Sequential Optimization of Global Sequence Alignments Relative to Different Cost Functions

Thesis by
Enas Mohammad Odat, Master Degree in Computer Science

Submitted in Partial Fulfillment of the Requirements for the degree of Masters of Computer Science

King Abdullah University of Science and Technology
Mathmatical and Computer Sciences and Engineering Division
Computer System

Thuwal, Makkah Province, Kingdom of Saudi Arabia
May, 2011
The dissertation/thesis of Enas Mohammad Odat is approved by the examination committee.

Committee Chairperson:

Committee Co-Chair:

Committee Member:

King Abdullah University of Science and Technology
2011
ABSTRACT

Sequential Optimization of Global Sequence Alignments Relative to Different Cost Functions

Enas Mohammad Odat

The purpose of this dissertation is to present a methodology to model global sequence alignment problem as directed acyclic graph which helps to extract all possible optimal alignments. Moreover, a mechanism to sequentially optimize sequence alignment problem relative to different cost functions is suggested.

Sequence alignment is mostly important in computational biology. It is used to find evolutionary relationships between biological sequences. There are many algorithms that have been developed to solve this problem. The most famous algorithms are Needleman-Wunsch and Smith-Waterman that are based on dynamic programming. In dynamic programming, problem is divided into a set of overlapping subproblems and then the solution of each subproblem is found. Finally, the solutions to these subproblems are combined into a final solution.

In this thesis it has been proved that for two sequences of length $m$ and $n$ over a fixed alphabet, the suggested optimization procedure requires $O(mn)$ arithmetic operations per cost function on a single processor machine.
The algorithm has been simulated using C#.Net programming language and a number of experiments have been done to verify the proved statements. The results of these experiments show that the number of optimal alignments is reduced after each step of optimization. Furthermore, it has been verified that as the sequence length increased linearly then the number of optimal alignments increased exponentially which also depends on the cost function that is used. Finally, the number of executed operations increases polynomially as the sequence length increase linearly.
Acknowledgments

I would like to thank my supervisor Professor Mikhail Moshkov for giving me a chance to work with him in his research area and his guidance through this long journey. Furthermore, I would like to thank the collaborators Shahid Hussain and Igor Chikalov for their work and support.

I thank my parents for their continuous encouragement and support and my husband for his patience and bearing with me during my master degree.

Lastly, I would like to thank KAUST, Thuwal, Makkah Province, Saudi Arabia for providing support and resources for this research work.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>List of Illustrations</strong></td>
<td>9</td>
</tr>
<tr>
<td><strong>List of Tables</strong></td>
<td>12</td>
</tr>
<tr>
<td><strong>I Introduction</strong></td>
<td>13</td>
</tr>
<tr>
<td>I.1 Sequence Alignment</td>
<td>13</td>
</tr>
<tr>
<td>I.2 Biological Aspects of Sequence Alignment</td>
<td>17</td>
</tr>
<tr>
<td>I.3 Literature Review</td>
<td>19</td>
</tr>
<tr>
<td>I.3.1 Dynamic Programming Algorithms for Sequence Alignment</td>
<td>20</td>
</tr>
<tr>
<td>I.3.2 Needleman-Wunsch</td>
<td>20</td>
</tr>
<tr>
<td>I.3.3 Smith-Waterman</td>
<td>21</td>
</tr>
<tr>
<td>I.3.4 Sequence Alignment Softwares</td>
<td>22</td>
</tr>
<tr>
<td>I.3.5 BLAST Family</td>
<td>22</td>
</tr>
<tr>
<td>I.3.6 FASTA</td>
<td>24</td>
</tr>
<tr>
<td>I.3.7 CLUSTAL</td>
<td>24</td>
</tr>
<tr>
<td>I.3.8 MUMer</td>
<td>25</td>
</tr>
<tr>
<td>I.3.9 GLASS</td>
<td>25</td>
</tr>
<tr>
<td>I.3.10 AVID</td>
<td>26</td>
</tr>
<tr>
<td>I.3.11 LAGAN</td>
<td>26</td>
</tr>
<tr>
<td>I.4 Research Focus and Questions</td>
<td>27</td>
</tr>
</tbody>
</table>
## II Methodology of Sequential Optimization of Global Sequence Alignments Relative to Different Cost Functions

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>II.1 Preliminaries</td>
<td>30</td>
</tr>
<tr>
<td>II.2 Representation of the Set of Alignments</td>
<td>31</td>
</tr>
<tr>
<td>II.3 Procedure of Optimization</td>
<td>32</td>
</tr>
<tr>
<td>II.4 Consecutive Optimization</td>
<td>35</td>
</tr>
<tr>
<td>II.5 Computational Complexity of Optimization</td>
<td>35</td>
</tr>
<tr>
<td>II.6 Examples</td>
<td>36</td>
</tr>
<tr>
<td>II.6.1 Example One</td>
<td>36</td>
</tr>
<tr>
<td>II.6.2 Example Two</td>
<td>38</td>
</tr>
</tbody>
</table>

## III Software and Experiments

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>III.1 Software Overview</td>
<td>41</td>
</tr>
<tr>
<td>III.2 Experiments</td>
<td>45</td>
</tr>
<tr>
<td>III.2.1 Experiment One: Sequence Length vs. Number of Executed Operations</td>
<td>46</td>
</tr>
<tr>
<td>III.2.2 Experiment Two: Sequence Length vs. Number of Optimal Alignments</td>
<td>47</td>
</tr>
<tr>
<td>III.2.3 Experiment Three: Sequence Length vs. Number of Optimal Alignments</td>
<td>50</td>
</tr>
<tr>
<td>III.2.4 Experiment Four: Sequence Length vs. Number of Optimal Alignments</td>
<td>51</td>
</tr>
<tr>
<td>III.2.5 Experiment Five: Sequence Length vs. Number of Optimal Alignments</td>
<td>53</td>
</tr>
<tr>
<td>III.3 Discussion</td>
<td>55</td>
</tr>
</tbody>
</table>

## IV Conclusion and Recommendations

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appendices</td>
<td>60</td>
</tr>
</tbody>
</table>
List of Illustrations

I.1 Set of possible solutions after each step of optimization . . . . . . . . 28

II.1 Graph $G_\psi$ after optimizing $G$ relative to the cost function $\psi$. . . . . . . . 37
II.2 Graph $(G_\psi)_\phi$ after optimizing $G_\psi$ relative to the cost function $\phi$ . . . 38
II.3 Graph $G_\psi$ after optimizing $G$ relative to the cost function $\psi$. . . . . . . . 39
II.4 Graph $(G_\psi)_\phi$ after optimizing $G_\psi$ relative to the cost function $\phi$ . . . 40

III.1 Example in which all of the three flags are true . . . . . . . . . . . . . . . . . . . 42
III.2 Example in which IsMiddle flag equal false . . . . . . . . . . . . . . . . . . . . . 43
III.3 Results of experiment one. Sequence length vs. number of executed
   operations during the first step of optimization . . . . . . . . . . . . . . . . . . . . . 47
III.4 Results of experiment one. Sequence length vs. number of executed
   operations during the second step of optimization . . . . . . . . . . . . . . . . . . . . 48
III.5 Results of experiment one. Sequence length vs. average number of executed
   operations during the initialization step of dynamic programming
   algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48
III.6 Results of experiment one. Sequence length vs. average number of
   executed operations during the filling step of dynamic programming
   algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 49
| III.7 Results of experiment one. Sequence length vs. average number of executed operations during traceback step of dynamic programming | 49 |
| III.8 Results of experiment two. Sequence length vs. number of optimal alignments resulted from optimizing relative to the first cost function | 50 |
| III.9 Results of experiment two. Sequence length vs. number of optimal alignments resulted from optimizing relative to the second cost function | 51 |
| III.10 Results of experiment three. Sequence length vs. number of optimal alignments resulted from optimizing relative to the first cost function | 52 |
| III.11 Results of experiment three. Sequence length vs. number of optimal alignments resulted from optimizing relative to the second cost function | 52 |
| III.12 Results of experiment four. Sequence length vs. number of optimal alignments resulted from optimizing relative to the first cost function | 53 |
| III.13 Results of experiment four. Sequence Length vs. Number of Optimal Alignments resulted from optimizing relative to the second cost function | 54 |
| III.14 Results of experiment five. Sequence length vs. number of optimal alignments resulted from optimizing relative to the first cost function | 54 |
| III.15 Results of experiment five. Sequence length vs. number of optimal alignments resulted from optimizing relative to the second cost function | 55 |
List of Tables

I.1 Initialization step of Needleman-Wunsch for sequences AATTGA, ATG: 20

III.1 Matrix corresponding to three structure: 44
Chapter I

Introduction

I.1 Sequence Alignment

The sequence alignment problem is a generalization of a classical problem known as the edit distance problem, see Levenshtein \[\text{[1]}\], which measures the difference between two sequences. The generalized version known as the sequence alignment problem lies at the core of bioinformatics. The solution to this problem helps identify several biological similarities present in long sequences of DNA/RNA and proteins.

Sequence alignment (Sequence matching, String matching, and String comparison) is the arrangement of two or more sequences to find the similarities between them. Arrangement is done by aligning the sequences on top of each other in matrix like fashion such that each sequence represent a row in that matrix. The aligned positions in both sequences are compared against each other to find the places of similarities and differences. A sequence or string \( X = x_1, x_2, ..., x_n \) is a set of characters or words over some finite alphabet \( \Sigma \), for example \( \Sigma \) could be the set of Arabic alphabet, English alphabet, DNA alphabet = \{G, C, T, A\}, Proteins =\{A, R, N, D, C, Q, E, G, H, I, L, K, M, F, P, S, T, W, Y, V\}, RNA = \{A, C, G, U\}, etc.

For example, to align two sequences or strings \( S_1 = \text{FUNNY}, S_2 = \text{HONEY} \)
over English alphabet, there are three possible alignments between them listed as following:

\[
\begin{array}{cccc}
F & U & N & N & Y \\
H & O & N & E & Y
\end{array}
\]

In the previous alignments three events happened when aligning two positions in both sequences. First, two positions are equal or they match each others, for example when aligning N from the first sequence with N from the second sequence. Second, two positions are different or they mismatch each others, for example when aligning N from first sequence with E from second sequence. Finally, one of the positions is character and the other one is dash or gap which is inserted to get better alignment, for example, when aligning U from first sequence with from the second sequence. The question that arises here: What is the best alignment among the three listed above?

The answer to this question is related to several concepts which are scoring schemes and optimal alignment. Scoring scheme is the set of cost functions that is used to measure how good the sequence alignment is. These cost functions include the cost of gaps, cost of mismatch and the cost of match. For example, if the gap cost = -1, mismatch cost = -2, and match cost = 0 and we want to apply them on the alignments of the previous example then the total cost of the first alignment is \((-2) + (-2) + (0) + (-2) + (0) = (-6)\), for the second one the cost is \((-1) + (-2) + (-1) + (0) + (-2) + (0) = (-6)\), and for the final one the cost is \((-2) + (-1) + (-1) + (-1) + (0) + (-2) + (-1) = (-8)\). Now, if the best alignment is defined as alignment with minimum cost then the best alignment will be the third one but if the best alignment is defined as alignment with maximum cost then best alignment would be either alignment one or two. The best alignment is also known as the optimal alignment.

Sequence alignment can be classified into several categories using different criteria. Depending on the number of sequences to be aligned there are two types of sequence
alignment: pairwise alignment and multiple alignment. In pairwise alignment, the number of sequences to be aligned is two whereas in multiple sequences alignment the number of sequences to be aligned is more than two. Practically, pairwise alignment is useful when there are small numbers of sequences to compare or to align while it becomes impractical when the number of sequences is large, for example, when searching databases against query sequence. In the latest case the multiple alignments becomes more useful.

Sequence alignment can be further classified depending on the length of the alignment and in this case there are two types of sequence alignment: Global alignment, and Local alignment. Global alignment works on the entire length of both sequences, end to end, even if they are not entirely similar and this is done by inserting gaps in sequences such that the final length, after alignment, of both sequences is the same. In Local alignment, motif finding or exact partial matching, parts of sequences are aligned. Local alignment is useful when the aligned sequences are not highly similar or have partial similarity between them; on the other hand, global alignment is useful when the two sequences are of the same length or nearly of the same length and when they are highly similar. The examples below show global alignment and local alignment of sequences AATTGA, ATG on alphabet $\Sigma = \{A,T,G,C\}$. The algorithms for finding such results will be discussed in the next subsections.

$$\text{Globalalignment} = \begin{array}{c} \text{T}T\ldots\text{CCAC} \\ \text{ATGA}\ldots\text{A}\ldots \end{array}$$

$$\text{Localalignment} = \begin{array}{c} \text{AATTGA} \\ \text{AT}\ldots\text{G}\ldots \end{array}$$

The naive method to perform sequence alignment is to compare sequences by hand to find different possible alignments. Although this method can be used for sequences of small lengths, it is impractical for long sequences thus many algorithms
have been developed to solve this problem. These algorithms can be classified into two categories: Dynamic programming methods and Heuristic methods.

Heuristic methods are those methods that approximate problem solution rather than giving exact solution hence they didn’t guarantee finding optimal alignment. They could be approximation for probabilistic methods like hidden Markov model or they could approximate dynamic programming methods. Since they are only approximation for dynamic programming, they are less sensitive to the solution but faster when compared with dynamic programming algorithms. Due to their relative speed, they are used in database searching where there could be multiple hits to a query sequence and also they are used for multiple sequence alignment. Many tools use this methods including FATSA [2] and BLAST [3]. Heuristic methods could be used for global and local sequence alignment. For local alignment they start by finding substrings of exact matches between both sequences (also called seeds) and then they search parts of both sequences around the seeds in a way to maximize the alignment score. All positions of seed in both sequences could be similar and in this case the seed is exact match like what is used in BLASTN [3], and some approaches allow for one mismatch to occur in any place of the seed in both sequences like BLAT [4], and others allow multiple mismatches to occur in pre specified positions of the seed which is called spaced seeds like PatternHunter [5], see [6] for survey about seed types. Global heuristic methods are somehow similar to local heuristic methods. For further details about these methods, see survey [7].

Dynamic programming is a method for solving problems by dividing them into a set of overlapping subproblems, solves each one of these subproblems, and finally to combine the solution of them to get the solution of the whole problem. Dynamic programming is different from recursion in that unnecessarily calculations are avoided by storing the intermediary results and retrieving them when required. It was originally used in 1940’s by Richard Bellman. Instead of exponential time for solving some
problems, dynamic programming take time in order of $O(n^3)$ or $O(n^2)$. So the algorithm works by taking a problem and divide it into smaller subproblems to find the optimal solution for them and finally to combine them into a whole optimal solution. It is memory inefficient technique because in this method the intermediate results are stored into intermediate table instead of recalculating them again. The most famous algorithms that are based on dynamic programming are Needleman-Wunsch algorithm for global alignment [8], Smith-Waterman algorithm for local alignment [9]. For further details about the origins of dynamic programming see [10]. Dynamic programming algorithms ensure finding the optimum alignment depending on a given score function in time complexity of order $O(mn)$ where $m, n$ are sequences length, and they require $O(nm)$ space so they are impractical for searching large databases.

For pairwise sequence alignment, dynamic programming is used along with other methods including words methods and dot matrix plot. In dot matrix plot method two sequences are aligned by placing one sequence at the top row and the other one in the first column of matrix. A dot is placed wherever two positions are matched. For multiple sequence alignment, there are multiple methods that could be used such as dynamic programming, hidden Markov model, and genetic algorithm and simulated annealing, iterative methods, and progressive methods. The importance of sequence alignment arises from its several applications in various fields of science. The next section explains some aspects of sequence alignment in computational biology.

### I.2 Biological Aspects of Sequence Alignment

Sequence alignment is mostly important in biology where the sequences could be DNA, RNA, or Protein. This importance arise from several facts [11]. First, when discovering new genomes the first step is to identify its function and this is done by finding similar genomes; i.e., similarities indicate similar functionality. Second,
it’s used in disease analysis, for example it was used to research multiple sclerosis in which the immune system attacks nerve cells. Similar sequences to myelin sheath proteins were sequenced and search for similar sequences in database was done and after that it was discovered that myelin sheath protein was treated as bacterial and viral proteins due to similarities between them. In biological sequence alignment gaps means insertion in one sequences and deletion in the sequence that contains the gaps. Third, Sequence alignment is important in computational biology in order to find the evolutionary similarities between sequences or to find homologous sequences that have common ancestor. If two positions match each other then there is probability that they share same ancestor and both positions are conserved in the subsequent generations. If two positions mismatch each other then there is probability of replacement mutation (substitution which means that one of nucleotide has been changed to another), and if gap appeared in any of the aligned positions (also called indel) then there is possibility of deletion mutation or insertion mutation (insert new DNA).

The question that arises at this point is about the set of cost functions or scoring schemes to be used, in other words what is the penalty or the match cost, the mismatch cost and the gap cost that are biologically meaningful? Mathematically, there is no limitation; i.e., any cost functions including positive and negative numbers could be used but in biology the question is more than it’s in math, and actually the answer depends on the type of the sequences to be aligned. In biology, scoring scheme consist of scoring matrix (match cost and mismatch cost) and the gap penalty. The are no specific criteria that help to choose a good scoring scheme. There are two types of gap cost that can be used:

1. Constant gap penalty: where the gap penalty is fixed regardless of the position of this gap in the sequences.

2. Affine gap penalty: in this case we have gap opening penalty for the first gap and gap extension penalty for the next gaps. So this gap penalty can be calculated
using the formula: $O_g + E_g \times L$ where $L$ is the length of the gap, $O_g$ is the gap opening penalty, $E_g$ is the gap extension penalty.

In general, depending on the type of biological sequences, there are two possibilities of scoring schemes:

1. In the alignment of DNA, RNA sequences one could use positive number for matching, negative number for mismatching and another negative number or any of the previously stated functions (Affine gap penalty, Linear gap penalty) for the gap cost.

2. For alignment of proteins, substitution matrix is used for match cost and mismatch cost and in this case we take the biological knowledge about the sequences being aligned. For gap cost, one can use a number or any function (Affine gap penalty, Linear gap penalty). Substitution matrix implements alphabet-weight scoring scheme. Substitution matrices includes PAM \[12\] and BLOSUM \[13\] and they are matrices that contain the probability that one letter could be mutated to other letters.

I.3 Literature Review

Up to the author’s knowledge, there are no algorithms that optimize sequence alignment problem sequentially as described in this work. This section will provide literature review about dynamic programming algorithms as well as a survey of biological software for sequence alignment. The most famous dynamic programming algorithms for sequence alignment are Needleman-Wunsch for global sequence alignment and Smith-Waterman for local sequence alignment.
I.3.1 Dynamic Programming Algorithms for Sequence Alignment

This research presents an extension to Needleman-Wunsch algorithm for global sequence alignment. A next step in this direction will be to study an extension to local sequence alignment based on Smith-Waterman algorithm. The main ideas of both algorithms are explained below. Given two sequences of length $m, n$ the dynamic programming algorithms find the set of optimal alignment between two sequences in $O(mn)$ time complexity.

I.3.2 Needleman-Wunsch

Needleman-Wunsch, 1969, [8] is a dynamic programming algorithm for global sequence alignment. In this method, one of the sequences is positioned at the top of a matrix and the other sequence is positioned at the left most of that matrix, see Table I.3.2. It has three different stages: initialization, matrix filling and traceback. In initialization step the matrix cell $(0, 0)$ is set to zero, positions at top row indexed by $i$ and left most column positions indexed by $j$ are filled with cost of gaps of length $i$ and $j$. For example, suppose that the mismatch cost = 2, match cost = 0, gap cost = 1 then the matrix cell at position $(1,4)$ is filled by computing the following:

$$\text{gap cost of cell (1,1) + gap cost of cell (1,2) + gap cost of cell (1,3) + gap cost of cell (1,4) = 1 + 1 + 1 + 1 = 4.}$$

Cell at position $(4, 1)$ is filled as following: gap cost of cell $(1,1) + \text{gap cost of cell (2,1) + gap cost of cell (3,1) + gap cost of cell (4,1) = 1 + 1 + 1 + 1 = 4.}$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>A</th>
<th>T</th>
<th>T</th>
<th>G</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table I.1: Initialization step of Needleman-Wunsch for sequences AATTGA, ATG
In matrix filling step the matrix is filled cell by cell starting from second row and second column using the following function for finding optimal cost at each cell:

\[
Cost(x) = \min \begin{cases} 
Cost(i - 1, j - 1) + S(x_i, x_j) \\
\text{cost}(i - 1, j) + w(x_i, \omega) \\
\text{cost}(i, j - 1) + w(\omega, y_j)
\end{cases}
\]

Where \(S(x_i, x_j)\) is the substitution score for letters \(i\) and \(j\), \(w(\omega, y_j), w(x_i, \omega)\) are gap cost.

In traceback stage, the matrix is traced from the last cell (at the last column and last row) following the path (either diagonally, up, left) which contains cells with maximum cost. In more details, suppose that we are at cell \((i, j)\) then in traceback step we have to check the direct neighbor cells (predecessors) in the left (corresponds to gap in sequence two), diagonal (corresponds to match/mismatch in both sequences) and upper cell (corresponds to gap in second sequence) to find the cell with maximum cost. If more than one cell has maximum cost then there are multiple optimal alignments.

I.3.3 Smith-Waterman

Smith-Waterman [9] is a dynamic programming algorithm for local sequence alignment. It differs from Needleman-Wunsch in that it requires mismatch cost to be negative and if the computation of optimal cost at any cell resulted in negative value then it is set to zero. The algorithm starts with initialization step where the first row and first column are filled out with zeros.

Next step is the filling step in which each cell is filled by optimal cost that is computed using the following formula:
\[ F(x) = \min \begin{cases} 
\text{cost}(i - 1, j - 1) + S(x_i, y_j), & \text{match/mismatch} \\
\text{cost}(i - 1, j) + w(x_i, \omega), & \text{insertion} \\
\text{cost}(i, j - 1) + w(\omega, y_j), & \text{deletion} 
\end{cases} \]

where \( S(x_i, y_j) \) is the substitution score for match/mismatch of letters \( x_i, y_j \), \( w(x_i, \omega) \) and \( w(\omega, y_j) \) are the gap costs. Again in this step if the optimum cost resulted in negative value then it is set to zero.

Finally, the traceback step will start to find the optimal alignments.

### I.3.4 Sequence Alignment Softwares

Sequence alignment is mostly important in biology where sequences of Proteins, DNA, or RNA are compared against each other to find functional and structural relationship between them. The softwares for sequence alignment are divided into two types: sequences alignment, and database searching. The system can be viewed as consisting of input which is for example DNA sequences (Amino acid, Nucleotide), output which is two aligned sequences and the associated match score with them and the problem is to find the optimal alignment among others. The methods that are described in this section are heuristic methods.

### I.3.5 BLAST Family

BLAST (Basic local alignment search tool) is the most widely used sequence alignment tool which has been designed by Eugene Myers, Stephen Altschul, Warren Gish, David J. Lipman, and Webb Miller, 1990. BLAST is a heuristic method, based on Smith-Waterman algorithm for local alignment that does approximate sequence matching using hashing method. This software allows the user to search a query sequence in database to find similar sequences. For example, suppose that biologist
has discovered new sequence for mouse, and then they will search using BLAST to check similar sequences in human genome which reflect similar function. What is important about this software is the speed as opposed to other algorithms that rely on Smith-Waterman (dynamic programming) which makes it practical for huge database of genomes (in terms of time). The accuracy of this software is reasonable but doesn’t guarantee finding optimal alignment nor finding all hits in the database.

The algorithm has three phases: compiling list of highly scoring words, scanning the database for hits, and extending the hits. Compiling list of highly scoring words means that the query sequence is divided into set of words of length $k$ (or called suffix trees) where the value of $k$ is 3 for proteins and 11 for DNA. For example, suppose that we have the following query sequence: ATGTAAACG then setting $k = 3$ then we will have: ATG, TGT, GTA, TAA, AAA, AAC, ACG. After that the generated words are compared with all $k$ words in the database, there are $20^3$ for $k = 3$ and $4^{11}$ for DNA/RNA, and the scoring values are found using scoring scheme. The score scheme that is used is 5 for match and -4 for mismatch or 2 for match and -3 for mismatch. Then the listed possible words are reduced by using threshold $T$ such that all scores above threshold are taken into consideration and words that are below are not considered. For the second phase, they employed two approaches: first, suppose that the word length = 4 and each one is mapped into number between 1 to $20^4$, so the word can be used as index in array of size $20^4$ with the $i$th entry contains all occurrences of the word in the sequence. The second approach is by using finite automata. Then the database is searched to find exact match. In other words, the algorithm identify maximum identical length parts of both sequences with highest score or called MSP (maximal segment pair) which could be of any length and above certain threshold, computed using dynamic programming methods.

Finally, the BLAST extends the alignment between the query sequence and the sequence in database starting from the point of exact match and from left and right.
The score value between the two sequences is computed until it begins to decrease. Then depending on certain threshold, the alignments of certain HSP (High-Scoring Segment Pairs) are considered. The scoring function that is used here is for amino acid, its PAM-120 matrix and 5 for match and -4 for mismatch for proteins. Note that a small value of threshold will result in multiple hits and hence the execution time of the algorithm. The HSP’s are compared again with threshold $T$ and the ones with highest scores are taken into consideration.

BLAST family includes other methods like: BLASTZ [14], BLAT [4], Gapped BLAST [3].

I.3.6 FASTA

FATSA is local, heuristic, and hashing search algorithm based on Smith-Waterman algorithm that is used to search database of DNA and proteins. The algorithm starts by finding matches between two sequences (the query sequence and the database sequence) using lookup table or hash table. The alignment between two sequences is represented using dot matrix where diagonals represent match between them. Next, the matches are scored using PAM 250 and choose the ten longest local (partial or conserved) that have score value greater than threshold (short diagonals are removed) are chosen. Then, long consecutive matches selected from previous step are joined using gaps. Finally, Smith-Waterman algorithm is used to do final alignment [2].

I.3.7 CLUSTAL

CLUSTAL is a heuristic method used for global multiple sequence alignment. The algorithm starts by calculating all pairwise similarities between sequences. Next, it does clustering analysis by establishing dendrogram using the similarity matrix in stage one. Finally, it does multiple sequence alignment progressively by taking two most similar sequences each time from the remaining set and doing pairwise alignment
between them [15].

### I.3.8 MUMer

MUMer is a pairwise alignment for global genome comparison. It uses three ideas which are: suffix trees, longest increasing subsequence and Smith-Waterman algorithm for local alignment. The algorithm starts by identifying maximal unique matching units (MUM) by using suffix trees (MUM is the maximal substring that appears exactly once in both sequences). First, it constructs the suffix trees in which each path from root to node is MUM. After that, the longest set of MUMs whom are in same order in both sequences are found. Then, local gaps are added to complete the alignment. So the idea is to find MUMs and to build global alignment around them. After finding all MUMs, they are sorted and the longest set of matches is found (that occur in both sequences in order). The algorithm is capable of identifying SNP (single mutation surrounded by exact matches), insert detection (where insert is a sequence that occurs in one genome but not in the others), highly polymorphic regions detection (regions that have many mutations in short regions), and repeat sequences [16].

### I.3.9 GLASS

GLASS is a pairwise global alignment algorithm. The algorithm starts by finding all $k$-mers (segments that are exactly match and whose surrounding areas have highly similar score) between two strings. Then these $k$-mers are converted into special characters. After that, dynamic programming algorithm is used to align pairs of $k$-mers and matches that have scores above certain threshold are selected. The process repeated for regions between matched $k$-mers but at this time smaller value for $k$ is used [17].
I.3.10 AVID

AVID is a global sequence alignment algorithm. The algorithm finds the maximal matches between the two sequences using suffix trees after concatenating the two strings with one separating character which is $N$, so the problem is converted into finding maximal repeated strings (the substrings of this string are not repeated in the sequences) that cross the border between two sequences. Next step is to find all anchors between two sequences which are non overlapping non crossing matches and this is done using Smith-Waterman algorithm (with mismatch penalty equal to infinity and gap penalty of one) and it’s similar to GLASS method in anchor selection. Finally, the process is repeated for the regions between the anchors. The process terminates when there are no significant matches or there are no remaining bases to be aligned [18].

I.3.11 LAGAN

LAGAN is an algorithm for global alignment. It performs alignment in three main steps. First, it finds all local alignments between two sequences using CHAOS algorithm [19]. Second, it construct global map by chaining an ordered subsets of local alignment using dynamic programming. Finally, it restricts search area and applies Needleman-Wunsch to that area [20].

The literature review of existing methods for sequence alignment shows that this problem has great interest and it is a part of other heuristic methods. Dynamic programming algorithms are the most sensitive methods that guarantee finding optimal alignments in the expense of execution time and space used. All methods described above are less sensitive (they don’t guarantee finding optimal alignments) than dynamic programming methods but they are faster. The fastest software and most sensitive method is BLAST.
I.4 Research Focus and Questions

Problem of sequential optimization of sequence alignment is an extension of results for decision trees [21, 22, 23, 24] obtained in the framework of rough set theory [25, 26], so this thesis can be considered as a step to one more area of applications of rough set theory.

The main goal of this dissertation is to study global sequence alignment problem using an extension of dynamic programming. Also, a method to model the global sequence alignment problem using a directed acyclic graph is provided, which helps to extract all optimal solutions specific to a certain cost function. Furthermore, a sequence of different cost functions to optimize sequence alignment is applied. For example, the first step of optimization works toward minimizing the number of indels (the number of gaps in an alignment) and at the second step of optimization the works continue to minimize of the number of mismatches (the number of matches of different letters in an alignment). Chapter 2 provides detailed analysis of the problem.

To simulate the suggested methodology, it was necessary to implement software for that. Chapter 3 of this thesis presents the results of the software implemented using C#.Net. The specific goals of this thesis are:

1. To test the applicability of the methodology. This is done by measuring the time complexity of the algorithm using different sequence length. The idea is that if this time is acceptable for long sequences then this is could be indication on the possibility of application of this methodology on real life problems.

2. To test the effect of different cost functions on the algorithm behavior especially the resulted number of optimal solutions after applying each cost function.

3. To study how the number of optimal solutions grows as the sequence length grows linearly.
Sequential optimization of sequence alignments reduces the search space of possible optimal alignments at each step of optimization with possibility of inserting and using any cost function or minimization criteria. Let we have two cost functions. After the first step of optimization we obtain the whole set of alignments which are optimal relative to the first cost function (which is the set $A$) and after the second step of optimization we will find all optimal alignments relative to the second cost function among alignments from $A$ (which is $B$). See Figure I.4.

Up to this point, no specific real life application to this problem has been found. The software has been tested on randomly generated biological sequences. Since this is verifiability step, the motivation to find a specific biological application to this problem is one of the future agenda.
Chapter II

Methodology of Sequential Optimization of Global Sequence Alignments Relative to Different Cost Functions

In this chapter, an analysis of sequential optimization for sequence alignment problem is provided. Section 1 provides preliminary concepts to understand and model the sequence alignment problem. Section 2 discusses the representation of sets of alignments in terms of subproblems and consequently in a directed acyclic graph. Section 3 presents the procedure of optimization. After that, the optimization procedure for application of different cost functions in sequence is extended in Section 4. Section 5 provides the computational complexity of the procedure followed by examples in Section 6.
II.1 Preliminaries

Let $\Sigma$ be a finite alphabet, $X = x_1x_2 \ldots x_m$ and $Y = y_1y_2 \ldots y_n$ be sequences (words) over this alphabet.

We consider the problem of sequence alignment for $X$ and $Y$, and denote this problem as $S(m,n)$. Furthermore, we consider subproblems of the initial problem. For $0 \leq i \leq m$ and $0 \leq j \leq n$ we denote by $S(i,j)$ the problem of sequence alignment for the prefixes $x_1 \ldots x_i$ and $y_i \ldots y_j$. If $i = 0$ or $j = 0$ we have empty prefixes, denoted as $\lambda$.

We describe inductively the set $P(i,j)$ of alignments for $S(i,j)$. Each alignment for $S(i,j)$ is a pair of sequences of equal length obtained from prefixes $x_1 \ldots x_i$ and $y_1 \ldots y_j$ by insertion of gaps “_” such that there is no gap in the same position in both sequences. We have $P(0,0) = \{(\lambda, \lambda)\}$ where $\lambda$ is the empty sequence, $P(0,j) = \{(\ldots, y_1 \ldots y_j)\}$, and $P(i,0) = \{(x_1 \ldots x_i, \ldots)\}$ for any $0 \leq i \leq m$ and $0 \leq j \leq n$. Furthermore, for $0 < i \leq m$ and $0 < j \leq n$,

$$P(i,j) = \{(\alpha x_i, \beta \omega) : (\alpha, \beta) \in P(i-1,j)\}$$
$$\cup \{(\alpha \omega, \beta y_j) : (\alpha, \beta) \in P(i,j-1)\}$$
$$\cup \{(\alpha x_i, \beta y_j) : (\alpha, \beta) \in P(i-1,j-1)\}.$$

We consider now the notion of cost function $\psi$. We fix integral values $w(x_i, y_i)$, $w(x_i, \omega)$, and $w(\omega, y_j)$ for $1 \leq i \leq m$ and $1 \leq j \leq n$. We set $w(\lambda, \lambda) = 0$. Then for any alignment $(\alpha_1 \ldots \alpha_t, \beta_1 \ldots \beta_t)$, we have

$$\psi(\alpha_1 \ldots \alpha_t, \beta_1 \ldots \beta_t) = \sum_{i=1}^{t} w(\alpha_i, \beta_i).$$
We can also define this function inductively: \( \psi(\lambda, \lambda) = 0 \) and
\[
\psi(\alpha_1 \ldots \alpha_t, \beta_1 \ldots \beta_t) = \psi(\alpha_1 \ldots \alpha_{t-1}, \beta_1 \ldots \beta_{t-1}) + w(\alpha_t, \beta_t).
\]

**II.2 Representation of the Set of Alignments**

We describe a directed acyclic graph (DAG) \( G_0 \) which allows us to represent all alignments \( P(i, j) \) for each subproblem \( S(i, j), 0 \leq i \leq m, 0 \leq j \leq n. \) The set of vertices of this graph coincides with the set
\[
\{S(i, j) : 0 \leq i \leq m, 0 \leq j \leq n\}.
\]

If \( i = 0 \) or \( j = 0 \) then \( S(i, j) \) has no outgoing edges. Let \( i > 0 \) and \( j > 0 \), then \( S(i, j) \) has exactly three outgoing edges. These edges end in \( S(i-1, j), S(i, j-1), \) and \( S(i-1, j-1) \) and labeled with indexes 1, 2, and 3 respectively.

Let \( G \) be a subgraph of \( G_0 \) which is obtained from \( G_0 \) by removal of some edges such that for each vertex \( S(i, j) \) with \( i > 0 \) and \( j > 0 \) at least one edge outgoing from \( S(i, j) \) remains intact. Such a subgraph will be called proper subgraph of \( G_0 \).

Now for each vertex \( S(i, j) \) we define, by induction, the set \( P_G(i, j) \) of alignments corresponding to \( S(i, j) \) in \( G \). We have \( P_G(0, 0) = \{ (\lambda, \lambda) \} \), \( P_G(0, j) = \{ (\omega \ldots \omega, y_1 \ldots y_j) \} \) and \( P_G(i, 0) = \{ (x_1 \ldots x_i, \omega \ldots \omega) \} \), \( 1 \leq i \leq m, 1 \leq j \leq n. \)

For \( i > 0 \) and \( j > 0 \), let \( K_G(i, j) \) be the set of indexes of remaining edges outgoing from \( S(i, j) \) in \( G \). Then
\[
P_G(i, j) = \bigcup_{k \in K_G(i, j)} P^{(k)}_G(i, j),
\]
where

\[
P_G^{(1)}(i, j) = \{(\alpha x_i, \beta) : (\alpha, \beta) \in P_G(i-1, j)\},
\]
\[
P_G^{(2)}(i, j) = \{(\alpha, \beta y_j) : (\alpha, \beta) \in P_G(i, j-1)\},
\]
\[
P_G^{(3)}(i, j) = \{(\alpha x_i, \beta y_j) : (\alpha, \beta) \in P_G(i-1, j-1)\}.
\]

One can show that \( P_G(0, i, j) = P(i, j), 0 \leq i \leq m, 0 \leq j \leq n. \)

\section*{II.3 Procedure of Optimization}

Let \( \psi \) be a cost function. We consider the procedure of optimization of alignments corresponding to vertices of \( G \) relative to \( \psi \). For each vertex \( S(i, j) \) of the graph \( G \) we mark this vertex by a number \( \psi_{i,j} \) (later we will prove that \( \psi_{i,j} \) is the minimum value of \( \psi \) on \( P_G(i, j) \)) and may remove some edges outgoing from \( S(i, j) \), if \( i > 0 \) and \( j > 0 \).

We set \( \psi_{0,0} = 0, \psi_{i,0} = w(x_1, \omega) + \cdots + w(x_i, \omega), \) and \( \psi_{0,j} = w(\omega, y_1) + \cdots + w(\omega, y_j) \) for \( 1 \leq i \leq m \) and \( 1 \leq j \leq n. \)

Let \( 1 \leq i \leq m \) and \( 1 \leq j \leq n. \) Then

\[
\psi_{i,j} = \min_{k \in K_G(i, j)} \psi_{i,j}^{(k)},
\]

where

\[
\psi_{i,j}^{(1)} = \psi_{i-1,j} + w(x_i, \omega),
\]
\[
\psi_{i,j}^{(2)} = \psi_{i,j-1} + w(\omega, y_j),
\]
\[
\psi_{i,j}^{(3)} = \psi_{i-1,j-1} + w(x_i, y_j).
\]

Now we remove from \( G \) each edge with index \( k \in K_G(i, j) \) outgoing from \( S(i, j) \)
such that
\[ \psi^{(k)}_{i,j} > \psi_{i,j}. \]

We denote by \( G_\psi \) the resulting subgraph of \( G \). It is clear that for each vertex \( S(i,j), i \geq 1, j \geq 1 \), at least one edge outgoing from \( S(i,j) \) is left intact. So \( G_\psi \) is a proper subgraph of \( G_0 \).

For \( 0 \leq i \leq m \) and \( 0 \leq j \leq n \), we denote by \( P_{G_\psi}^{opt}(i,j) \) the set of all alignments from \( P_G(i,j) \) which have minimum cost relative to \( \psi \) among alignments from \( P_G(i,j) \).

The following theorem summarizes the procedure of optimization.

**Theorem 1.** Let \( G \) be a proper subgraph of \( G_0 \), and \( \psi \) be a cost function. Then for any \( i \) and \( j \) such that \( 0 \leq i \leq m \) and \( 0 \leq j \leq n \), \( \psi_{i,j} \) is the minimum cost of an alignment from \( P_G(i,j) \), and the set \( P_{G_\psi}(i,j) \) coincides with the set of all alignments \( (\alpha, \beta) \in P_G(i,j) \) for which \( \psi(\alpha, \beta) = \psi_{i,j} \), i.e., \( P_{G_\psi}(i,j) = P_{G_\psi}^{opt}(i,j) \).

**Proof.** The proof is by induction on \( i+j \). One can show that the considered statement is true if \( i = 0 \) or \( j = 0 \) (in this case \( P_G(i,j) \) contains exactly one alignment, and this alignment has the minimum cost relative to \( \psi \) among alignments from \( P_G(i,j) \)). Hence the given statement is true if \( i+j \leq 1 \). Let us assume that for some \( t \geq 1 \), the considered statement is true for each \( i \) and \( j \) such that \( i+j \leq t \). Let \( i+j = t+1 \).

If \( i = 0 \) or \( j = 0 \) then the statement holds. Let \( i > 0 \) and \( j > 0 \). We know that

\[ P_G(i,j) = \bigcup_{k \in K_G(i,j)} P^{(k)}_G(i,j). \] (II.1)

Since \( (i-1)+j \leq t \), \( i+(j-1) \leq t \) and \( (i-1)+(j-1) < t \), we have, by induction hypothesis, that \( \psi_{i-1,j}, \psi_{i,j-1}, \) and \( \psi_{i-1,j-1} \) are the minimum costs of alignments from \( P_G(i-1,j), P_G(i,j-1), \) and \( P_G(i-1,j-1), \) respectively. From here it follows that \( \psi^{(1)}_{i,j} = \psi_{i-1,j} + w(x_i, \omega) \) is the minimum cost of an alignment from \( P^{(1)}_G(i,j) \) and similarly \( \psi^{(2)}_{i,j} = \psi_{i,j-1} + w(\omega, y_j) \) and \( \psi^{(3)}_{i,j} = \psi_{i-1,j-1} + w(x_i, y_j) \) are respectively the
minimum costs for alignments from $P_G^{(2)}(i, j)$ and $P_G^{(3)}(i, j)$. From here and \[\boxcheck{II.1}\] it follows that $\psi_{i,j}$ is the minimum cost of an alignment from $P_G(i, j)$.

After applying the procedure of optimization to $G$ relative to $\psi$, we have

$$P_{G_\psi}(i, j) = \bigcup_{k \in K_{G_\psi}(i,j)} P_{G_\psi}^{(k)}(i, j),$$

(\[\boxcheck{II.2}\])

where

$$P_{G_\psi}^{(1)}(i, j) = \{(\alpha x_i, \beta y_j) : (\alpha, \beta) \in P_{G_\psi}(i - 1, j)\},$$

$$P_{G_\psi}^{(2)}(i, j) = \{(\alpha x_i, \beta y_j) : (\alpha, \beta) \in P_{G_\psi}(i, j - 1)\},$$

$$P_{G_\psi}^{(3)}(i, j) = \{(\alpha x_i, \beta y_j) : (\alpha, \beta) \in P_{G_\psi}(i - 1, j - 1)\}$$

and

$$K_{G_\psi}(i, j) = \{k \in K_G(i, j) : \psi_{i,j}^{(k)} = \psi_{i,j}\}.$$  

By induction hypothesis, $P_{G_\psi}(i - 1, j)$, $P_{G_\psi}(i, j - 1)$, and $P_{G_\psi}(i - 1, j - 1)$ respectively coincide with the sets of alignments in $P_G(i - 1, j)$, $P_G(i, j - 1)$, and $P_G(i - 1, j - 1)$ for which the respective costs are $\psi_{i-1,j}$, $\psi_{i,j-1}$, and $\psi_{i-1,j-1}$, i.e., $P_{G_\psi}(i - 1, j) = P_{G_\psi}^{opt}(i - 1, j)$, $P_{G_\psi}(i, j - 1) = P_{G_\psi}^{opt}(i, j - 1)$, and $P_{G_\psi}(i - 1, j - 1) = P_{G_\psi}^{opt}(i - 1, j - 1)$.

From here and \[\boxcheck{II.2}\] it follows that the cost of each alignment from $P_{G_\psi}(i, j)$ is equal to $\psi_{i,j}$ and $P_{G_\psi}(i, j) \subseteq P_{G_\psi}^{opt}(i, j)$. Let us show that $P_{G_\psi}^{opt}(i, j) \subseteq P_{G_\psi}(i, j)$.

Let $(\alpha, \beta) \in P_G(i, j)$ and $\psi(\alpha, \beta) = \psi_{i,j}$. From \[\boxcheck{II.1}\] it follows that $(\alpha, \beta) \in P_G^{(k)}(i, j)$ for some $k \in K_G(i, j)$. Let $k = 3$. Then $(\alpha, \beta) = (\alpha' x_i, \beta' y_j)$, where $(\alpha', \beta') \in P_G(i - 1, j - 1)$, and $\psi(\alpha, \beta) = \psi(\alpha', \beta') + w(x_i, y_j)$. Since $\psi(\alpha, \beta) = \psi_{i,j}$, we have $\psi(\alpha', \beta') = \psi_{i,j} - w(x_i, y_j)$. It is clear that $\psi_{i-1,j-1} \leq \psi(\alpha', \beta')$. Therefore $\psi_{i,j}^{(3)} = \psi_{i-1,j-1} + w(x_i, y_j) \leq \psi_{i,j} - w(x_i, y_j) + w(x_i, y_j) = \psi_{i,j}$. Hence $3 \in K_{G_\psi}(i, j)$ and $P_{G_\psi}^{(3)}(i, j) \subseteq P_{G_\psi}(i, j)$.
Let us assume that $\psi_{i-1,j-1} < \psi(\alpha', \beta')$. Then $\psi_{i,j}^{(3)} < \psi_{i,j}$ which is impossible. Therefore $\psi_{i-1,j-1} = \psi(\alpha', \beta')$, $(\alpha', \beta') \in P_{G_{\psi}}(i - 1, j - 1)$, $(\alpha, \beta) \in P_{G_{\psi}}^{(3)}(i, j)$ and $(\alpha, \beta) \in P_{G_{\psi}}(i, j)$. The cases when $k = 1$ and $k = 2$ can be considered in the same way. Therefore $P_{G_{\psi}}^{opt}(i, j) \subseteq P_{G_{\psi}}(i, j)$.

II.4 Consecutive Optimization

The optimization procedure presented in previous section allows sequential optimization relative to different cost functions. Similar techniques have been used extensively for optimization of decision trees [21, 22, 23, 24] for a wide range of cost functions.

Initially we get a DAG $G = G_0$. This graph $G$ can be optimized according to a cost function $\psi_1$. As a result we obtain the proper subgraph $G_{\psi_1}$ of $G_0$ which describes the whole set of alignments that are optimal relative to $\psi_1$. We can apply the procedure of optimization relative to another cost function $\psi_2$ to $G_{\psi_1}$. As a result we obtain the proper subgraph $(G_{\psi_1})_{\psi_2}$ of $G_0$, which describes the set of all alignments that are optimal regarding $\psi_2$ among all alignments optimal regarding $\psi_1$, etc.

II.5 Computational Complexity of Optimization

Initially we get a DAG $G_0$. After optimization relative to a sequence of cost functions we get a proper subgraph $G$ of $G_0$, which is obtained from $G_0$ by removal of some edges. Let us now consider the process of optimization of $G$ relative to a cost function $\psi$.

The number of subproblems $S(i, j)$ (vertices in $G$) for a problem of aligning two sequences $X$ and $Y$ of lengths $m$ and $n$, respectively is exactly $(m + 1)(n + 1)$. It is enough to make three operations of addition and four operations of comparison for
the vertex $S(i, j)$ to compute the value $\psi_{i,j}$ and to find all edges for which

$$\psi^{(k)}_{i,j} > \psi_{i,j},$$

if the values $\psi_{i-1,j}$, $\psi_{i,j-1}$ and $\psi_{i-1,j-1}$ are already known. Therefore, the total number of arithmetic operations (comparison is also considered as an arithmetic operation) to obtain $G_\psi$ from $G$ is exactly $7(m + 1)(n + 1) = O(mn)$.

II.6 Examples

In this section two examples of sequential optimization for sequence alignment relative to two cost functions are provided.

II.6.1 Example One

Let $X$ and $Y$ be sequences over $\Sigma = \{A, T, G, C\}$, where

$$X = \text{CAATCG}, \quad \text{and} \quad Y = \text{AGGA}.$$ 

We consider a cost function $\psi$ such that

$$w(x_i, \omega) = w(\omega, y_j) = 0, \quad \text{and} \quad w(x_i, y_j) = \begin{cases} 0, & \text{if } x_i = y_j, \\ 1, & \text{otherwise}. \end{cases}$$

That is we consider matching a gap with penalty zero, a match penalty of zero and mismatch with penalty one.

One can construct the initial DAG $G = G_0$ for the considered problems. There are only 316 different alignments that are optimal relative to $\psi$, here are three of them:
Figure II.1: Graph $G_\psi$ after optimizing $G$ relative to the cost function $\psi$.

\[ \text{CAATCG} \quad \text{CAATCG} \quad \text{CAATCG} \quad \text{CAATCG} \]
\[ \text{AAGA} \quad \text{AAGA} \quad \text{AAGA} \quad \text{AAGA} \]

see Figure II.1 for the resulting DAG $G_\psi$.

We further sequentially optimize the resulting DAG $G_\psi$ relative to another cost function $\phi$, such that

\[ w(x_i, \omega) = w(\omega, y_j) = 1, \quad \text{and} \quad w(x_i, y_j) = \begin{cases} 0, & \text{if } x_i = y_j, \\ 0, & \text{otherwise.} \end{cases} \]

That is, the gap penalty is one and match/mismatch penalty is zero. see Figure II.2 for the resulting DAG $(G_\psi)_\phi$. There are 10 optimal alignment. Here are three
II.6.2 Example Two

In this example, two sequences $X$ and $Y$ over $\Sigma = \{A, T, G, C\}$, where

$$X = TTCCAC, \quad \text{and} \quad Y = ATGAA,$$

are aligned relative to $\psi$ which is

$$w(x_i, \omega) = w(\omega, y_j) = 0.5, \quad \text{and} \quad w(x_i, y_j) = \begin{cases} 0, & \text{if } x_i = y_j, \\ 1, & \text{otherwise.} \end{cases}$$
Figure II.3: Graph $G_\psi$ after optimizing $G$ relative to the cost function $\psi$.

One can construct the initial DAG $G = G_0$ for the considered problems. There are only 28 different alignments that are optimal relative to $\psi$, here are three of them:

- $TT\_CCAC$
- $\_TTCCAC$
- $\_TTCCAC$
- $ATGA\_A\_\_\_\_$
- $\_AT\_GAA$
- $\_ATGA\_A\_\_\_$

see Figure II.3 for the resulting DAG $G_\psi$.

We further sequentially optimize the resulting DAG $G_\psi$ relative to another cost function $\phi$, such that

$$w(x_i, \omega) = w(\omega, y_j) = 1, \text{ and } w(x_i, y_j) = \begin{cases} 0, & \text{if } x_i = y_j, \\ 0.5, & \text{otherwise}. \end{cases}$$

It is clear to see that now there are only 3 different alignments possible which are optimal relative to $\phi$ on the DAG $G_\psi$:

- $TTCCAC$
- $TTCCAC$
- $TTCCAC$
- $\_AT\_GAA$
- $\_ATGAA\_\_\_\_$
- $\_ATG\_AA$
Figure II.4: Graph \((G_\psi)_\phi\) after optimizing \(G_\psi\) relative to the cost function \(\phi\)

see Figure II.4 for the resulting DAG \((G_\psi)_\phi\)
Chapter III

Software and Experiments

III.1 Software Overview

The sequential optimization method suggested has been implemented using C# .Net. The software consists of graphical user interface where the user can either insert two sequences or generate sequences randomly of certain lengths he specified. Also the user has the possibility to insert the number of pairs, the gap cost, match cost, and mismatch cost for a number of optimization steps specified by him. The results are saved in a text files. These results include sequences generated, the minimum cardinality (the number of optimal solutions) after each stage, the number of operation executed in each stage.

The software has been implemented using the idea of user defined tree structure which is represented by matrix of tree nodes (matrix of struct data structure). Each tree node is implemented by struct (data structure which can holds other data types) and consists of a number of attributes. These attributes include: optimal cost of this node (same as $\psi_{i,j}$ defined in chapter 2), and the cardinality (number of children nodes), and three other boolean attributes (flags) that are computed as following:

1. IsLeft: corresponds to the left child. Boolean variable that is set to true if the
The software begins working by either reading two sequences from the textbox in the interface of the software or generating two random sequences of length specified by user. Next, the initialization step begins in which the number of optimal solutions for computation of the formula $\psi_{i,j}^{(1)} = \psi_{i-1,j} + w(x_i, \omega)$ resulted in optimal cost value at this node.

2. IsRight: corresponds to the right child. Boolean variable that is set to true if the computation of the formula $\psi_{i,j}^{(2)} = \psi_{i,j-1} + w(\omega, Y_i)$ resulted in optimal cost value at this node.

3. IsMiddle: corresponds to the middle child. Boolean variable that is set to true if the computation of the formula $\psi_{i,j}^{(3)} = \psi_{i-1,j-1} + w(x_i, Y_i)$ resulted in optimal cost value at this node.

In this implementation DAG is represented as a matrix, and its vertices are represented by tree nodes (cells of the matrix). Setting any one of the three flags above to true means that the edge to the corresponding child of the current node at $S(i,j)$ exists and we will continue with optimization of this edge in the next steps of optimization. Note that all of the three edges (corresponds to three children) can exist, two of them, or one of them depending on their optimum cost values. Note that in Figure III.1 all children have optimal cost value so all flags are set to true. In Figure III.2 the left and right children have optimal value so their corresponding attributes are set to true and the middle child have a cost value greater than the other two children so we have to set IsMiddle to false and this corresponds to removing the middle edge from parent $S(2, 1)$ to the child $S(1, 0)$.
the nodes located at upper row and leftmost column (correspond to leaves in DAG) of the tree matrix is set to number of gaps multiplied by the index of the node and the cardinality of each of these nodes are set to zero since they are leaves in the tree.

The following step is the matrix filling step in which each node (other than located in the first row and the first column) attributes is computed. Optimal cost value at each node is computed by:

$$\psi_{i,j} = \min_{k \in K_G(i,j)} \psi_{i,j}^{(k)},$$

where

$$\psi_{i,j}^{(1)} = \psi_{i-1,j} + w(x_i, \omega),$$
$$\psi_{i,j}^{(2)} = \psi_{i,j-1} + w(\omega, y_j),$$
$$\psi_{i,j}^{(3)} = \psi_{i-1,j-1} + w(x_i, y_j).$$

The values of each one of the flags (IsMiddle, IsRight, IsLeft) is computed as described above, and cardinality is computed by adding the cardinality of the children for this node. Finally, the traceback starts at the node located in the last row and last column corresponding to parent node and choosing the nodes with optimal cost that are true (have edge from parent) and this ends the first step of optimization. Next steps of optimizations are done in the same way but with exception that we consider only edges that are true in the computation of optimal cost at each node instead of considering the three edges at each node, and if one of edges is not optimal relative to the new cost function then we subtract its cardinality from its parent node, set its flags to their correct values, and we don’t consider it in the traceback step.
To see mapping between DAG and tree matrix let us study the following example. Let us check example one of chapter three to see how the algorithm works. The cost functions are described as following:

For the first step of optimization:

\[ w(x_i, \omega) = w(\omega, y_j) = 0, \quad \text{and} \quad w(x_i, y_j) = \begin{cases} 
0, & \text{if } x_i = y_j, \\
1, & \text{otherwise}.
\end{cases} \]

For the second step of optimization:

\[ w(x_i, \omega) = w(\omega, y_j) = 1, \quad \text{and} \quad w(x_i, y_j) = \begin{cases} 
0, & \text{if } x_i = y_j, \\
0, & \text{otherwise}.
\end{cases} \]

The Table III.1 shows representation of the tree structure that is used to represent the tree.

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>T</th>
<th>C</th>
<th>C</th>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table III.1: Matrix corresponding to three structure

If we consider cell(3,3) then this node is represented as struct of the following attribute value: optimal cost = 2, IsLeft = true (Indicates that the value of the left cell, it is in this case cell(2,3), has the minimum value), Ismiddle = true (indicates that the value of the diagonal cell, in this case cell(2,2), has the minimum value, IsRight = true (Indicates that the value of the upper cell above it, in this case cell(3,2), has the minimum value). Note that the minimum value at this step (as well as others) is computed as following:
\psi_{3,3}^{(1)} = \psi_{2,3} + w(C, \omega) = 1.5 + 0.5 = 2,
\psi_{3,3}^{(2)} = \psi_{3,2} + w(\omega, T) = 1.5 + 0.5 = 2,
\psi_{3,3}^{(3)} = \psi_{2,2} + w(C, T) = 1 + 1 = 2.

So all of these values are equal and hence they are minimum so we set the three flags to true. In the second step of optimization all of them will be considered. Let us check second step of optimization. The minimum cost at each node is computed using the following formula:

\psi_{3,3}^{(1)} = \psi_{2,3} + w(C, \omega) = 1.5 + 1 = 2.5,
\psi_{3,3}^{(2)} = \psi_{3,2} + w(\omega, T) = 1.5 + 1 = 2.5,
\psi_{3,3}^{(3)} = \psi_{2,2} + w(C, T) = 0.5 + 0.5 = 1.

Here the minimum value is 1 and this indicates that this is the only cell that will be considered in traceback step while looking for optimal alignments. Note that setting one of the flags to zero means removing that edge from the DAG. Note also that the number of optimal alignments for any node is the sum of the numbers of optimal solutions for its children nodes. Finally, nodes at first row and first columns are terminal nodes and they have "cardinality" of zero.

III.2 Experiments

In this section, experiments that have been done are described along with their results. The set of experiments were done based on the dissertation goals to be verified. In each of the following experiments, 100 pairs of randomly generated sequences of
certain length have been used and the final result at each length has been considered as average value of all of them. The number of optimization steps that have been studied is two and in each step a different cost function, corresponding to match, mismatch and gap cost, has been used. Note that the number of optimization steps could be any number of steps other than two. The length of both sequences in each pair could be similar or different but in the following experiments they are equal. The random number generator that is used is built in the .Net environment. The alphabet that is used is \{A, C, G, T\}. The algorithm steps are as following:

1. Read sequences lengths \{L_1, L_2\}, number of Optimization steps, the cost functions for each step, the number of sequences pairs.

2. Generate 100 pairs of sequences of length \{L_1, L_2\}.

3. For each sequence pair do the following two steps. Perform the first step of optimization relative to the first cost function, compute the number of optimal solutions and proceed to the second step of optimization.

4. Perform the second step of optimization relative to the second cost function on the set of optimal solutions resulted from the first step of optimization. Compute the number of optimal alignments. Save the results to a text files.

III.2.1 Experiment One: Sequence Length vs. Number of Executed Operations

In this experiment, the cost functions that have been used for the first step of optimization are: mismatch cost = 1, match cost = 0 and gap cost = 1. For the second step of optimization the gap cost = 0, match cost = 0 and mismatch cost = 1. The meaning of the second cost function is to minimize the number of mismatches in the aligned sequences. The experiment has been done for 100 pairs of sequences per length
III.2.2 Experiment Two: Sequence Length vs. Number of Optimal Alignments

In this experiment, the cost function that is used for the first step of optimization is: mismatch cost = 1, match cost = 0 and gap cost = 1. The meaning of this cost function is to minimize the number of mismatches and gaps in the aligned sequences. For the second step of optimization, the cost function that has been used is: match cost = 0, mismatch cost = 0 and gap cost = 1 and here the number of gapped alignments are to be minimized. Sequences have been generated starting from length...
Figure III.4: Results of experiment one. Sequence length vs. number of executed operations during the second step of optimization

Figure III.5: Results of experiment one. Sequence length vs. average number of executed operations during the initialization step of dynamic programming algorithm
Figure III.6: Results of experiment one. Sequence length vs. average number of executed operations during the filling step of dynamic programming algorithm.

Figure III.7: Results of experiment one. Sequence length vs. average number of executed operations during traceback step of dynamic programming algorithm.
Figure III.8: Results of experiment two. Sequence length vs. number of optimal alignments resulted from optimizing relative to the first cost function 200 up to 4800 with step length of 200. The goal of this experiment is to study the relation between the number of optimal alignments after each step of optimization and the sequence length. The results are shown in Figures III.8, III.9.

III.2.3 Experiment Three: Sequence Length vs. Number of Optimal Alignments

In this experiment, the cost function for the first step of optimization is: match cost = 0, mismatch cost = 1, gap cost = 1. The meaning is that we are trying to minimize the number of mismatches and the gapped alignments between both sequences. For the second step of optimization the cost function is: match cost = 0, mismatch cost = 0.5, gap cost = 1. The meaning of this cost function is that we want to minimize both gapped alignment and mismatches between both sequences but we give the mismatches better chance to occur than gaps. The sequence length has been taken as multiple of 20 starting from 20 up to 500. The goal of this experiment is
III.2.4 Experiment Four: Sequence Length vs Number of Optimal Alignments

The goal of this experiment is to study the relation between the number of optimal alignments after each step of optimization and the sequence length. The cost function that is used at the first step of optimization is: match cost = 0, mismatch cost = 1, gap cost = 1. For second cost function, match cost = 0, mismatch cost = 0.5, gap cost = 1. Sequences lengths have been taken as a multiple of 200. The results are shown in Figures III.12, III.13. What is different from experiment three is that the
Figure III.10: Results of experiment three. Sequence length vs. number of optimal alignments resulted from optimizing relative to the first cost function.

Figure III.11: Results of experiment three. Sequence length vs. number of optimal alignments resulted from optimizing relative to the second cost function.
Figure III.12: Results of experiment four. Sequence length vs. number of optimal alignments resulted from optimizing relative to the first cost function sequences are taken as multiple of 200 starting from 200 up to 4000.

### III.2.5 Experiment five: Sequence Length vs. Number of Optimal Alignments

The cost function of the first stage is: match = 0, mismatch = 1 and gap cost = 1. For the second stage we have match cost = 0, mismatch cost = 1 and gap cost = 0.5, which means that at the first step we are trying to optimize the number of gaps and mismatches but we are giving gaps better chance to occur during the second step of optimization. The goal of this experiment is to study the relation between number of optimal alignments after each step of optimization and sequence length but using different cost function than that are used in the previous experiments. The results are shown in Figures III.14, III.15.
Figure III.13: Results of experiment four. Sequence Length vs. Number of Optimal Alignments resulted from optimizing relative to the second cost function.

Figure III.14: Results of experiment five. Sequence length vs. number of optimal alignments resulted from optimizing relative to the first cost function.
Discussion

After performing the previous experiments, results have been analyzed using Excel sheets. In experiments two, three and four the goal was to study the relation between sequence length and the number of optimal solutions after each step of optimization and to see how these results are affected by the cost function that is used. The results for 100 sequences of each length have been entered to Excel sheet. After that, the logarithm of each result has been computed. Finally, average of logarithms, maximum of logarithms and minimum value of logarithms have been computed and plotted versus sequences lengths. The plots corresponding to logarithmic values of the actual values are clearer than plotting the absolute values which result in exponential curve. The results of experiments show that sequential optimization for sequence alignment problem reduces the number of optimal solutions after each step of optimization. For example, from the results of experiment two, which are represented in Figures III.8 and III.9 the average number for binary logarithms of number of optimal solutions
for sequences of length 4600 after optimization relative to the first cost function is 298.1618466 while this number drops to 58.13909589 after the second stage of optimization.

The plots corresponding to these experiments show that the behavior of this algorithm is almost stable because there are no outliers among the resulted points and the average of logarithms curve is bounded below by minimum of logarithms curve and bounded above by maximum of logarithms curve which are behaving in similar way as average curve.

Also the results of experiments show that the number of optimal solutions grows exponentially as the sequence length grows linearly and this is proved by doing exponential curve fitting for the absolute values resulted from experiments and doing linear curve fitting for the logarithmic values for the average, maximum, and minimum values of results.

Furthermore, the results of these experiments show that the number of optimal solutions after each step of optimization varies depending on the cost function that is used. For example, in experiment two, the average number for binary logarithms of number optimal solutions for sequences with length 4000 relative to the first cost function is 259.1357977 while this number drops to 50.92728274 after optimization relative to the second cost function. In experiment four, the average number for binary logarithms of optimal solutions for sequences with length 4000 is $2.73E+02$ after optimizing relative to the first cost function and this value drops to $1.74E+02$ after optimization relative to the second cost function. So the cost function itself affects the results.

Figures III.3, III.4 show that the time (number of executed operations) grows polynomially as the sequence length grows linearly which verifies the proved time complexity of the algorithm. Since both sequences are of the same lengths then $O(mn)$ bound that is proved in this thesis is equal to $O(n^2)$. Polynomial behavior
has been proved using polynomial curve fitting in Excel. Another point is that the
time (number of executed operations) might increase or decrease in the second stage
of optimization and this is due to the fact that the same sequences can have different
paths in traceback step using different cost function. So if the time increase that
means that the sequences are following longer paths in traceback step and the opposite
is true, less time means shorter path in traceback step. Finally, sequences of higher
length might take less time than sequences of shorter lengths and this is because
different sequences have different structure and might take different paths in the
traceback step; one sequence might have shorter paths than others. For example, for
sequence length of 4000 the number of operations executed is 348047582.7 but for the
second step of optimization the number of executed operations is 418761685.3 which
is larger than the number of steps executed in the first step of optimization.

Finally, analysis for different steps of the algorithms shows that the initialization
step is executed in linear time, filling step in polynomial time, traceback step in linear
time as the number of sequence length is increased linearly and this behavior has been
proved using curve fitting in Excel. See Figures III.5, III.6, III.7.
Chapter IV

Conclusion and Recommendations

This dissertation discussed in details a procedure to sequentially optimize pairwise global sequence alignment relative to different cost functions. Chapter 1 provided an overview about sequence alignment as well as literature review of the existing algorithms and softwares to solve this problem. In chapter two, a detailed theoretical analysis for the suggested methodology has been done along with a set of examples to explain the concepts. It has been proved that for two sequences of length $m$ and $n$, each step executes in $O(mn)$ time. Chapter 3 explained the software that has been implemented to simulate the suggested methodology as well as a description of experiments that have been carried out to test the methodology. Chapter 3 ended with brief discussion of the results obtained.

The objectives of this dissertation have been formulated as following:

1. To test the applicability of the methodology. This is done by measuring time complexity of the algorithm on different sequence length. The idea is that if this time is acceptable on long sequences then this is could be indication on the possibility of application of this methodology on real life problems.

2. To test the effect of different cost functions on the algorithm behavior especially the resulted number of optimal solutions after applying each cost function.
3. To study how the number of optimal solutions grows as the sequence length grows linearly.

The results of experiments verified the proved statements in that each step executed in polynomial time depending on the sequences length which was reasonable execution time so this give an evidence of possibility of applying this method on real life biological sequences. Furthermore, this method becomes impractical for long sequences. Second, testing different cost functions resulted into getting different values for execution time and a different number of optimal solutions. Third, the number of optimal solutions after each step of optimization was reduced, while this number grows exponentially as sequence length grows linearly. Future work and agenda include:

1. To study local sequence alignment.

2. To find totally optimal alignment, which are alignments that are optimal relative to two cost functions, using different cost functions.

3. To optimize the code and memory space used by this algorithm so we can apply it on sequences of longer lengths.

4. To apply this method on multiprocessor machines.

5. To study the effects of using different cost function for different letters of alphabet of the sequence.

6. To find biological application for the software.
APPENDICES
Appendix A

Papers Published

- Igor Chikalov, Shahid Hussain, Mikhail Moshkov, Enas Odat, ”Sequential Optimization of Global Sequence Alignments Relative to Different Cost Functions”, In Proceedings of International Conference on Convergence and Hybrid Information Technology (ICHIT 2010).
Bibliography/References


