum length was 1780, for S2. With regards to percentage similarity, the sets exhibited low, medium, high and very high similarities. Sets also varied with respect to the number of sequences contained. The smallest set contained 5 sequences for S2 while the largest set contained 21 sequences for S3. The number of gaps allowed also varied, but did not contribute as a true independent factor because it was derived from two other factors, as explained next.

The method of allocating the number of gaps allowed for the alignment process resulted from observations from the experiments on the S8 sequences. From these experiments it was observed that the S-PSO loses its accuracy when dealing with more gaps. In order to reduce the number of gaps allowed, a relation between number of gaps and sequence similarity is considered. For sequences with a low percentage similarity, more gaps would be required to produce an acceptable alignment. Conversely, in the case where the global similarity of the set is high, fewer gaps would be required. As already mentioned, Chellapilla and Fogel [22] proposed that the maximum number of gaps should be equal to 20% of the longest sequence length in the set of sequences. This is based on the observation that valid alignment rarely contained a gaps/characters ratio of more than 20%. Therefore, an initial rather large number of gaps that may eventually decrease over time makes sense to begin aligning with. Alternative approaches to calculate the number of gaps allowed include:

- Using the number of gaps published in literature. This approach is not used because one of the objectives of this work is to compare PSO results against two MSA programs found in the research literature: CLUSTAL X and T-COFFEE.

- Randomly selecting the number of gaps allowed. This is not used for two reasons (observed from preliminary trial tests): The probability of getting an optimal number of gaps allowed is too small to ensure (1) an accurate resulting alignment and (2) no unnecessary extra computational cost.
- Calculating the number of gaps as a function of similarity percentage and the maximum sequence length found in the set. For the purpose of this study, this thesis proposes

$$\#gapsAllowed = \varpi\% (20\% (\max\{|\mathcal{S}_i|\})), \quad \forall \mathcal{S}_i \in \mathcal{T}$$

where $\varpi = 100$ —similarity percentage of the set.

The latter option provides a more accurate estimation of the initial number of gaps allowed. Moreover, this option yielded a minimum acceptable threshold which resulted, most of the time, in a considerable decrease in initial gaps (instead of a generic 20%). The stochastic aspect of aligning sequences with PSO should be kept as minimal as possible, which can be achieved if the third approach is used instead of the second approach. At equal computational cost, the third approach has the advantage of being deterministic while the second method is not (random).

5.3.2 PSO Algorithms and Configurations

This preliminary section serves to introduce the experimental results. This step provides information concerning the key players in the next set of experiments. Three kinds of PSOs were evaluated, namely the standard PSO (S-PSO), the mutating PSO (M-PSO), and the cooperative split PSO (CPSO- S_K).

Standard PSO Configuration

Characteristics for the S-PSO and MSA were kept identical throughout all tests in order to maintain comparison consistency. The objective of this particular set of experiments was to identify possible limitations encountered by the S-PSO when optimizing harder MSAs.

Most importantly, the set of experiments conducted with the S-PSO made use of the ‘optimized’ configurations of S-PSOs for the similarity and matches methods. On the one hand, the reference S-PSO configuration associated with the optimization of the similarity method OF was as follows: a LBest topology using asynchronous iterations with 3 neighbours; a swarm containing 25 particles; 1000 iterations for a complete run; no restriction on V_{max} ; a random velocity initialization; an inertia weight kept constant at 0.729844; and $c_1 = c_2 = 1.496180$ throughout. On the other hand, the reference S-PSO configuration associated with the optimization of the matches method OF was as follows: a LBest topology using asynchronous iterations with 6 neighbours; a swarm containing 50 particles; 1000 iterations for a complete run; no restriction on V_{max} ; a zero velocity initialization; an inertia weight kept constant at 0.729844; and $c_1 = c_2 = 1.496180$ throughout.

Mutating PSO Configuration

The M-PSO was described in Section 3.4.4.

The main objective and motivation for these experiments are to test the increased diversity ability of the M-PSO [50] and to discover any potential gain in performance, such as accuracy or speed. Results obtained with the M-PSO were systematically subjected to direct comparison with S-PSO results to identify advantages and disadvantages.

Two M-PSOs, one used for the similarity method and another one used for the matches method, used the reference configurations for S-PSO as an initial configuration. The mutation rate (refer to Section 3.3.2 for details) was linearly decreased from 0.9 to 0.1 over time. Lastly, it is important to note that the M-PSO employed for this set of experiments was modified from the original version. A newly mutated particle position replaced the non-mutated position if and only if a better fitness was returned. This conditional replacement has the advantage of discarding worse solutions while retaining the better ones.

Cooperative Split PSO- S_K Configuration

The CPSO- S_K , which was described in Section 3.4.4, was identified as a valuable candidate algorithm to optimize high complexity MSAs.

To achieve the cooperative mechanism on MSAs, the problem dimensionality, m , was perfectly split into an equal number of chunks (refer to Section 3.4.4 for details on how this is calculated). Furthermore, sets exhibiting both high similarity and low dimensionality were excluded from the CPSO- S_K experiments (S1, S3 and S7) because S-PSO already found the global optimum for S1, S3 and S7. Only harder sets were used to investigate whether CPSO- S_K can improve on S-PSO's performance. Experiments were thus conducted on the S2, S4, S5 and S6 data sets. Table 5.8 provides the respective split factor, K , pertaining to the selected data sets. Also shown in Table 5.8 is the respective number of gaps allowed per sub-swarm.

Table 5.8: Split factors calculated for the four selected data sets used in the CPSO- S_K experiments

ID	m	K	$\#gapsAllowed \text{ per sub} - swarm$
S2	678	6	113
S4	1,701	9	189
S5	120	2	60
S6	427	7	61

Two CPSO- S_K s, one for the similarity method and one for the matches method, using the reference configurations for S-PSO as an initial configuration, were used.

5.3.3 Experimental Procedure

All subsequent experiments were conducted using the default set of MSA characteristics as outlined in Section 5.2.2, with regard to both similarity and matches methods. PSO characteristics and configurations are as defined in the previous subsections.

Tables 5.9 and 5.10 facilitate the comparison between the three PSOs used. The ' PSO ' column indicates which optimizer was applied. Furthermore, numerical figures that relate to the number of matches found (' $\#Mat$ '), the number of gaps inserted (' $\#Gaps$ ') and the number of fully aligned columns (' $\#FullC$ ') originated from the best alignment solution out of 30 simulations. In all cases, the best alignment solution was elected according to the highest score produced (' $Best$ '). The two columns remaining – the ' $Mean$ ' and ' $Ci[lo, hi]$ ' – represent the typical statistical values computed over

30 simulation runs. The alignment method is specified with either an ‘S’ or ‘M’ letter, which stand for similarity and matches method respectively.

Finally, note that no more alignment visualizations will be provided due to the large size of alignments, although bar graphs will be presented for purposes of visual analysis.

5.3.4 Experimental Results

This section presents the results acquired by aligning the new data sets with the three versions of PSO discussed earlier – the S-PSO, the M-PSO and the CPSO- S_K . An analysis of the results, which entails inter-PSO comparisons and alignment methods, is also included. Note that observations and analyses will be provided one data set at a time, from S1 to S7.

Before discussing the results, attention must be given to the comparison of MSA scores. It is reasonable to compare solutions of ‘low difficulty’ MSAs using only the number of fully aligned columns as an indicator, but this was not entirely satisfactory for harder MSAs. Certainly, MSAs with a high percentage of similarity were more prone to achieve fully aligned columns. However, lower similarity MSAs which achieved a low number of fully aligned columns did *not* necessarily indicate that they were badly aligned. They may have contained many matches, despite not having entirely matched columns. For this reason, it is more sensible to look at the number of matches – a finer-grained indicator of quality – when comparing more difficult MSAs. It is evident that all three scores (number of matches, gaps and fully aligned columns) were needed in order to evaluate and determine the worth of the alignment. Ultimately, all three indicators needed to be evaluated to judge whether an MSA was better or worse. Nevertheless, the emphasis shifted from the number of fully aligned columns indicator to the number of matches and gaps indicators when the evaluation changed from low difficulty to medium/high difficulty MSAs.

Tables 5.9 and 5.10 summarize all the results obtained from aligning the seven ‘harder’ sets by the three PSOs, with the similarity method and matches method respectively. Four 3-D bar graphs are provided to visually compare the performance.

Table 5.9: Performance results from S-PSO, M-PSO and CPSO- S_K optimizing the similarity method OF on data sets S1 through S7

<i>ID</i>	<i>PSO</i>	<i>#Mat</i>	<i>#Gaps</i>	<i>#FullC</i>	<i>Best</i>	<i>Mean</i>	<i>Ci[lo, hi]</i>
S1	S-PSO	9,392	1	198	9,454.18164	9,454.18457	[9,454.18352, 9,454.18562]
S1	M-PSO	9,392	1	198	9,454.18164	9,454.18457	[9,454.18352, 9,454.18562]
S2	S-PSO	13,443	13	782	14,882.21582	14,545.82324	[14,469.62164, 14,622.02485]
S2	M-PSO	14,452	28	1,073	15,748.76953	14,324.12988	[14,169.14688, 14,479.11289]
S2	CPSO- S_6	15,380	68	1,251	16,189.75781	11,088.62305	[9,241.24541, 12,936.00068]
S3	S-PSO	24,503	0	109	25,043.65430	25,043.64648	[25,043.64369, 25,043.64928]
S3	M-PSO	24,503	0	109	25,043.65430	25,043.64648	[25,043.64369, 25,043.64928]
S4	S-PSO	17,821	381	242	22,778.13672	22,667.14258	[22,641.18671, 22,693.09844]
S4	M-PSO	16,027	381	137	20,901.01367	19,569.87305	[19,372.85226, 19,766.89384]
S4	CPSO- S_9	21,344	461	302	25,577.53125	21,329.78516	[20,480.33813, 22,179.23218]
S5	S-PSO	44,910	0	1,440	45,949.80469	45,949.79688	[45,949.79408, 45,949.79967]
S5	M-PSO	44,910	0	1,440	45,949.80469	45,949.79688	[45,949.79408, 45,949.79967]
S5	CPSO- S_2	44,910	0	1,440	45,949.80469	44,779.08984	[44,318.67753, 45,239.50216]
S6	S-PSO	17,920	43	887	19,415.15039	19,199.84570	[19,176.03514, 19,223.65627]
S6	M-PSO	17,699	43	866	19,280.79688	19,156.85156	[19,145.23530, 19,168.46782]
S6	CPSO- S_7	17,934	109	843	19,077.05078	14,168.64941	[12,653.37313, 15,683.92570]
S7	S-PSO	12,713	0	449	12,754.33789	12,754.34082	[12,754.33977, 12,754.34187]
S7	M-PSO	12,713	0	449	12,754.33789	12,754.34082	[12,754.33977, 12,754.34187]

For the first data set, S1, the S-PSO and the M-PSO achieved statistically equivalent scores with the similarity method and the matches method. The optimal alignment was found, having 198 fully aligned columns, only one gap and a total of 9392 matches. The fact that the optimal alignment was consistently obtained, shows that having 10 sequences of about 212 characters did not affect the convergence onto an optimum as long as the dimensionality was kept low ($m=31$). This set was, therefore, labelled as ‘low difficulty’, in consideration of the fact that it had a high overall similarity and a low dimensionality.

The second data set, S2, contained half the number of sequences than S1 does and was roughly eight times longer. The characteristics of S2 were a medium similarity (63%) and a dimensionality of 668, which represented more than 20 times the dimensionality of S1. The ‘staircase’ pattern that emerged, as evidenced in Figure 5.18, shows that all three PSOs achieved improved performance with respect to the number of fully aligned columns. The S-PSO achieved the worst performance with 13,443 matches and 782 fully aligned columns when optimizing the OF that used the similarity method. However, the S-PSO and the M-PSO had statistically equal similarity scores.

Table 5.10: Performance results from S-PSO, M-PSO and CPSO- S_K optimizing the matches method OF on data sets S1 through S7

<i>ID</i>	<i>PSO</i>	<i>#Mat</i>	<i>#Gaps</i>	<i>#FullC</i>	<i>Best</i>	<i>Mean</i>	<i>Ci[lo, hi]</i>
S1	S-PSO	9,392	1	198	51,157.19922	50,828.14844	[50,714.08840, 50,942.20848]
S1	M-PSO	9,392	1	198	51,157.19922	50,958.87891	[50,881.35093, 51,036.40688]
S2	S-PSO	13,463	108	929	36,254.0	28,083.51953	[26,686.45668, 29,480.58239]
S2	M-PSO	14,640	38	1,120	40,802.0	31,307.66211	[30,334.65203, 32,280.67219]
S2	CPSO- S_6	14,296	368	1,086	38,842.39844	32,326.90625	[30,333.84675, 34,319.96575]
S3	S-PSO	24,503	0	109	263,570.09375	263,466.06250	[263,358.74562, 263,573.37938]
S3	M-PSO	24,503	0	109	263,570.09375	263,453.28125	[263,326.14591, 263,580.41659]
S4	S-PSO	17,180	613	83	49,217.0	40,677.69141	[39,415.82245, 41,939.56037]
S4	M-PSO	18,848	453	92	57,188.250	39,234.89063	[37,553.18986, 40,916.59139]
S4	CPSO- S_9	19,021	981	206	58,503.3750	43,390.49219	[39,506.64287, 47,274.34150]
S5	S-PSO	43,481	64	1,307	187,769.3750	175,396.64063	[172,696.20020, 178,097.08105]
S5	M-PSO	44,910	0	1,440	197,274.0	197,274.0	[197,274.0, 197,274.0]
S5	CPSO- S_2	44,663	88	1,420	195,467.3750	191,293.18750	[190,196.93100, 192,389.44400]
S6	S-PSO	16,131	151	691	49,759.16797	29,457.95117	[27,929.11248, 30,986.78987]
S6	M-PSO	17,409	37	853	55,641.16797	41,596.00391	[40,010.52637, 43,181.48145]
S6	CPSO- S_7	17,938	277	845	56,975.33203	49,386.95703	[47,092.48738, 51,681.42668]
S7	S-PSO	12,713	0	449	57,038.6250	57,038.6250	[57,038.6250, 57,038.6250]
S7	M-PSO	12,713	0	449	57,038.6250	57,038.6250	[57,038.6250, 57,038.6250]

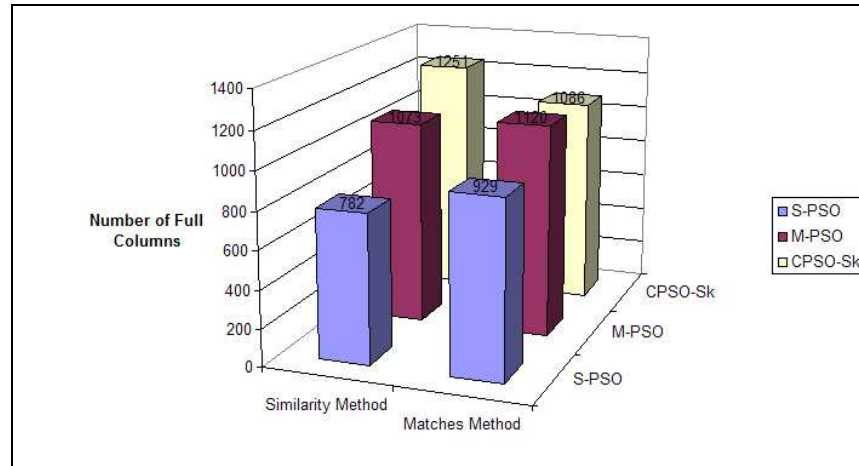


Figure 5.18: Number of full columns aligned in S2 with three PSOs using both the similarity and the matches methods

As illustrated in Figure 5.18, the M-PSO optimizing the OF that used the matches method on S2 achieved the highest *Best* score with 1120 fully aligned columns and 14640 matches. The CPSO- S_6 , using the similarity method, achieved the highest *Best* score

and managed a total of 15380 matches as well as the greatest number of fully aligned columns (1251). The CPSO- S_6 that optimized the OF using the similarity method proved to be instable on account of its large confidence interval. For this data set, the similarity method was consistently outperformed by the matches method except in the case of the CPSO- S_6 . Lastly, the significantly larger number of gaps observed in solutions that used the matches method was a recurrent phenomenon.

Data set S3 produced very similar results to data set S1. S3 had roughly twice the number of sequences than S1, as well as half the sequence length of S1. Most importantly, S3 similarly had a very high percentage of similarity (95.6%) and a low dimensionality ($m=21$), which also classified S3 as ‘low difficulty’. The S-PSO and M-PSO results showed that optimal alignment solutions were found consistently. Both PSOs had statistically identical scores and converged on the best solutions using both methods. In other words, it appeared that doubling the number of sequences did not affect the accuracy of the PSO, bearing in mind that the set offered a high overall similarity and a low dimensionality. In fact, secondary tests demonstrated that S3 converged perfectly, even with its length twice that of S1.

The fourth data set, S4, was, according to observations so far, the hardest MSA to optimize. S4 has eight relatively long sequences (1437). S4 was opposite to S1 and S3 in terms of overall similarity, which was the lowest with 44.4%, and dimensionality, which was the highest with $m=1701$. CPSO- S_K again dominated in the S4 results. Figure 5.19 depicts results again suggesting the superiority of the similarity method over the matches method. Indeed, the matches method frequently inserted gaps that were unnecessary since no increase in the number of matches was gained for the undesirable increase in gaps. With the similarity method, the S-PSO was statistically superior to the CPSO- S_9 and the M-PSO. However, with 21344 matches and 302 fully aligned columns, the CPSO- S_9 managed to find the best S4 alignment. As for the matches method optimization, all PSOs were statistically identical but, again, the CPSO- S_9 achieved the highest number of full columns and matches. Lastly, it appeared that the CPSO- S_K finds excellent solutions but in a very instable manner.

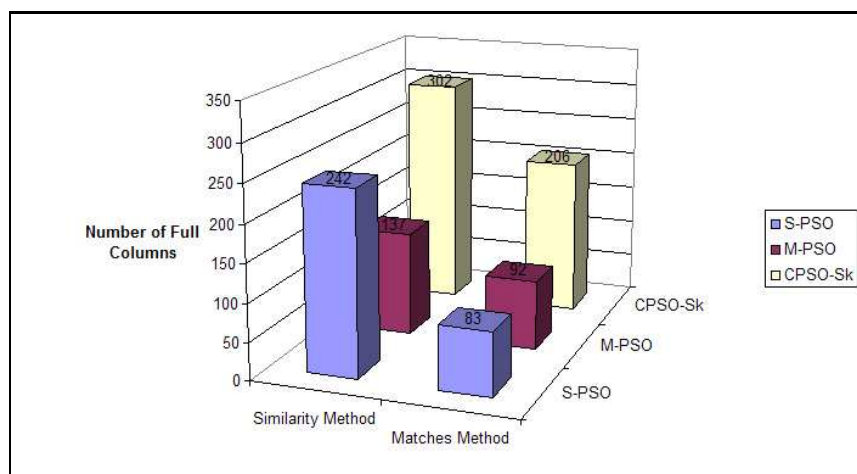


Figure 5.19: Number of full columns aligned in S4 with three PSOs using both the similarity and the matches methods

S5 and S4 both had eight sequences as well as roughly the same sequence length. S5, however, critically differed from S4 through its greater percentage of similarity (95.4%) and a much lower dimensionality ($m=120$). S5, therefore, also qualified as a ‘low difficulty’ MSA. As illustrated in Figure 5.20, this set was well optimized by the M-PSO, the only PSO which managed to produce top solutions with both the similarity and matches methods: 1440 fully aligned columns. The direct consequence of the lower dimensionality and greater similarity observed in S5 was that all three PSOs using the similarity method performed extremely well and reached convergence by finding the optimal alignment. S5 therefore confirmed that dimensionality and overall set similarity have a critical influence on accuracy. Note again how the matches method was worse, in that it produced more gaps and fewer matches.

The sixth data set, S6, provided mixed results that failed to accurately indicate which alignment method was better overall. Notice that the CPSO- S_7 managed to generate the highest number of matches of all S6 results, which again proved its superiority – recall that the number of matches prevailed in evaluations of difficult MSAs. However, the S-PSO managed to achieve the highest number of fully aligned columns (887) and a remarkably low number of gaps with the similarity method, with the M-PSO closely following. Furthermore, the S-PSO achieved statistically the worst performance with the matches method. The stability of the M-PSO surfaced strongly in this set, while the

CPSO- S_7 again showed instability.

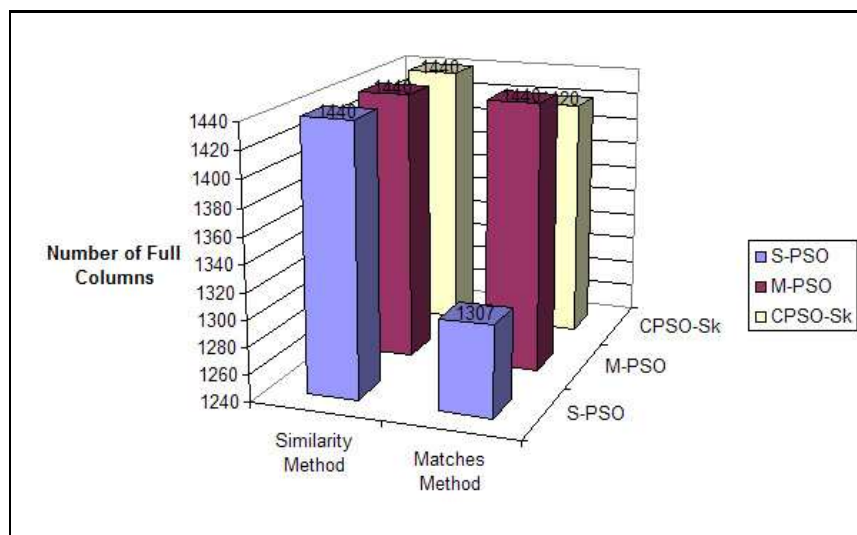


Figure 5.20: Number of full columns aligned in S5 with three PSOs using both the similarity and the matches methods

As depicted in Figure 5.21, for the first time using the similarity method and considering the number of fully aligned columns, the CPSO- S_7 ranked third after the M-PSO, which also produced the lowest number of gaps using the matches method (37). There is no straightforward way to determine which alignment was better. Instead, it should be acknowledged that all three PSOs produced equally good solutions (CPSO- S_7 had the greatest number of matches, M-PSO had the lowest number of gaps and S-PSO produced the highest number of fully aligned columns). Data set S6 was special in the sense that it was the only set to share a high level of similarity (77.7%) with a significantly large dimensionality ($m=427$). With such characteristics and six long sequences, S6 positioned itself directly after data set S2 in terms of MSA difficulty. Remember that the CPSO- S_7 largely dominated data set S2, which suggested that the CPSO- S_7 is more inclined to prevail in higher complexity sets where other PSOs might fail.

The last data set tested was S7. This set also provided an opportunity to analyze scalability in terms of length of sequences. S7 contained roughly the same number of sequences as S1 and has roughly double the length of S1, while keeping a very high percentage of similarity and a low dimensionality.

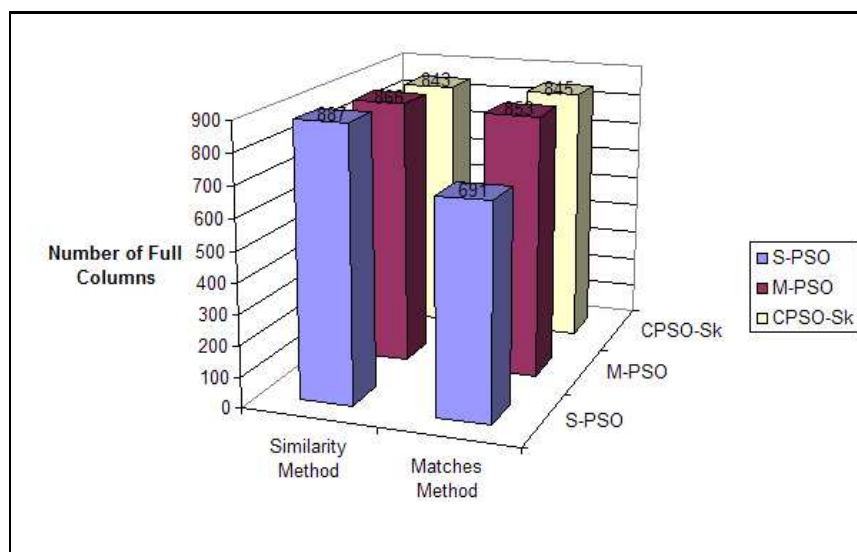


Figure 5.21: Number of full columns aligned in S6 with three PSOs using both the similarity and the matches methods

Both S-PSO and M-PSO, optimizing both alignment methods, successfully converged onto optimal solutions (449 fully aligned columns). Observations here confirmed that ‘low difficulty’ MSAs were affected neither by the number of sequences presented in the set, nor by the sequence length.

5.3.5 Comparison with Other MSA Programs

Top alignments produced with PSO algorithms in this research are now compared against widely used MSA programs. CLUSTAL X (version 1.83) and T-COFFEE (version 4.99) were used for this purpose (refer to Sections 2.6.2 and 2.6.4 for background).

Score Comparison

The main criterion subjected to comparison was alignment accuracy, using a scoring scheme identical to the one applied in all the PSO experiments. All alignment scores from T-COFFEE and CLUSTAL X are summarized in Table 5.11. Additionally, a graph in Figure 5.22 provides a visual comparison of the alignment scores obtained from the hardest sets (S2, S4 and S6) between the best PSO, T-COFFEE and CLUSTAL X.

Table 5.11: Comparison of results obtained from sets S1 through S7 with T-COFFEE and CLUSTAL X

<i>ID</i>	<i>MSA technique</i>	<i>#Mat</i>	<i>#Gaps</i>	<i>#FullC</i>	<i>Sim_score</i>	<i>Match_score</i>
S1	S-PSO, M-PSO	9,392	1	198	9,454.18180	51,157.19999
S1	T-COFFEE	9,392	1	198	9,454.18180	51,157.19999
S1	CLUSTAL X	9,392	1	198	9,454.18180	51,157.19999
S2	CPSO- S_6	15,380	68	1,251	16,189.75781	—
S2	T-COFFEE	17,058	28	1,611	17,305.91219	50,306.39999
S2	CLUSTAL X	17,055	28	1,613	17,305.30551	50,307.79999
S3	S-PSO, M-PSO	24,503	0	109	25,043.65468	263,570.09523
S3	T-COFFEE	24,503	0	109	25,043.65468	263,570.09523
S3	CLUSTAL X	24,503	0	109	25,043.65468	263,570.09523
S4	CPSO- S_9	21,344	461	302	25,577.53125	—
S4	T-COFFEE	34,462	1,037	914	32,521.76227	142,679.0
S4	CLUSTAL X	34,274	629	899	34,203.86675	142,251.75
S5	ALL PSOs	44,910	0	1,440	45,949.80498	197,274.0
S5	T-COFFEE	44,910	0	1,440	45,949.80498	197,274.0
S5	CLUSTAL X	44,910	0	1,440	45,949.80498	197,274.0
S6	S-PSO	17,920	43	887	19,415.15039	—
S6	T-COFFEE	19,832	133	1,131	20,256.87428	66,400.33333
S6	CLUSTAL X	19,777	67	1,122	20,440.58684	66,204.16666
S7	S-PSO, M-PSO	12,713	0	449	12,754.33785	57,038.625
S7	T-COFFEE	12,713	0	449	12,754.33785	57,038.625
S7	CLUSTAL X	12,713	0	449	12,754.33785	57,038.625

When dealing with ‘low difficulty’ MSAs (i.e. S1, S3, S5 and S7), both T-COFFEE and CLUSTAL X managed to match all the optimal alignment solutions found by the PSOs.

However, a ‘high difficulty’ MSA such as S2 was aligned almost identically by both CLUSTAL X and T-COFFEE, since they managed to discover more than 17000 matches and roughly 1600 fully aligned columns. In comparison, the top PSO solution achieved by the CPSO- S_6 with the similarity method managed to discover 15380 matches and 1251 fully aligned columns. Additionally, the CPSO- S_6 solution managed to reduce the total number of gaps to 68, which was close to the 28 gaps produced by the CLUSTAL X and T-COFFEE solutions.

The hardest MSA of all data sets was S4. T-COFFEE dominated this alignment by having found 34462 matches, but produced the largest number of gaps (1037). CLUSTAL X was behind T-COFFEE in terms of matches, but was better with regards to the number of gaps, namely only 629. Once again, the top alignment solution amongst all proposed PSOs was the CPSO- S_9 with 21344 discovered matches and 461 gaps. The comparison between the number of fully aligned columns was quite contrastive, considering that the CPSO- S_9 ‘only’ found 302 of them, while T-COFFEE discovered a little over 900. The CPSO- S_9 was worse with regards to the number of matches too, but had the lowest number of gaps.

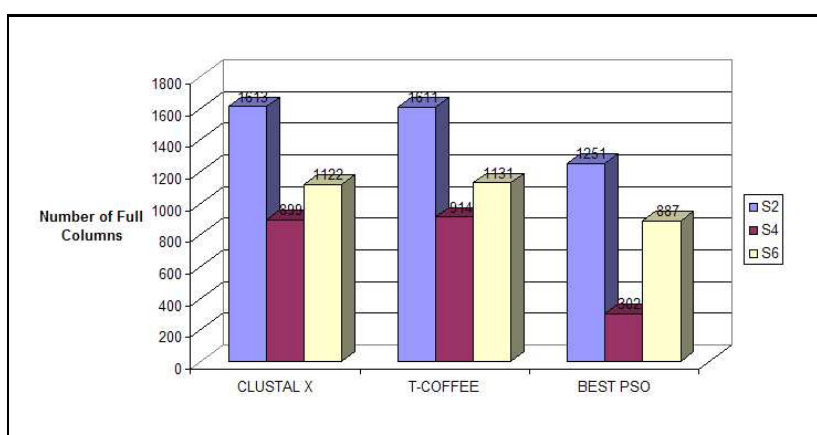


Figure 5.22: Best score comparison between PSO, T-COFFEE and CLUSTAL X on the three hardest sets (S2, S4 and S6)

Lastly, data set S6, classified as ‘medium difficulty’, was best aligned by T-COFFEE with 19832 matches found and 1131 fully aligned columns. CLUSTAL X closely followed, by having found fewer matches and fewer fully aligned columns, but minimized the number of gaps more successfully. The S-PSO produced a good sub-optimal S6 alignment, with the best number of fully aligned columns out of the PSOs (887). However, the CPSO- S_7 also managed to find a satisfactory alignment with the top number of matches (17938) amongst PSOs, although with more gaps as well.

Discussion

No MSA program can rightly claim its superiority in all aspects of solving MSAs. It is, therefore, about appreciating which qualities one MSA program has to offer. Indeed, MSA programs excel in specialized aspects of MSAs, such as speed or biological accuracy. It is in the user's best interest to make use of an MSA tool that would provide, according to each user's specific needs, the most adequate service.

Commonly used approaches, such as CLUSTAL X, are assumed to be very successful when an MSA has a small to medium number of sequences, short to medium length and a high to medium overall similarity. Due to its algorithmic nature and the time complexity of $O(k^4 + n^2)$, CLUSTAL X grows exponentially, which, as a consequence, makes it a slow algorithm on large alignments. A similar phenomenon occurs when using exact algorithms such as dynamic programming [80]. It is a fact that most biologists use some degree of post-processing to refine the alignment that is produced by CLUSTAL X [22]. The reason for this lies in the limitations that CLUSTAL X has in the kind of objective function it can optimize.

T-COFFEE is a GA-based algorithm that proved to be very robust, but very slow. The order of its time complexity is $O(k^3n^2)$, which explains the lack of speed. In compensation, however, it allows for better accuracy. The convergence of T-COFFEE is slow, which is once again not a disadvantage, but rather an advantage for the user who expects solutions of high accuracy for difficult problems.

The basic PSO algorithm has the crucial advantages of low computational complexity, $O(Tsm)$, and low memory space requirements, which makes the PSO a fast algorithm to execute. The size and length of the sequences were the criteria that proportionally affected the OF in terms of time complexity. For a total computational complexity of PSO for MSA, the cost of the specific OF used must be included. For instance, the OF that uses the similarity method requires $O(k^2\hat{n} + \hat{n}^2)$ to compute the SoP and aGP scores. A thorough study of the speed of PSO for MSA is listed as a recommendation for future work in Section 6.2.

5.3.6 Conclusion

This section finalizes the portion of this research dedicated to expanding the potential of PSOs to optimize harder MSAs. This last part of the experiments is understood as a scalability study. Firstly, results on seven ‘harder’ MSA data sets between the three proposed PSOs (the standard PSO, the mutating PSO and the cooperative split PSO) were compared and analyzed. Secondly, top PSO solutions were tested against solutions found by widely used MSA programs.

These last experiments on harder MSAs were rich in observations. One important observation was that the greater the similarity percentage, the less the similarity method scores differed among the three PSOs. This means that the similarity percentage played a central role in determining PSO capabilities. Another major observation was that the matches method had a tendency to find solutions involving many unnecessary gaps, compared to the similarity method. This is a parameter problem and, therefore, may be alleviated by optimizing the affine gap penalties.

The S-PSO demonstrated good performance by successfully aligning difficult MSAs, but lacked accuracy when the MSA problem dimensionality became too large. Indeed, the main phenomenon that emerged from experiments on the first data set, S8, was that S-PSO seemed to lose its accuracy when dealing with a higher number of gaps. The S-PSO converged quickly, which made it a competitive algorithm for easy problems. However, it performed poorly on a few hard problems because of local optima traps. The dimensionality of particles and the degree of difficulty associated with finding an optimum solution usually played a significant role in performance.

The increased diversity of the M-PSO did not always suffice to optimize a higher degree of alignment complexity but, compared to the S-PSO, showed a clear improvement. The M-PSO was superior in several cases, but failed to distinctively prove itself as a better algorithm. Unfortunately, the M-PSO suffered from similar disadvantages than the S-PSO.

The cooperative feature of the CPSO- S_K is designed to scale well with growing MSA complexity. The cooperative approach is flexible and can cater for more participating algorithms where necessary. Results produced with the CPSO- S_K were encouraging and overall largely better than the S-PSO and M-PSO. The CPSO- S_K successfully demonstrated that it represents a robust MSA solver and a very promising MSA alternative

solver.

5.4 Summary

This chapter examined and discussed the experimental procedure that has been conducted in this thesis. Section 5.1 positioned the objectives and scope of the experimental work to be accomplished. Section 5.2 presented the experimental work on S8, testing the viability of the S-PSO and B-PSO as an effective MSA solver. Section 5.3 analyzed the performance of PSO algorithms on a variety of MSA problems, differing in complexity.

Chapter 6

Conclusion and Future Work

“V’la autch’!”

- Andrzej Romanshkov a.k.a. W-Dad

This chapter summarizes the major findings derived from the work conducted in this thesis. Finally, a set of future directions worth investigating is suggested.

6.1 Conclusion

The main objective of this study was to verify the following hypothesis: *“Is particle swarm optimization a viable approach to solving the multiple sequence alignment problem, and, if so, to what degree of success?”* Considering the novelty of the approach, steps towards defining a proper experimental procedure were carefully designed.

The research work started in the Chapter 2 with a discussion of the background of multiple sequence alignment (MSA). MSAs are complex in many respects and, thus, need to be explored thoroughly. Important concepts and basic building blocks were defined in order to construct a solid base of knowledge. Focus was given to alignment manipulation strategies, evaluation methods and current alignment techniques.

Chapter 3 was mainly dedicated to particle swarm optimization (PSO). Genetic algorithms were covered to provide a computational intelligence perspective on the GA-based MSA tool, T-COFFEE. Definitions of nondeterministic polynomial complete (NP-complete) problems, which represent a critical characteristic of MSA problems, were

provided. Particle swarm optimization was thereafter explored in depth, with respect to the algorithm, topologies and parameters. Modifications to PSOs were also discussed, which identified the candidates to be tested on MSAs. The PSOs under evaluation were the standard PSO (S-PSO), the binary PSO (B-PSO), the mutating PSO (M-PSO) and the cooperative split PSO (CPSO- S_K).

The core of this work took the form of experiments. The experimental work that was conducted throughout the completion of this thesis is presented in Chapter 5. The experimental procedure was divided into two parts.

The first part extensively analyzed the S-SPO and B-PSO on a small, low complexity MSA. Several characteristics were tested to study their impact on the optimization process. Either of two aligning methods were employed as part of the objective function, namely the similarity method or the matches method. Success was achieved when both methods proved to optimally align a small MSA data set. However, many S-PSO variations failed to consistently demonstrate convergent behaviour. The B-PSO showed some promise, but did not convincingly represent a valuable MSA optimizer. The major discoveries can be summarized with the following statements. The first is that randomly initializing the S-PSO velocity yielded superior results, especially when using the similarity method. The other is that increasing the number of particles within the swarm usually improved solutions. The goal of this part of the experiments was achieved when a proper identification of the impact of each factor regarding MSA optimization was made. An important fact should not be overlooked: the S-PSO seemed to lose its accuracy when dealing with more gaps (higher complexity). From the S8 set of experiments, two ‘optimized’ versions of S-PSOs were derived. These two versions managed to outperform T-COFFEE and CLUSTAL X on the small data set. This means that the PSO approach excels at solving small alignments of a high similarity.

The second part investigated to what extent the S-PSO scales for harder MSA problems. Applying the empirical analysis done in the S8 experiments, two optimized S-PSOs were tested on higher complexity MSAs. Seven harder MSA data sets were used. Experiments that involved the M-PSO and CPSO- S_K were also conducted on the same data sets. With more complex MSAs, several aspects of the problem were modified: the problem space was considerably increased, the overall similarity percentage was lowered, and the number of gaps to insert became larger. Ways to reduce complexity in hard

MSAs were used, such as decreasing the number of gaps allowed in the alignment or breaking the alignment into chunks. The S-PSO and M-PSO showed some degree of success but were both significantly outperformed by the CPSO- S_K . The main conclusion is that the CPSO- S_K offers a flexible solution that creates a new range of possibilities. As the PSO approach needs to mature, several improvements remain that could increase its potential as a robust MSA solver. A comparative analysis with widely used MSA programs revealed that CPSO- S_K solutions could not improve on results of CLUSTAL X and T-COFFEE.

Conclusions about the advantages of using PSO as an MSA optimizer include the following:

- The PSO approach, unlike CLUSTAL X, relaxes heuristics limitation by allowing the use of any arbitrary objective function. Custom-tailored objective functions could then be oriented towards either accuracy or speed, to be more adequate for specific MSAs.
- PSOs are fast algorithms, low in memory requirement, and simple to implement.
- Extra computations such as pre-processing and post-processing, i.e. tree creation, sequence grouping, calculating distance between sequences or sequence selection, are not needed.
- No specific knowledge about species' origins or sequence structure is required. Again, the PSO is a stochastic approach that uses heuristics, so it does not guarantee that alignments are optimal. However, sub-optimal alignments are still satisfactory for many applications.
- For time constrained executions, the PSO is a valuable alternative as the PSO can be limited to a certain number of iterations or desired fitness for a sufficiently accurate or fit solution. Any sequence alignment algorithm must trade-off computation speed for alignment accuracy, especially when there exists a low similarity between the sequences that are to be aligned.
- For the same set of sequences, a method based on pairwise DP may require hours to align, while the PSO can produce results of slightly less or similar quality within a much smaller time period.

Lastly, an area where the PSO approach and particularly the CPSO- S_K revealed itself particularly useful – as the scalability analysis revealed – was in the alignment of long and large sets of high similarity sequences. In fact, the PSO accuracy was negligibly affected by the number and length of the sequences, as long as the similarity remained high. Furthermore, its accuracy combined with its efficiency in computing time formed a major strength; this is especially significant when aligning vast sets of similar sequences.

6.2 Future Work

Throughout this thesis, several ideas and avenues for future research have been identified. A brief overview is listed below.

Scalability

Understanding scalability factors ultimately helps to calibrate the optimization parameters. The second part of the experiments aimed to cover several aspects of PSO scalability. However, several other scalability facets could be investigated. One facet would involve aligning a ten sequences set by starting with only two sequences at first, then three, four ... until ten and observing the impact on performance. Another facet might entail comparing the performance of a set against the same set but with its number and length of sequences doubled, tripled, and so forth.

Alignment Refining

Studies refining alignment might attempt to produce alignments of better overall quality/accuracy. Several techniques could be applied that are independent of the PSO process. The first technique involves applying any improver listed in Section 2.6 after the PSO to refine a solution (post-processing). Another technique involves using solutions from an external MSA program such as CLUSTAL X to seed and initialize PSO particles. PSO solutions might also be fed into an external MSA program to be further improved. Indeed, PSO solutions could be improved before (pre-processing), during (each iteration with a local search mechanism such as DP) or after (post-processing/improver).

Multiple Solution PSOs

It would be worthwhile to investigate the performance and behaviour of PSOs that have the capability to optimize multiple solutions such as the multiple objective PSO (MOPSO) [26] or the niche PSO [16].

Particle Initialization

Future research might include an investigation into the impact of different particle initialization strategies. In effect, various ways of positioning the particles in the search space may influence convergence and performance on the MSA problem.

Speed Analysis

An in-depth study of the impact of the optimization of very large alignments (more sequences and/or longer sequences) versus computational time would be useful.

Cooperative Behaviour

The CPSO- S_K proved to be robust and efficient. More investigation into this promising approach needs to be conducted to expand its potential. For instance, future research may include studying the impact of different numbers of participant algorithms, and the effect of hybridizing the cooperative approach with different optimizers.

Proteins

This study only made use of DNA sequences. However, proteins also need to be aligned in many bioinformatics applications. An in-depth study was not conducted in this work, but early testing using the BALiBASE 2.01 benchmark [114] showed promising results, which need to be further investigated.

Bibliography

- [1] K. A. Abd-Elsalam. Bioinformatic tools and guideline for PCR primer design. *African Journal of Biotechnology*, 2(5):91–95, May 2003.
- [2] S. Abdeddaïm and B. Morgenstern. Speeding up the DIALIGN multiple alignment program by using the ‘greedy alignment of BIOlogical sequences LIBrary’ (GABIOS-LIB). In Olivier Gascuel and Marie-France Sagot, editors, *Journées Ouvertes Biologie, Informatique et Mathématiques*, volume 2066 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 2000.
- [3] S. F. Altschul, R. J. Carroll, and D. J. Lipman. Weights for data related by a tree. *Journal of Molecular Biology*, 207:647–653, 1989.
- [4] S. F. Altschul, W. Gish, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, October 1990.
- [5] S. F. Altschul and D. J. Lipman. Trees, stars and multiple biological sequence alignment. *Society for Industrial Applied Mathematics*, 49(1):197–209, February 1989.
- [6] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and psi-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25:3389–3402, 1997.
- [7] L. A. Anbarasu, P. Narayanasamy, and V. Sundararajan. Multiple sequence alignment using parallel genetic algorithms. In Bob McKay, Xin Yao, Charles S. Newton, Jong-Hwan Kim, and Takeshi Furuhashi, editors, *Simulated Evolution and Learn-*

- ing, volume 1585 of *Lecture Notes in Computer Science*, pages 130–137. Springer, 1998.
- [8] P. J. Angeline. Using selection to improve particle swarm optimization. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pages 84 – 89, July 1999.
- [9] K. Arms and P. S. Camp. *Biologie*, volume 1, chapter 5. Editions Universitaires, Paris, 1989.
- [10] T. Bäck, D. B. Fogel, and T. Michalewicz, editors. *Basic Algorithms and Operators*, volume 1 of *Evolutionary Computation*. Institute of Physics Publishing, Bristol and Philadelphia, 1999.
- [11] A. Bairoch, P. Bucher, and K. Hofmann. The PROSITE database, its status in 1997. *Nucleic Acids Research*, 25(1):217–221, 1997.
- [12] A. Bateman, E. Birney, R. Durbin, S. Eddy, and E. Sonnhammer. The Pfam protein families database. *Nucleic Acids Research*, 28:263–266, 2000.
- [13] E. Beitz. **TEXshade**: shading and labeling multiple sequence alignments using $\text{\LaTeX} 2_{\epsilon}$. *Bioinformatics*, (16):135–139, 2000.
- [14] D. A. Benson, M. Boguski, D. J. Lipman, and J. Ostell. Genbank. *Nucleic Acids Research*, 22(17):3441–3444, September 1994.
- [15] P. Bonizzoni and G. D. Vedova. The complexity of multiple sequence alignment with SP-score that is a metric. *Theoretical Computer Science*, 259(1–2):63–79, 2001.
- [16] R. Brits, A.P. Engelbrecht, and F. van den Bergh. A niching particle swarm optimizer. In *Proceedings of the Fourth Asia-Pacific Conference on Simulated Evolution and Learning*, pages 692–696, 2002.
- [17] C.F. Brunk and L.A. Sadler. Characterization of the promoter region of *Tetrahymena* genes. *Nucleic Acids Research*, 18(2):323–329, 1990.

- [18] N. A. Campbell and J. B. Reece. *Biology*, chapter 16–17. Pearson, seventh edition, 2006.
- [19] H. Carillo and D. Lipman. The multiple sequence alignment problem in biology. *Society for Industrial Applied Mathematics*, 48:1073–1082, 1988.
- [20] A. Carlisle and G. Dozier. An off-the-shelf PSO. In *Proceedings of the Workshop on Particle Swarm Optimization*, pages 1–6, 2001.
- [21] R. Catizone, G. Russell, and S. Warwick-Armstrong. Deriving translation data from bilingual texts. In Zernik, editor, *Lexical Acquisition Workshop*, 1989.
- [22] K. Chellapilla and G. B. Fogel. Multiple sequence alignment using evolutionary programming. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proceedings of the Congress on Evolutionary Computation*, volume 1, pages 445–452, Washington D.C., USA, 6-9 July 1999. IEEE Press.
- [23] Y. Chen, Y. Pan, J. Chen, W. Liu, and L. Chen. Multiple sequence alignment by ant colony optimization and divide-and-conquer. In Vassil N. Alexandrov, G. Dick van Albada, Peter M. A. Sloot, and Jack Dongarra, editors, *International Conference on Computational Science (2)*, volume 3992 of *Lecture Notes in Computer Science*, pages 646–653. Springer, 2006.
- [24] M. Clerc and J. Kennedy. The particle swarm – explosion, stability and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58 – 73, February 2002.
- [25] E. Cloete and R. Atlas, editors. *Basic and Applied Microbiology*, chapter 6. Van Schaik Publishers, 2006.
- [26] C. A. Coello Coello and M. Salazar Lechuga. MOPSO: a proposal for multiple objective particle swarm optimization. In *Congress on Evolutionary Computation*, volume 2, pages 1051–1056, New Jersey, May 2002. IEEE Service Center.
- [27] F. Corpet. Multiple sequence alignment with hierarchial clustering. *Nucleic Acids Research*, 16(22):10881–10890, 1988.

- [28] C. Darwin. *On the Origin of Species by Means of Natural Selection*. J. Murray, London, 1859.
- [29] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. A model of evolutionary change in proteins. In M. O. Dayhoff, editor, *Atlas of Protein Structure*, volume 5 (Suppl. 3), pages 345–352. National Biomedical Research Foundation, Silver Spring, 1979.
- [30] L. N. de Castro and J. Timmis. *Artificial Immune Systems: A New Computational Intelligence Paradigm*. Springer Verlag, November 2002.
- [31] E. Depiereux and E. Feytmans. MATCH-BOX: a fundamentally new algorithm for the simultaneous alignment of several protein sequences. *Computer Applications in the Biosciences*, 8(5):501–509, 1992.
- [32] J. Devereux, P. Haeberli, and O. Smithies. A comprehensive set of sequence analysis programs for the Vax. *Nucleic Acids Research*, 12:387–395, 1984.
- [33] M. Dorigo and G. Di Caro. The ant colony optimization meta-heuristic. In David Corne, Marco Dorigo, and Fred Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, London, 1999.
- [34] R. Eberhart, P. Simpson, and R. Dobbins. *Computational Intelligence—PC Tools*. AP Professional, Academic Press, Inc., 1996.
- [35] R. C. Eberhart and J. Kennedy. *Swarm Intelligence*. Morgan Kaufmann, 2001.
- [36] S. R. Eddy. Multiple alignment using hidden markov models. In *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology*, pages 114–120, 1995.
- [37] R. C. Edgar. MUSCLE: a multiple sequence alignment method with reduced time and space complexity. *BioMed Central Bioinformatics*, 5(1), August 2004.
- [38] A. P. Engelbrecht. *Computational Intelligence: An Introduction*. Wiley and Sons, October 2002.
- [39] A. P. Engelbrecht. *Computational Swarm Intelligence*. Wiley and Sons, 2005.

- [40] S. C. Esquivel and C. A. Coello Coello. On the use of particle swarm optimization with multimodal functions. In *Proceedings of the 2003 Congress on Evolutionary Computation*, volume 2, pages 1130–1136, Canberra, December 2003. IEEE Press.
- [41] D. Feng and R. F. Doolittle. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *Journal of Molecular Evolution*, 25:351–360, 1987.
- [42] A. S. Fraser. Simulation of genetic systems by automatic digital computers I. introduction. *Australian Journal of Biological Sciences*, 10:484–491, 1957.
- [43] W. A. Gale and K. W. Church. A program for aligning sentences in bilingual corpora. *Computational Linguistics*, 19(1):75–102, March 1993.
- [44] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, Reading, MA, 1989.
- [45] D. E. Goldberg and J. Richardson. Genetic algorithm with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 41 – 49, 1987.
- [46] G. H. Gonnet, M. A. Cohen, and S. A. Benner. Exhaustive matching of the entire protein sequence database. *Science*, 256:1443–1445, June 1992.
- [47] Google. 593 ways of spelling Britney Spears, 2006. Accessed <http://www.google.com/jobs/britney.html> on 8 August 2006.
- [48] O. Gotoh. Significant improvement in accuracy of multiple protein sequence alignments by iterative refinement as assessed by reference to structural alignments. *Journal of Molecular Biology*, 264:823–838, 1996.
- [49] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. In *Proceedings of the National Academy of Sciences USA*, volume 89, pages 10915–10919, 1992.
- [50] H. Higashi and H. Iba. Particle swarm optimization with Gaussian mutation. In *Proceedings of the IEEE Swarm Intelligence Symposium*, pages 72–79, April 2003.

- [51] D. G. Higgins and P. M. Sharp. CLUSTAL: a package for performing multiple sequence alignment on a microcomputer. *Gene*, 73:237–244, 1988.
- [52] R. S. Hine and E. Martin, editors. *Oxford Dictionary of Biology*, page 194. Oxford University Press, fifth edition, 2004.
- [53] J. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.
- [54] J. Horng, C. Lin, B. Liu, and C. Kao. Using genetic algorithms to solve multiple sequence alignments. In Darrell Whitley, David Goldberg, Erick Cantu-Paz, Lee Spector, Ian Parmee, and Hans-Georg Beyer, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 883–890, Nevada, USA, July 2000. Morgan Kaufmann.
- [55] Y. Hsiao and C. Chuang. Particle swarm optimization approach for multiple biosequence alignment. In *Proceedings of the Genomic Signal Processing and Statistics*, Rhode Island, May 2005. IEEE press.
- [56] J. W. Hunt and M. D. McIlroy. An algorithm for differential file comparison. Computer Science Technical Report No. 41, Bell Laboratories, New Jersey, June 1976.
- [57] IBM and National Geographic. The genographic project, 2006. Accessed <http://www.ibm.com/za> on 10 September 2006.
- [58] C. Z. Janikow and Z. Michalewicz. An experimental comparison of binary and floating point representations in genetic algorithms. In *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 31 – 36. Morgan Kaufmann, San Diego, USA, 1991.
- [59] W. Just. Computational complexity of multiple sequence alignment with SP-score. *Journal of Computational Biology*, 8(6):615–623, 2001.
- [60] J. D. Kececioğlu. The maximum weight trace problem in multiple sequence alignment. In Alberto Apostolico, Maxime Crochemore, Zvi Galil, and Udi Manber,

- editors, *Combinatorial Pattern Matching, 4th Annual Symposium*, volume 684 of *Lecture Notes in Computer Science*, pages 106–119, Padova, Italy, 2-4 June 1993. Springer.
- [61] J. Kennedy. The behavior of particles. In *Proceedings of the 7th International Conference on Evolutionary Programming*, pages 581–589, 1998.
- [62] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on neural networks*, pages 1942–1948, NJ, 1995. IEEE Service Center.
- [63] J. Kennedy and R. C. Eberhart. A discrete binary version of the particle swarm algorithm. In *Proceedings of the Conference on Systems, Man and Cybernetics*, pages 4104 – 4109, 1997.
- [64] J. Kennedy, R. C. Eberhart, and Y. Shi. *Swarm Intelligence*. Evolutionary Computation Series. Morgan Kaufman, San Francisco, 2001.
- [65] J. Kennedy and R. Mendes. Population structure and particle swarm performance. In *Proceedings of the IEEE World Congress on Evolutionary Computation*, pages 1671 – 1676, Honolulu, Hawaii, May 2002.
- [66] J. Kennedy and W. M. Spears. Matching algorithms to problems: an experimental test of the particle swarm and some genetic algorithms on the multimodal problem generator. In *Proceedings of the IEEE World Congress on Computational Intelligence*, pages 78 – 83, Anchorage, Alaska, 1998.
- [67] J. Kim, S. Pramanik, and M. J. Chung. Multiple sequence alignment using simulated annealing. *Computer Applications in the Biosciences*, 10(4):419–426, 1994.
- [68] D. T. Kingsbury. Computational biology. *ACM Computing Surveys*, 28(1):101–103, March 1996.
- [69] W. S. Klug, M. R. Cummings, and C. A. Spencer. *Concepts of Genetics*, chapter 14. Pearson Education, Inc., eighth edition, 2006.

- [70] T. Lancaster and F. Culwin. Towards an error free plagiarism detection process. In *Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education, University of Kent at Canterbury, England, June 25-27, 2001*, pages 57–60, New York, 2001. ACM SIGCSE, ACM Press.
- [71] C. E. Lawrence, S. F. Altschul, M. S. Boguski, J. S. Liu, A. F. Neuwald, and J. C. Wootton. Detecting subtle sequence signals: a gibbs sampling strategy for multiple alignment. *Science*, 262:208–214, October 1993.
- [72] D. J. Lipman, S. F. Altschul, and J. D. Kececioglu. A tool for multiple sequence alignment. *Proceedings of the National Academy of Science USA*, 86:4412–4415, 1989.
- [73] A. Löytynoja and M. C. Milinkovitch. A hidden Markov model for progressive multiple alignment. *Bioinformatics*, 19(12):1505–1513, 2003.
- [74] V. Miranda and N. Fonseca. EPSO - best-of-two worlds meta-heuristic applied to power system problems. In David B. Fogel, Mohamed A. El-Sharkawi, Xin Yao, Garry Greenwood, Hitoshi Iba, Paul Marrow, and Mark Shackleton, editors, *Proceedings of the 2002 Congress on Evolutionary Computation*, pages 1080–1085. IEEE Press, 2002.
- [75] B. Morgenstern. DIALIGN 2: improvement of the segment-to-segment approach to multiple sequence alignment. *Bioinformatics*, 15(3):211–218, 1999.
- [76] B. Morgenstern, K. Frech, A. W. M. Dress, and T. Werner. DIALIGN: finding local similarities by multiple sequence alignment. *Bioinformatics*, 14(3):290–294, 1998.
- [77] B. Morgenstern, J. Stoye, and A. Dress. Consistent equivalence relations: a set-theoretical framework for multiple sequence alignment. *Technical Report Materialien und Preprints 133, University of Bielefeld*, 1999.
- [78] B. Morgenstern and T. Werner. DIALIGN 1.0: multiple alignment by segment rather than by position comparison. In *German Conference on Bioinformatics*, pages 69–71, 1997.

- [79] J. D. Moss and C. G. Johnson. An ant colony algorithm for multiple sequence alignment in bioinformatics. In David W. Pearson, Nigel C. Steele, and Rudolf F. Albrecht, editors, *Artificial Neural Networks and Genetic Algorithms*, pages 182–186. Springer, April 2003.
- [80] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarity in the amino acid sequences of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.
- [81] C. Notredame. Recent progress in multiple sequence alignment: a survey. *Pharmacogenomics*, 3(1):131–144, January 2002.
- [82] C. Notredame and D. G. Higgins. SAGA: sequence alignment by genetic algorithm. *Nucleic Acids Research*, 24:1515 – 1524, April 1996.
- [83] C. Notredame, D. G. Higgins, and J. Heringa. T-COFFEE: a novel method for fast and accurate multiple sequence alignment. *Journal of Molecular Biology*, 302(1):205–217, September 2000.
- [84] C. Notredame, L. Holm, and D. G. Higgins. COFFEE: an objective function for multiple sequence alignments. *Bioinformatics*, 14(5):407–422, 1998.
- [85] U.S. Department of Energy and the National Institutes of Health. Human genome project, December 2005. Accessed http://www.ornl.gov/sci/techresources/Human_Genome/project/about.shtml on 10 October 2006.
- [86] E. Ozcan and C. K. Mohan. Analysis of a simple particle swarm optimization system. In *Intelligent Engineering Systems Through Artificial Neural Networks*, volume 8, pages 253 – 258, 1998.
- [87] E. Ozcan and C. K. Mohan. Particle swarm optimization: surfing the waves. In *Proceedings of the International Congress on Evolutionary Computation*, pages 1939 – 1944, Washington, USA, 1999.
- [88] K. E. Parsopoulos and M. N. Vrahatis. *Particle swarm optimization method for constrained optimization*, volume 76, pages 214 – 220. IOS Press, 2002.

- [89] W. R. Pearson. Rapid and sensitive sequence comparison with fastp and fasta. *Methods Enzymol*, 183:63–98, 1990.
- [90] W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Acadademy of Science USA*, 85(8):2444–2448, April 1988.
- [91] E. S. Peer. A serendipitous software framework for facilitating collaboration in computational intelligence. Master’s thesis, Department of Computer Science, University of Pretoria, Pretoria, South Africa, 2004.
- [92] M. A. Potter. *The Design and Analysis of a Computational Model of Cooperative Coevolution*. PhD thesis, George Mason University, Fairfax, Virginia, USA, 1997.
- [93] T. K. Rasmussen and T. Krink. Improved hidden Markov model training for multiple sequence alignment by a particle swarm optimization - evolutionary algorithm hybrid. *Biosystems*, 72(1–2):5–17, November 2003.
- [94] K. Reinert, J. Stoye, and T. Will. An iterative method for faster sum-of-pairs multiple sequence alignment. *Bioinformatics*, 16(9):808–814, 2000.
- [95] E. Ruppín and J. A. Reggia. Seeking order in disorder: computational studies of neurologic and psychiatric diseases. *Artificial Intelligence in Medicine*, 13(1-2):1–12, 1998.
- [96] B. R. Secrest and G. B. Lamont. Visualizing particle swarm optimization - Gaussian particle swarm optimization. In *Proceedings of the IEEE Swarm Intelligence Symposium*, pages 198–204, 2003.
- [97] J. Setubal and J. Meidanis. *Introduction to Computational Molecular Biology*. Brooks-Cole, 1997.
- [98] Y. Shi and R. C. Eberhart. A modified particle swarm optimizer. In *Proceedings of IEEE World Conference on Computational Intelligence*, pages 69–73, Anchorage, Alaska, May 1998. IEEE Service Center.

- [99] Y. Shi and R. C. Eberhart. Parameter selection in particle swarm optimization. In *Proceedings of Evolutionary Programming VII*, pages 561–600, Springer Verlag, 1998.
- [100] Y. Shi and R. C. Eberhart. Empirical study of particle swarm optimization. In *Proceedings of Congress on Evolutionary Computation*, pages 1945–1950, Piscataway, NJ, 1999. IEEE Service Center.
- [101] C. Shyu, L. Sheneman, and J. A. Foster. Multiple sequence alignment with evolutionary computation. *Genetic Programming and Evolvable Machines*, 5(2):121–144, 2004.
- [102] P. R. Sibbald and P. Argos. Weighting aligned protein or nucleic acid sequences to correct for unequal representation. *Journal of Molecular Biology*, (216):813–818, 1990.
- [103] A. Stacey, M. Jancic, and I. Grundy. Particle swarm optimization with mutation. In Ruhul Sarker, Robert Reynolds, Hussein Abbass, Kay Chen Tan, Bob McKay, Daryl Essam, and Tom Gedeon, editors, *Proceedings of the 2003 Congress on Evolutionary Computation*, pages 1425–1430, Canberra, 8-12 December 2003. IEEE Press.
- [104] J. Stoye. Divide-and-conquer multiple sequence alignment. Master’s thesis, Technische Fakultät der Universität Bielefeld, 1997.
- [105] J. Stoye, S. Perrey, and A. Dress. Improving the divide-and-conquer approach to sum-of-pairs multiple sequence alignment. *Applied Mathematics Letters*, 10, 1997.
- [106] A. R. Subramanian, J. Weyer-Menkhoff, M. Kaufmann, and B. Morgenstern. DIALIGN-T: an improved algorithm for segment-based multiple sequence alignment. *Bioinformatics*, 6:66, 2005.
- [107] P. N. Suganthan. Particle swarm optimizer with neighborhood operator. *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1958 – 1961, July 1999.

- [108] W. R. Taylor. Multiple sequence alignment by a pairwise algorithm. *Computer Applications in the Biosciences*, 3(2):81–87, 1987.
- [109] W. R. Taylor. A flexible method to align large numbers of biological sequences. *Journal of Molecular Evolution*, 28:161–169, 1988.
- [110] W. R. Taylor, G. Salensminde, and I. Eidhammer. Multiple protein sequence alignment using double-dynamic programming. *Computers & Chemistry*, 24(1):3–12, 2000.
- [111] J. Thompson, F. Plewniak, and O. Poch. A comprehensive comparison of multiple sequence alignment programs. *Nucleic Acids Research*, 27(13):2682–2690, 1999.
- [112] J. D. Thompson, T. J. Gibson, F. Plewniak, F. Jeanmougin, and D. G. Higgins. The CLUSTAL X windows interface: flexible strategies for multiple sequence alignment aided by quality analysis tools. April 2003.
- [113] J. D. Thompson, D. G. Higgins, and T. J. Gibson. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22:4673–4680, 1994.
- [114] J. D. Thompson, F. Plewniak, and O. Poch. BALiBASE: a benchmark alignment database for the evaluation of multiple alignment programs. *Bioinformatics*, 15(1):87–88, 1999.
- [115] F. van den Bergh. *An Analysis of Particle Swarm Optimizers*. PhD thesis, Department of Computer Science, University of Pretoria, Pretoria, South Africa, 2002.
- [116] F. van den Bergh and A. P. Engelbrecht. Cooperative learning in neural networks using particle swarm optimizers. *South African Computer Journal*, 26:84–90, 2000.
- [117] F. van den Bergh and A. P. Engelbrecht. A cooperative approach to particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):225–239, 2004.

- [118] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1(4):337–348, 1994.
- [119] S. Wehmeier, editor. *Oxford Advanced Learner's Dictionary*, pages 35–36. Oxford University Press, seventh edition, 2005.
- [120] X. Xie, W. Zhang, and Z. Yang. A dissipative particle swarm optimization. In *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1456–1461, May 2002.
- [121] C. Zhang and A. K. Wong. A genetic algorithm for multiple molecular sequence alignment. *Computer Applications in the Biosciences*, 13(6):565–581, 1997.
- [122] H. Zhang and M. Ishikawa. Evolutionary particle swarm optimization (EPSO) - estimation of optimal PSO parameters by GA. In S. I. Ao, Oscar Castillo, Craig Douglas, David Dagan Feng, and Jeong-A Lee, editors, *Proceedings of the International MultiConference of Engineers and Computer Scientists 2007 Volume I, Hong Kong*, Lecture Notes in Engineering and Computer Science, pages 13–18. International Association of Engineers, Newswood Limited, 2007.

Appendix A

Data Sets

This appendix offers information about real data sets used throughout the research experiments. Information regarding each data set is composed of accession IDs and the data set's source. All sequences can be retrieved at the European Bioinformatics Institute website from the EMBL database at <http://www.ebi.ac.uk> by entering the accession IDs in the query.

Accession IDs for all data sets follow:

Data Set S1 [121] (DNA): HCV2L1A10 HCV2L3A5 HCV2L3C1 HCV2L3C8 HCV2L3D4
HCV2L3E6 HCV2L3A7 HCV2L3A9 HCV2L3B2 HCV2L3B1

Data Set S2 [121] (DNA): HS06674 HS06675 HS06676 HS06677 HS06679

Data Set S3 [17] (DNA): TPAHISIN TNIHISIN TNHISIN TMIHISIN TMHISIN
THHISIN TFHISIN TEHISIN TCUHISIN TCHISIN TBHISIN TAUHISIN
TAHISIN TTHISIN TSHISIN TRHISIN TPYHISIN TPIHISIN TPHISIN
TCAHISIN TLHISIN

Data Set S4 [121] (DNA): HI1U16764 HI1U16766 HI1U16768 HI1U16776 HI1U16778
HI1U16770 HI1U16774 HI1U16772

Data Set S5 [121] (DNA): HI1U16765 HI1U16767 HI1U16769 HI1U16771 HI1U16773
HI1U16775 HI1U16777 HI1U16779

Data Set S6 [121] (mRNA): PP59651 PP59652 PP59653 PP59654 PP59655
PP59656

Data Set S7 [14] (rRNA): AB023287 AB023286 AB023285 AB023284 AB023283
AB023279 AB023278 AB023276

Appendix B

Cilib XML Configuration

This appendix lists the content of an XML configuration file which illustrates how simulations are generated within Cilib version 0.6.5. Because of space constraints, all simulations have not been included, but rather a representative example of each of the four PSOs used throughout the experiments is reproduced here. The XML content is divided into four sections:

- Algorithm definitions: Four PSO algorithms were defined, in order – the S-PSO (“Lbest-PSO” ID tag), one CPSO- S_6 participant (“LbestPSO50-Matches” ID tag), the B-PSO (“BinaryPSO-Explorer” ID tag) and the M-PSO (“MutPSO” ID tag).
- Problem definitions: Four different MSA problems were defined, in order – S8 (in binary mode) with 3 gaps allowed and the matches method (“Mmsa-3gap” ID tag), S6 with 65 gaps allowed and the similarity method (“Smsa-gapS6” ID tag), S2 with 134 gaps allowed and the matches method (“Mmsa-gapS2” ID tag) and S7 with 1 gap allowed and the similarity method (“Smsa-gapS7” ID tag).
- Measurement definitions: Two different measurement suites were defined – both launch 30 simulations (**samples**), report a single fitness every 250 iterations (**resolution**) and the actual alignment solution. The first is used in conjunction with all PSOs except B-PSO (“**measurements1**” ID tag) and the second one is used exclusively in conjunction with the B-PSO (“**measurements2**” ID tag).
- Simulation definitions: Four different simulations were defined, in order – the S-

PSO solving S7 (“Lbest-PSO” ID tag), the B-PSO solving S8 (“BinaryPSO-Explorer” ID tag), M-PSO solving S6 (“MutPSO” ID tag) and CPSO- S_6 solving S2 (“coop-pso” ID tag).

The XML listing follows:

```
<?xml version="1.0"?> <!DOCTYPE simulator [
  <!--ATTLIST algorithm id ID #IMPLIED-->
  <!--ATTLIST problem id ID #IMPLIED-->
  <!--ATTLIST measurements id ID #IMPLIED-->]
<simulator>

<!--*****-->
<!--Algorithm Definitions-->
<!--*****-->

  <algorithm id="Lbest-PSO" class="pso.PSO" particles="25">
    <addStoppingCondition class="stoppingcondition.MaximumIterations" iterations="1000"/>
    <topology class="entity.topologies.LBestTopology" neighbourhoodSize="3"/>
    <iterationStrategy class="pso.iterationstrategies.ASynchronousIterationStrategy"/>
    <initialisationStrategy class="algorithm.initialisation.ClonedEntityInitialisationStrategy">
      <prototypeEntity class="pso.particle.StandardParticle">
        <positionUpdateStrategy class="
          "pso.positionupdatestrategies.StandardPositionUpdateStrategy" />
        <velocityInitialisationStrategy class="
          "pso.particle.initialisation.RandomInitialVelocityStrategy" />
      </prototypeEntity>
    </initialisationStrategy>
  </algorithm>

  <algorithm id="LbestPSO50-Matches" class="pso.PSO" particles="50">
    <topology class="entity.topologies.LBestTopology" neighbourhoodSize="6"/>
    <iterationStrategy class="pso.iterationstrategies.ASynchronousIterationStrategy"/>
  </algorithm>

  <algorithm id="BinaryPSO-Explorer" class="pso.PSO" particles="30">
    <addStoppingCondition class="stoppingcondition.MaximumIterations" iterations="2000"/>
    <topology class="entity.topologies.VonNeumannTopology"/>
    <iterationStrategy class="pso.iterationstrategies.ASynchronousIterationStrategy"/>
    <initialisationStrategy class="algorithm.initialisation.ClonedEntityInitialisationStrategy">
      <prototypeEntity class="pso.particle.StandardParticle">
        <positionUpdateStrategy class="
          "pso.positionupdatestrategies.BinaryPositionUpdateStrategy" />
        <velocityInitialisationStrategy class="
          "pso.particle.initialisation.ZeroInitialVelocityStrategy" />
        <velocityUpdateStrategy class="
          "pso.velocityupdatestrategies.StandardVelocityUpdate"/>
      </prototypeEntity>
    </initialisationStrategy>
  </algorithm>
```

```

        <vMax class=
        "controlparameterupdatestrategies.LinearIncreasingUpdateStrategy"
            range="R(0.001, 4.0)"/>
        <inertiaWeight class=
            "controlparameterupdatestrategies.LinearIncreasingUpdateStrategy"
                range="R(0.001, 0.7)"/>
        <cognitiveAcceleration class=
            "controlparameterupdatestrategies.LinearIncreasingUpdateStrategy"
                range="R(0.5, 2.0)"/>
        <socialAcceleration class=
            "controlparameterupdatestrategies.LinearIncreasingUpdateStrategy"
                range="R(0.5, 2.0)"/>
    </velocityUpdateStrategy>
</prototypeEntity>
</initialisationStrategy>
</algorithm>

<algorithm id="MutPSO" class="pso.PSO" particles="25">
    <addStoppingCondition class="stoppingcondition.MaximumIterations" iterations= "1000" />
    <topology class="entity.topologies.LBestTopology" neighbourhoodSize = "3"/>
    <iterationStrategy class= "pso.iterationstrategies.ASynchronousIterationStrategy"/>
    <initialisationStrategy class="algorithm.initialisation.ClonedEntityInitialisationStrategy">
        <prototypeEntity class="pso.particle.StandardParticle">
            <positionUpdateStrategy class=
                "pso.positionupdatestrategies.GaussianPositionUpdateStrategy" bestReplace="true">
                <pMut class=
                    "controlparameterupdatestrategies.LinearDecreasingUpdateStrategy"
                        range = "R(0.9, 0.1)"/>
            </positionUpdateStrategy>
            <velocityInitialisationStrategy class=
                "pso.particle.initialisation.RandomInitialVelocityStrategy" />
        </prototypeEntity>
    </initialisationStrategy>
</algorithm>

<!--*****>
<!--Problem Definitions-->
<!--*****>

<problems>

    <problem id="Mmsa-3gap" class="bioinf.sequencealignment.BinaryMSAProblem" weight1="1.0" weight2="1.0">
        <dataSetBuilder class="bioinf.sequencealignment.FASTADatasetBuilder">
            <dataSet class="problem.dataset.LocalDataSet" file="inputSeq/S8.fasta" />
        </dataSetBuilder>
        <alignmentCreator class="bioinf.sequencealignment.BinaryAlignmentCreator" justEvaluate="false">
            <scoringMethod class= "bioinf.sequencealignment.MatchFogel" verbose="false"/>
        </alignmentCreator>
        <gapPenaltyMethod class= "bioinf.sequencealignment.GapOpeningAndExtensionPenalty"

```

```

        gapOpeningPenalty = "2" gapExtensionPenalty = "1" verbose="false"/>
    <maxSequenceGapsAllowed value="3"/>
</problem>

<problem id="Smsa-gapS6" class="bioinf.sequencealignment.MSAPProblem" weight1="1.0" weight2="1.0">
    <dataSetBuilder class="bioinf.sequencealignment.FASTADatasetBuilder">
        <dataSet class="problem.dataset.LocalDataSet" file="inputSeq/S6.fasta" />
    </dataSetBuilder>
    <alignmentCreator class="bioinf.sequencealignment.AlignmentCreator" justEvaluate="false">
        <scoringMethod class="bioinf.sequencealignment.Similarity" MATCH="2" MISMATCH="0"
            GAP_PENALTY="-1" weight="true" verbose="false"/>
    </alignmentCreator>
    <gapPenaltyMethod class="bioinf.sequencealignment.GapOpeningAndExtensionPenalty"
        gapOpeningPenalty = "2" gapExtensionPenalty = "1" verbose="false"/>
    <maxSequenceGapsAllowed value="65"/>
</problem>

<problem id="Mmsa-gapS2" class="bioinf.sequencealignment.MSAPProblem" weight1="1.0" weight2="1.0">
    <dataSetBuilder class="bioinf.sequencealignment.FASTADatasetBuilder">
        <dataSet class="problem.dataset.LocalDataSet" file="inputSeq/S2.fasta" />
    </dataSetBuilder>
    <alignmentCreator class="bioinf.sequencealignment.AlignmentCreator" justEvaluate="false">
        <scoringMethod class="bioinf.sequencealignment.MatchFogel"
            verbose="false" linearScale="true"/>
    </alignmentCreator>
    <gapPenaltyMethod class="bioinf.sequencealignment.GapOpeningAndExtensionPenalty"
        gapOpeningPenalty = "2" gapExtensionPenalty = "1" verbose="false"/>
    <maxSequenceGapsAllowed value="134"/>
</problem>

<problem id="Smsa-gapS7" class="bioinf.sequencealignment.MSAPProblem" weight1="1.0" weight2="1.0">
    <dataSetBuilder class="bioinf.sequencealignment.FASTADatasetBuilder">
        <dataSet class="problem.dataset.LocalDataSet" file="inputSeq/S7.fasta" />
    </dataSetBuilder>
    <alignmentCreator class="bioinf.sequencealignment.AlignmentCreator" justEvaluate="false">
        <scoringMethod class="bioinf.sequencealignment.Similarity" MATCH="2"
            MISMATCH="0" GAP_PENALTY="-1" weight="true" verbose="false"/>
    </alignmentCreator>
    <gapPenaltyMethod class="bioinf.sequencealignment.GapOpeningAndExtensionPenalty"
        gapOpeningPenalty = "2" gapExtensionPenalty = "1" verbose="false"/>
    <maxSequenceGapsAllowed value="1"/>
</problem>

</problems>

<!--*****-->
<!--Measurement Definitions-->
<!--*****-->

```

```

<measurements id="measurements1" class="simulator.MeasurementSuite" resolution="250" samples="30">
  <addMeasurement class="measurement.single.Fitness" />
  <addMeasurement class="bioinf.sequencealignment.AlignmentVisualizer" fullColumns = "true" />
</measurements>

<measurements id="measurements2" class="simulator.MeasurementSuite" resolution="250" samples="30">
  <addMeasurement class="measurement.single.Fitness" />
  <addMeasurement class="bioinf.sequencealignment.BinaryAlignmentVisualizer" fullColumns="true"/>
</measurements>

<!--*****-->
<!--Simulation Definitions-->
<!--*****-->

<simulations>

  <simulation>
    <algorithm idref="Lbest-PS0" />
    <problem idref="Smsa-gapS7" />
    <measurements idref="measurements1" file="output/S7_similarity_S-PS0.txt" />
  </simulation>

  <simulation>
    <algorithm idref="BinaryPS0-Explorer" />
    <problem idref="Mmsa-3gap" />
    <measurements idref="measurements2" file="output/S8_3gaps_matches_B-PS0Explorer.txt" />
  </simulation>

  <simulation>
    <algorithm idref="MutPS0" />
    <problem idref="Smsa-gapS6" />
    <measurements idref="measurements1" file="output/S6_similarity_M-PS0trueLinDec.txt" />
  </simulation>

  <simulation>
    <algorithm id="coop-pso" class="cooperative.SplitCooperativeAlgorithm">
      <addStoppingCondition class="stoppingcondition.MaximumIterations" iterations="1000"/>
      <algorithm idref="LbestPS050-Matches"/>
      <algorithm idref="LbestPS050-Matches"/>
      <algorithm idref="LbestPS050-Matches"/>
      <algorithm idref="LbestPS050-Matches"/>
      <algorithm idref="LbestPS050-Matches"/>
      <algorithm idref="LbestPS050-Matches"/>
      <splitStrategy class="cooperative.splitstrategies.PerfectSplitStrategy"/>
      <populationIterator class="cooperative.populationiterators.SequentialPopulationIterator"/>
      <contributionUpdateStrategy class=
        "cooperative.contributionupdatestrategies.StandardContributionUpdateStrategy"/>
      <fitnessUpdateStrategy class=
        "cooperative.fitnessupdatestrategies.StandardFitnessUpdateStrategy"/>
    </algorithm>
  </simulation>

```



```
</algorithm>
  <problem idref="Mmsa-gapS2" />
  <measurements idref="measurements1" file="output/S2_matches_C-PS0_6.txt" />
</simulation>
</simulations>
</simulator>
```

Appendix C

Acronyms & Abbreviations

This appendix lists the acronyms and abbreviations used throughout this thesis. The list is sorted alphabetically, with acronyms and abbreviations typeset in bold. The meaning associated with each acronym or abbreviation is provided alongside.

ACO	Ant Colony Optimization
aGP	affine Gap Penalty
AI	Artificial Intelligence
B-PSO	Binary Particle Swarm Optimization
BLAST	Basic Local Alignment Search Tool
BLOSUM	BLOcks SUBstitution Matrices
CCGA	Cooperative Coevolutionary Genetic Algorithm
CI	Computational Intelligence
Ci	Confidence interval
CIlib	Computational Intelligence library
CIRG	Computational Intelligence Research Group
CPSO-S_K	Cooperative Split Particle Swarm Optimization
CPU	Central Processing Unit
CSI	Computational Swarm Intelligence

DCA	Divide-and-Conquer Algorithm
DNA	Deoxyribonucleic Acid
DoE	Design of Experiments
DP	Dynamic Programming
EA	Evolutionary Algorithm
EBI	European Bioinformatics Institute
EC	Evolutionary Computation
EF	Evaluation Function
EP	Evolutionary Programming
GA	Genetic Algorithm
GBest	Global Best
HGP	Human Genome Project
HMM	Hidden Markov Model
IBM	International Business Machine
IS	Identity Score
LBest	Local Best
lGP	linear Gap Penalty
MSA	Multiple Sequence Alignment
M-PSO	Mutating PSO
MOPSO	Multiple Objective PSO
MWTP	Maximum Weight Trace Problem
NP	Nondeterministic Polynomial
OF	Objective Function
PAM	Point Accepted Mutation
PCR	Polymerase Chain Reaction
PDF	Portable Document File
PID	Percentage Identity
PSA	Pairwise Sequence Alignment
PSO	Particle Swarm Optimization
RNA	Ribonucleic Acid
SA	Simulated Annealing
SAGA	Sequence Alignment by Genetic Algorithm



SI	Swarm Intelligence
SoP	Sum of Pairs
T-COFFEE	Tree-based Consistency Objective Function For alignmEnt Evaluation
TSP	Traveling Salesman Problem
XML	Extensible Markup Language

Appendix D

Symbols

This appendix lists, alphabetically, the mathematical symbols used within this thesis, along with their definitions.

α	Confidence percentage
\mathcal{A}	Alphabet
$\hat{\mathcal{A}}$	Extended alphabet
β	Total number of possible combinations of an alignment
\mathbb{B}^m	Binary-valued, multidimensional domain
c_1, c_2	Acceleration coefficients
C_g	Population at g^{th} generation
$C_{g,q}$	Individual q at g^{th} generation
$\Delta(t, z)$	Non-uniform mutating function
f	Objective function
$f(\mathbf{x})$	Function of \mathbf{x}
\mathbf{f}	Objective function vector
$f(\mathbf{y}_s(t))$	Entire swarm evaluation of personal best positions at time step t
f_ϕ	Minimum fitness threshold
$f(\mathbf{x}_i(t))$	Fitness evaluation function of particle i at time step t
g	Generation counter
g_{ext}	Extension gap penalty

g_{fixed}	Fixed gap penalty
g_{max}	Generation limit
g_{op}	Opening gap penalty
i	Particle index
i, j	Sequence index within a set
j	Dimension index
k	Number of sequences within a set
k	Sub-swarm index
K	Split factor
κ	Number of indels
λ	Degree of dependency
M	Matches method
M_c	Number of matches in column c
m	Number of dimensions in hyperspace
MSA_{func}	MSA objective/evaluation function
n	Length of sequence \mathcal{S}
$N(0, \sigma)$	Samples a random number from a normal Gaussian distribution with 0 mean and σ deviation
\mathfrak{N}_i	Neighbourhood of particle i
\hat{n}	Length of aligned sequence $\hat{\mathcal{S}}$
O_g	Generated offspring at g^{th} generation
$O_{g,q}$	Generated offspring q at g^{th} generation
Ω	m -dimensional search space
ω	Inertia weight
p	Number of objective in \mathbf{f}
Φ	Swarm of particles
$P_m(t)$	Mutation probability
\mathbb{R}^m	Real-valued, multidimensional domain
r	Random value

$r_{1,j}(t)$	Random value 1 for j^{th} dimension at time step t
$r_{2,j}(t)$	Random value 2 for j^{th} dimension at time step t
$r_{3,j}(t)$	Random value 3 for j^{th} dimension at time step t
r_1, r_2	Random values
\mathcal{S}	Sequence
S_k	k^{th} sub-swarm
s	Size of swarm, i.e. number of particles
$s_{\mathbb{N}_i}$	Number of particles in the neighbourhood of particle i
$S_{k.s}$	Size of k^{th} sub-swarm
$sig(v_{i,j}(t))$	Sigmoid function of $v_{i,j}(t)$ at time step t
$\hat{\mathcal{S}}$	Aligned sequence
S	Similarity method
s_p	Symbol at position p in sequence \mathcal{S}
$\mathcal{S}[p]$	Symbol at position p in sequence \mathcal{S}
$\mathcal{S}[p, q]$	Sub-sequence delimited by index p and q within sequence \mathcal{S}
$ \mathcal{S} $	Length of sequence \mathcal{S}
\mathcal{T}	Set of k sequences
$\hat{\mathcal{T}}$	Set of k aligned sequences
T	Total number of iterations
t	Time step counter
$U(a, b)$	Samples a random number between a and b from an uniform distribution
V_{max}	Maximum velocity
$v_{i,j}(t)$	Velocity value in j^{th} dimension of particle i at time step t
\mathbf{v}_i	Velocity vector of particle i
w	Alignment score
w^*	Optimal alignment score
x_i	Value at i^{th} position in \mathbf{x}
$x_{i,j}(t)$	j^{th} component of position vector of particle i at time step t
$x'_{i,j}(t)$	j^{th} mutated component of position vector of particle i at time step t

\mathbf{x}	Vector of values over Ω
\mathbf{x}_{min}	Vector of minimum values for all \mathbf{x}_i
\mathbf{x}_{max}	Vector of maximum values for all \mathbf{x}_i
\mathbf{x}_i	Position vector of i^{th} particle
\mathbf{x}^*	Solution vector for minimization
ξ_1, ξ_2	Weights associated to <i>SymbolScore</i> and <i>GapScore</i> , respectively
\mathbf{y}_i	Personal best position vector of particle i
$y_{i,j}(t)$	j^{th} component of personal best position vector of particle i at time step t
$\hat{\mathbf{y}}_i(t)$	Best position vector in neighbourhood of particle i at time step t
$\hat{\mathbf{y}}(t)$	Best position vector in swarm at time step t
$\hat{y}_{i,j}(t)$	j^{th} component of best position vector in \aleph_i at time step t
\mathbf{z}	Position vector used in context vector

Appendix E

Alphabets and S. Matrices

This appendix displays tables of residue alphabets and common substitution matrices currently used in bioinformatics. Tables E.1, E.2 and E.3 show the names of the corresponding symbols from the DNA, RNA and amino acids alphabets, respectively.

Table E.1: Residue alphabet for DNA

DNA code (4)	Meaning
A	Adenine
C	Cytosine
G	Guanine
T	Thymine

Table E.2: Residue alphabet for RNA

RNA code (4)	Meaning
A	Adenine
C	Cytosine
G	Guanine
U	Uracil

A presentation of a typical identity matrix used for DNA substitutions is given in Table E.4.

Table E.3: Residue alphabet for amino acids (proteins)

Amino Acid code (24)	Meaning
A	Alanine
E	Glutamate
H	Histidine
L	Leucine
P	Proline
S	Serine
W	Tryptophan
C	Cysteine
F	Phenylalanine
I	Isoleucine
M	Methionine
Q	Glutamine
T	Threonine
Y	Tyrosine
D	Aspartate
G	Glycine
K	Lysine
N	Asparagine
R	Arginine
V	Valine
B	Aspartic acid or Asparagine
Z	Glutamic acid or Glutamine
X	any
*	translation stop

Table E.4: A typical DNA identity matrix

-	A	C	T	G
A	1	-100	-100	-100
C	-100	1	-100	-100
T	-100	-100	1	-100
G	-100	-100	-100	1

The full PAM250 and BLOSUM62 substitution matrices, both of which used for protein sequence alignments, are respectively illustrated in Figure E.1 and Figure E.2.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	Z	X	*
A	2	-2	0	0	-2	0	0	1	-1	-1	-2	-1	-1	-3	1	1	1	-6	-3	0	0	0	0	-8
R	-2	6	0	-1	-4	1	-1	-3	2	-2	-3	3	0	-4	0	0	-1	2	-4	-2	-1	0	-1	-8
N	0	0	2	2	-4	1	1	0	2	-2	-3	1	-2	-3	0	1	0	-4	-2	-2	2	1	0	-8
D	0	-1	2	4	-5	2	3	1	1	-2	-4	0	-3	-6	-1	0	0	-7	-4	-2	3	3	-1	-8
C	-2	-4	-4	-5	12	-5	-5	-3	-3	-2	-6	-5	-5	-4	-3	0	-2	-8	0	-2	-4	-5	-3	-8
Q	0	1	1	2	-5	4	2	-1	3	-2	-2	1	-1	-5	0	-1	-1	-5	-4	-2	1	3	-1	-8
E	0	-1	1	3	-5	2	4	0	1	-2	-3	0	-2	-5	-1	0	0	-7	-4	-2	3	3	-1	-8
G	1	-3	0	1	-3	-1	0	5	-2	-3	-4	-2	-3	-5	0	1	0	-7	-5	-1	0	0	-1	-8
H	-1	2	2	1	-3	3	1	-2	6	-2	-2	0	-2	-2	0	-1	-1	-3	0	-2	1	2	-1	-8
I	-1	-2	-2	-2	-2	-2	-2	-3	-2	5	2	-2	2	1	-2	-1	0	-5	-1	4	-2	-2	-1	-8
L	-2	-3	-3	-4	-6	-2	-3	-4	-2	2	6	-3	4	2	-3	-3	-2	-2	-1	2	-3	-3	-1	-8
K	-1	3	1	0	-5	1	0	-2	0	-2	-3	5	0	-5	-1	0	0	-3	-4	-2	1	0	-1	-8
M	-1	0	-2	-3	-5	-1	-2	-3	-2	2	4	0	6	0	-2	-2	-1	-4	-2	2	-2	-2	-1	-8
F	-3	-4	-3	-6	-4	-5	-5	-5	-2	1	2	-5	0	9	-5	-3	-3	0	7	-1	-4	-5	-2	-8
P	1	0	0	-1	-3	0	-1	0	0	-2	-3	-1	-2	-5	6	1	0	-6	-5	-1	-1	0	-1	-8
S	1	0	1	0	0	-1	0	1	-1	-1	-3	0	-2	-3	1	2	1	-2	-3	-1	0	0	0	-8
T	1	-1	0	0	-2	-1	0	0	-1	0	-2	0	-1	-3	0	1	3	-5	-3	0	0	-1	0	-8
W	-6	2	-4	-7	-8	-5	-7	-7	-3	-5	-2	-3	-4	0	-6	-2	-5	17	0	-6	-5	-6	-4	-8
Y	-3	-4	-2	-4	0	-4	-4	-5	0	-1	-1	-4	-2	7	-5	-3	-3	0	10	-2	-3	-4	-2	-8
V	0	-2	-2	-2	-2	-2	-2	-1	-2	4	2	-2	2	-1	-1	-1	0	-6	-2	4	-2	-2	-1	-8
B	0	-1	2	3	-4	1	3	0	1	-2	-3	1	-2	-4	-1	0	0	-5	-3	-2	3	2	-1	-8
Z	0	0	1	3	-5	3	3	0	2	-2	-3	0	-2	-5	0	0	-1	-6	-4	-2	2	3	-1	-8
X	0	-1	0	-1	-3	-1	-1	-1	-1	-1	-1	-1	-1	-2	-1	0	0	-4	-2	-1	-1	-1	-1	-8
*	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	1

Figure E.1: The PAM250 S. matrix

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	Z	X	*
A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0	-2	-1	0	-4
R	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3	-1	0	-1	-4
N	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3	3	0	-1	-4
D	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3	4	1	-1	-4
C	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1	-3	-3	-2	-4
Q	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2	0	3	-1	-4
E	-1	0	0	2	-4	2	5	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2	1	4	-1	-4
G	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3	-1	-2	-1	-4
H	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1	-2	-1	-2	-1	-2	-2	2	-3	0	0	-1	-4
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3	1	0	-3	-2	-1	-3	-1	3	-3	-3	-1	-4
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1	-4	-3	-1	-4
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5	-1	-3	-1	0	-1	-3	-2	-2	0	1	-1	-4
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2	-1	-1	-1	-1	1	-3	-1	-1	-4
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6	-4	-2	-2	1	3	-1	-3	-3	-1	-4
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7	-1	-1	-4	-3	-2	-2	-1	-2	-4
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2	0	0	0	-4
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0	-1	-1	0	-4
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11	2	-3	-4	-3	-2	-4
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1	-3	-2	-1	-4
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4	-3	-2	-1	-4
B	-2	-1	3	4	-3	0	1	-1	0	-3	-4	0	-3	-3	-2	0	-1	-4	-3	-3	4	1	-1	-4
Z	-1	0	0	1	-3	3	4	-2	0	-3	-3	1	-1	-3	-1	0	-1	-3	-2	-2	1	4	-1	-4
X	0	-1	-1	-1	-2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-2	0	0	-2	-1	-1	-1	-1	-1	-4
*	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	-4	1

Figure E.2: The BLOSUM62 S. matrix

Index

- alphabet
 - definitions, 11
- binary PSO
 - introduction, 47
 - pseudocode, 48
- bioinformatics, 1, 7
 - biological data, 1
- Cilib, 63
- CLUSTAL, 27
- computational intelligence, 1, 32
- cooperative split PSO
 - introduction, 50
 - pseudocode, 51
- Darwin
 - theories, 9
- dynamic programming, 2, 26
- gaps
 - indels, 17
 - penalty, 17
- genetic algorithm, 29
 - cross-over, 36
 - introduction, 34
 - mutation, 37
 - population, 34
 - pseudocode, 35
 - selection, 38
- genetic information
 - genotype, 7
 - phenotype, 7
- hidden markov model, 28
- human genome project, 6
- matches approach, 16
 - example, 16
- multiple sequence alignment, 6
 - exact approach, 26
 - computer complexity, 24
 - consistency-based approach, 29
 - difficulty, 24
 - evolutionary algorithms, 29
 - iterative approach, 28
 - problem, 23
 - progressive approach, 27
 - scoring function, 24
 - statement, 2
- mutating PSO
 - introduction, 48
 - pseudocode, 49
- nucleic acids
 - deoxyribonucleic acid, 7

- ribonucleic acid, 7
- optimization
 - evaluation function, 33
 - maximization, 32
 - minimization, 32
 - multiple objective, 33
 - search space, 33
- particle swarm optimization, 1
 - acceleration coefficients, 44
 - binary PSO, 47
 - cooperative split PSO, 50
 - fitness function, 40
 - GBest, 42
 - inertia weight, 45
 - introduction, 39
 - iterations, 46
 - LBest, 43
 - mutating PSO, 48
 - particles, 39
 - position, 40
 - pseudocode, 41
 - swarm, 46
 - topology, 42
 - velocity, 40
 - velocity clamping, 45
 - Von Neumann, 43
- polynomial problem
 - nondeterministic, 33
- protein synthesis, 8
- SAGA, 29
- scoring matrix, *see* substitution matrix
- sequence
 - definition, 11
- sequence alignment
 - biological use, 20
 - definition, 12
 - general use, 19, 22
 - multiple, 13
 - pairwise, 13
 - process, 11
 - proteins, 17
 - representation, 12
 - sequence weights, 18
 - types, 12
 - validity, 12
- sequence mutation
 - deletions, 9
 - insertions, 9
- sequence relatedness, 10
- similarity approach, 14
 - costs, 15
 - example, 16
- simulated annealing, 28
- sub-sequence
 - definition, 11
- substitution matrix, 18
 - BLOSUM62, 19
 - Gonnet, 19
 - identity matrix, 18
 - PAM250, 19
- sum of pairs, 15
- T-COFFEE, 29