GENERIC C++ IMPLEMENTATIONS OF PAIRWISE SEQUENCE ALIGNMENT: INSTANTIATION FOR LOCAL ALIGNMENT

Xiao Yang

A Major Report

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

August 2003

Your file Votre référence
ISBN: 0-612-91147-0
Our file Notre référence
ISBN: 0-612-91147-0

ABSTRACT

Generic C++ Implementations of Pairwise Sequence Alignment: instantiation for local alignment

Xiao Yang

Although there are already several C implementations of pairwise sequence alignment in the EMBOSS library for bioinformatics, all of them are quite independent of each other. The main purpose of this project is to develop a generic application to unify the different implementations and to provide the developer with the capability to develop a pairwise alignment algorithm with little effort.

C++ template technology provides high levels of performance and reusability of programming abstractions. The template mechanism in C++ is used in this project to achieve generic algorithm. An alignment algorithm is defined as a function object, which will be passed as a parameter to a generic implementation of dynamic programming. In this report, the local alignment algorithm of Smith-Waterman is instantiated.

This application provides two kinds of reusability: generic algorithm reusability and function objects reusability.

ACKNOWLEDGEMENTS

The author wishes to thank all those who made the final realization of this major report possible. It is not possible to mention all their names, however the author would like to express my special gratitude to the following contributors.

The author is extremely thankful to the author's supervisor Prof. Greg Butler for his professional supervision, intellectual guidance, and patience, through this study. It was a pleasure working with him.

The author wishes to express his thanks to his partner, Yan Zhang, for the countless hours of devotion and hard work she has put into this major report, and for steadily developing her section of the work despite the author's whim and occasional deficiency in communication.

This report is dedicated to the author's family for their help, lover and encouragement to achieve excellence. The author's sincere thanks are due to the support by his company, Motorola Software Centre, Montreal, Quebec, Canada. The author's special thanks to Alf Lund, project manager for his inspiration and encouragement in many ways during the author's master program study.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Bioinformatics Preliminaries

In the last few decades, advances in molecular biology and the equipment available for research in this field have allowed the increasingly rapid sequencing of large portions of the genomes of several species. Before long, the DNA sequence of the complete human genome will have been completely determined. This achievement might seem an end in itself, but it is really only the beginning [10].

In fact, up to date, many bacterial genomes, as well as those of some simple *eukaryotes* and more complex *eukaryotes* have been sequenced in full. The Human Genome Project, designed to sequence all 24 of the human chromosomes, is also progressing and a rough draft was completed in the spring of 2000. Popular sequence databases, such as GenBank and EMBL, have been growing at exponential rates. This flood of information has demanded the careful storage, organization and indexing of sequence information [3].

The sum of all this information is enormous and its potential in our understanding of life processes, if rightly explored, is far-reaching. In order to exploit this wealth of information a new field of science has arisen that combines biology and medicine on one side with mathematics, statistics and computer science on the other side. This new field of science is known as bioinformatics [15].

Bioinformatics is a new area where biology, medicine, computer science, mathematics and information technology are merged into a single discipline. Moreover, Bioinformatics is the recording, annotation, storage, analysis, searching and retrieval of nucleic acid sequence, protein sequence and structural information. This includes databases of the sequences and structural information as well methods to access, search, visualize and retrieve the information.

There are three main aims of bioinformatics. First is to organize data to allow researchers to access existing information and to submit new entries when they are produced. While gathering accurate data is essential, the information stored in these databases is useless unless they are analysed. Thus, this leads to the second objective, which is to develop new algorithms and statistics with which to assess relationships among members of large data sets. The third aim is to develop and implement tools that enable efficient access, management and analysis of different types of information; moreover, to use these tools to analyse and interpret various types of data including nucleotide and amino acid sequences, protein domains, and protein structures.

Traditionally, biologists examined the data individually and compared them only with a few that are related. With the aid of computational techniques, we can now conduct global analyses of all available sequences and enable the discovery of new biological insights as well as to create a global perspective from which unifying principles in biology can be discerned.

The molecular sequences we are studying and those we find in the database that provide useful information are related to each other by having a common ancestor in the genomes in some ancient organism. Molecular sequences that share a common ancestral molecular sequence are referred to as homologous. Homology is not directly observable. It is inferred from the observation of sequence identity, or similarity. Therefore, homology is a conclusion drawn that the two genes share a common evolutionary history. Homology is not a matter of degree, at any given position in alignment, sequences and individual positions either share a common ancestor or they do not.

Historically, bioinformatics as a concept was invented to describe the task of handling, presenting and analysing large amounts of sequence data. Today, due to intense efforts at a number of large research centres throughout the world, data can be rather easily accessed by anyone over the Internet and World Wide Web servers. As a consequence, it is currently almost an everyday activity in most molecular biology labs to screen these sequence databases to find sequence homology of a particular gene.

Sequence alignment aims to provide an explicit mapping between the residues of two or more sequences. These techniques are central to modern molecular biology. The main goal of sequence alignment is to enable researchers to determine whether two sequences display sufficient similarity such that an inference of homology is justified. When choosing the appropriate algorithm, it depends on the type of problem we need to solve. Suppose we are searching in protein and DNA databases, it is best to use local alignment methods, i.e. to find the strongest similarity between two sequences and ignore the differences outside the most similar region. When we have two homologous sequences and we are comparing the overall pattern of the two, it is better to use global alignment methods. Also, we may distinguish the sequence alignment methods into pairwise alignments, which involve only two sequences, and multiple alignments, which compare more than two sequences.

1.2 Contributions

The research work for this project was supervised by Prof. G. Butler. The study was started in September 2002. First we worked to understand the fundamental concepts of bioinformatics, especially focused on pairwise sequence alignment algorithms. Second we began to study the EMBOSS libraries for the C implementations of pairwise alignment algorithms. Third we started another learning session to understand the methodology of generic programming and STL. Then we began our final implementation of pairwise sequence alignment algorithm using the template mechanism and the dynamic programming algorithm in C++ to represent a generic algorithm and a specific objective function.

The major contribution of this report is an implementation of the pairwise sequence alignment algorithm using generic programming in C++. In spirit of this engineering design approach, our implementation will provide a generic skeleton for pairwise alignment algorithm in C++, which will overcome the common drawbacks from the C implementations such as dependency on the algorithm, lack of reusability, and difficulty to maintain. The motivation for this major report is two fold: first, to enable the user to instantiate any kind of pairwise alignment algorithm with little effort by re-using several components; second, to give flexibility at implementing the method. To achieve the goals of flexibility, reusability and ease of maintenance, this major report will carefully design the skeleton, identify the common entities of a pairwise alignment method and use generic programming technology based on templates.

1.3 Joint Effort

This project is a joint project, in cooperation with Yan Zhang. We shared the understanding of bioinformatics domain technology and discussion the methodology of object-oriented programming and generic programming. We worked together in the following components of this project:

- Framework design and implementation
- Objective function common interface design

The following contents may be similar in our major report:

- Experiment data
- Recommendation for the future work

We implemented global alignment algorithm and local alignment individually.

1.4 Pair Programming

There are two developers involved in this project. Actually it is not just pair "programming", it is also pair working. We did not work separately as solo programmer. From the beginning of the project, we are working in same physical place, discussing and analyzing the requirement, coming out with a design together. In the code and unit test phase, we applied pair programming by sitting in front of the same desktop from time to time.

According to our experience of pair programming in this project, and the paper "The Costs and Benefits of Pair Programming" [6], the following advantages can be highlighted:

1. **Teamwork:** It is pair programming that let me understand the spirit of teamwork. From my experience in this project, collaborating with the other person means sharing ideas, understanding our differences and compromising.
2. **Review code:** Today, almost everyone in software engineering field knows how important the code review and code inspection are. Code review can catch the errors in the early stage. Pair programming is the perfect way to carry out code review; the pair shares the same domain background of code and same domain knowledge. Every line of the code is produced by two people. So the review is effective.
3. **Be creative:** Exchanging / sharing the ideas can be creative. Thinking out loud and brainstorming are two good ways to share and exchange ideas.
4. **Be productive:** Pair programming with code review can produce a high quality product. Sharing ideas can quickly lead to a solution.

1.5 Outline of the report

The organization of this report is as follows: Chapter 2 reviews the Pairwise Sequence Alignment. Chapter 3 describes the Generic Programming and STL as needed in this report. Chapter 4 briefly presents the requirements. Chapter 5 covers the Object-Oriented Design and C++ implementation of our Basic Sequence Algorithms using C++ template mechanisms. Chapter 6 presents the experimental results. Chapter 7 discusses the use of the application. Finally, Chapter 8 presents the conclusion of the report and the future work.

2 Pairwise Sequence Alignment

2.1 Introduction

Sequence alignment is a crucial operation in bioinformatics and genomics research. An alignment refers to the procedure of comparing two or more sequences by looking for a series of individual characters or character patterns that are in the same order in the sequences. Early in the days of protein and gene sequence analysis, it was discovered that the sequences from related proteins or genes were similar, in the sense that one could align the sequences so that many corresponding residues match. This discovery was very important, since strong similarity between two genes is a strong argument for their homology (that is, for an evolutionary relationship) [8].

In sequence alignment, two or more strings are aligned together in order to get the highest number of matching characters. Gaps may be inserted into a string in order to shift the remaining characters into better matches. Typically a scoring function, substitution or scoring matrix is used to rank different alignments so that biologically plausible alignments score higher. The task of optimal sequence alignment is to find the best possible alignment for a given scoring function and set of sequences.

In an optimal alignment, non-identical characters and gaps are so placed to bring as many identical or similar characters as possible into vertical register. Two types of sequence alignment have been recognized: global and local. The global alignment optimizes the alignment over the full-length of the sequences. In local alignment, stretches of sequence with the highest density of matches are given the highest priority [23].

The rationale behind the comparison of sequences may be manifold. Above all, the theory of evolution tells us that gene sequences may have derived from common ancestral sequences. Thus it is of interest to trace the evolutionary history of mutations and other evolutionary changes. Comparison of biological sequences in this context is understood as comparison based on the criteria of evolution. For example, the number of mutations, insertions, and deletions of bases necessary to transform one DNA sequence into another one is a measure reflecting evolutionary relatedness.

There are many features of sequence alignments that give interesting information. For example, a closer analysis of the alignment can reveal which parts of the sequences that are likely to be important for the function, if the proteins are involved in similar processes. In parts of the sequence of a protein, which are not very critical for its function, the random mutations can accumulate more easily. In parts of the sequence that are critical for the function of the protein, hardly any mutations will be accepted; nearly all changes in such regions will destroy the function.

2.2 Substitution Matrices

Early sequence alignment programs used unitary scoring matrix. A unitary matrix scores all matches the same and penalizes all mismatches the same. Although this scoring is sometimes appropriate for DNA and RNA comparisons, for protein sequence alignments using a unitary matrix amounts to proclaiming ignorance about protein evolution and structure.

In bioinformatics, scoring matrices for computing alignment scores are often based on observed substitution rates, derived from the substitution frequencies seen in multiple alignments of sequences. Every possible identity and substitution is assigned a score based on the observed frequencies of such occurrences in alignments of related proteins. The score is calculated from the frequency of occurrence of a match of the two individual amino acids in evolutionarily related sequences, and provides a measure of a chance alignment of the two amino acids [8].

This score will also reflect the frequency that a particular amino acid occurs in nature, as some amino acids are more abundant than others. Higher scores indicate that the probability that those two amino acids aligned by chance is very small, and lower scores indicate a high probability the two amino acids aligned by chance, and are evolutionarily unrelated. Thus, identities are assigned the most positive scores, frequently observed substitutions also receive positive scores and matches that are unlikely to have been a result of evolution, but are more likely indicative of unrelatedness at that position, are given negative scores.

Matrices with scoring schemes based on observed substitution rates are superior to simple identity scores, or scores based solely on sidechain moiety similarity. The two most commonly used types of scoring matrices are the PAM matrices [25] and the BLOSUM matrices [13].

PAM (Percentage of Acceptable point Mutations per years) matrices are based on global alignments of closely related proteins. The PAM 1 matrix is calculated from comparisons of sequences with no more than 1% divergence. Scores are derived from a mutation probability matrix where each element gives the probability of the amino acid in column X mutating to the amino acid in row Y after a particular evolutionary time, for example after 1 PAM, or 1% divergence. A PAM matrix is specific for a particular evolutionary distance, but may be used to generate matrices for greater evolutionary distances by multiplying it repeatedly by itself. However, at large evolutionary distances the information present in the matrix is essentially degenerated. It is rare that a PAM matrix would be used for an evolutionary distance any greater than 250 PAMs (see Table 1).

Table 1: The log odds matrix for PAM250 (multiplied by 10)

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

Whereas the PAM matrices have been developed from global alignments, the BLOSUM (BLOcks SUbstitution Matrix) matrices are based on local multiple alignments of more distantly related sequences. For instance, BLOSUM 62 (see Table 2), the default matrix in BLAST, is a matrix calculated from comparisons of sequences with no less than 62% identity. Unlike PAM matrices, new BLOSUM matrices are never extrapolated from existing BLOSUM matrices, but are always based on local multiple alignments. So, the BLOSUM 80 matrix is derived from a set of sequences having 80% sequence identity [8].

Table 2: The log odds matrix for BLOSUM62

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

The level of relatedness of a set of sequences, therefore, directly effects which scoring matrix is most appropriate for aligning the set, whether or not it is a PAM or a BLOSUM matrix. Comparisons of closely related sequences should use BLOSUM matrices with higher numbers and PAM matrices with lower numbers, and BLOSUM matrices with low numbers and PAM matrices with high numbers are preferable for comparisons of distantly related proteins. A single matrix may nevertheless be reasonably efficient over a relatively broad range of evolutionary change.

2.3 Gap Penalties

The concept of a gap in an alignment is important in many biological applications, because the insertion or deletion of an entire subsequence often occurs as a single mutational event. Moreover, many of these single mutational events can create gaps of quite varying sizes. At the protein level, two protein sequences might be relatively similar over several intervals but differ in intervals where one contains a protein subunit that the other does not. One concrete illustration of the use of gaps in the alignment model comes from the problem of cDNA matching [12].

Gaps help create alignments that better conform to underlying biological models and more closely fit patterns that one expects to find in meaningful alignments. The idea is to take into account the number of contiguous gaps and not only the number of spaces when calculating an alignment.

The simplest choice is the constant gap weight, where each individual space is free, and each gap is given a weight of W_s independent of the number of spaces in the gap. A generalization of the constant gap weight model is to add a weight W_s for each space in the gap. In this case, W_g represents the cost of starting a gap; called the Opening Gap Penalty [8], which is a penalty for the initiation of the gap in the sequence or structure. To make the match more significant you can try to make the gap penalty larger. It will decrease the number of gaps. Alternatively, W_s can represent the cost of extending the gap by one space, called the Extension Gap Penalty, which is applied for increasing an already existing gap by one residue. If you do not like long gaps, just increase the extension gap penalty. As well as in the opening gap penalty case, increasing an extension gap penalty may increase the significance of the match. This leads us to the affine gap weight model. This is called affine gap weight model because the weight contributed by a single gap of length q is given by the affine function $W_g + q\ W_s$. This form of penalty function is referred to as affine and has efficiency advantages over more elaborate penalty functions [12]. The constant gap weight model is simply the affine model with $W_s = 0$.

It has been suggested that some biological phenomena are better modeled by a gap weight function where each additional space in a gap contributes less to the gap weight than the preceding space. In other words, a gap weight is a convex, but not affine function of its length. Finally, the most general gap weight that might be considered is the arbitrary gap weight, where the weight of a gap is an arbitrary function of its length q. The constant, affine and convex weight models are restricted cases of the arbitrary weight model.

2.4 Dynamic Programming

The term Dynamic Programming originally only applied to solving certain kinds of operations research problems, just as Linear Programming did. In this context it has no particular connection to programming at all, and there is a mere coincidence of name. In the context of programming, Dynamic Programming is an important algorithm technique which belongs to the theory of optimization.

Dynamic Programming is efficient in finding optimal solution for cases with lots of overlapping subproblems. It solves problems by recombining solutions to subproblems, when the subproblems themselves may share sub-subproblems. In order to avoid solving these sub-subproblems several times, their results are computed and memorized, starting from the simpler problems, until the overall problem itself is solved. To apply Dynamic Programming for finding optimal solutions, the problem under concern must have optimal substructure. Optimal substructure means that the optimal solutions of local problems can lead to the optimal solution of the global problem.

Dynamic Programming was the brainchild of an American Mathematician [2], Richard Bellman, who described the way of solving problems where you need to find the best decisions one after another. In the forty-odd years since this development, the number of uses and applications of Dynamic Programming has increased enormously [4].

Dynamic Programming is a most fundamental programming technique in bioinformatics. It is particularly important in bioinformatics as it is the basis of sequence alignment algorithms for comparing protein and DNA sequences. Dynamic Programming for sequence comparison was independently invented in several fields, many of which are discussed in [24]. An introduction to Dynamic Programming in the wider context of string comparison can be found in [12]. Needleman and Wunsch [21] are often attributed as the first application of Dynamic Programming in molecular biology, while slightly different formulations of the same algorithm were described in [26] and [29].

With the variant of the Dynamic Programming algorithm first published in [11], it became possible to compute optimal alignments with affine linear gap penalties in time proportional to the product of the lengths of the two sequences to be aligned. This was a speed-up by one order of magnitude compared to a naive algorithm using this more general gap function. A further breakthrough in alignment algorithms development was an algorithm that could compute an optimal alignment using computer memory only proportional to the length of one sequence instead of their product [17].

In the bioinformatics application, Dynamic Programming gives a spectacular efficiency gain over a purely recursive algorithm. It converts what would be an infeasible $O(2^N)$ algorithm to an $O(N^2)$ one. The standard dynamic programming algorithm requires storage of at least one mxn matrix in order to calculate the alignment. On current computers, this is not a problem for protein sequences, but for large DNA sequences, or complete genomes, space requirements can be prohibitive. Linear-space algorithms for dynamic programming [20] overcome this problem by a recursive strategy, albeit at some sacrifice in execution time.

2.5 Local Alignment: Smith – Waterman algorithm

Dynamic programming algorithms that locate optimal alignments of two sequences are central techniques for the comparison of biological sequences or three-dimensional structures. The algorithms can be divided broadly into those that seek to find a global or local alignment between the sequences. Global alignment methods [21] optimize the score for alignment over the full length of both sequences, and are most appropriate when the sequences are known to be similar over their entire length. Local alignment methods [27] allow the common sub-regions of the two sequences to be identified and are appropriate when it is not known in advance if the sequences being compared are similar.

Local alignment methods are effective in locating common sub-domains between long sequences that otherwise share little similarity. This feature makes such algorithms suitable for scanning large sequence databanks for similarities to a newly determined sequence.

The Smith-Waterman algorithm [27] is perhaps the most widely used local similarity algorithm for biological sequence comparison. The algorithm identifies the single highest scoring sub-sequence alignment and allows for gaps (insertions/deletions). However, it is often true that there may be more than one biologically important alignment between two sequences. For example, a protein domain may be repeated, or domains may be shuffled within multi-domain proteins.

Waterman and Eggert [28] have shown how the Smith-Waterman algorithm may be extended to locate the second-best and subsequent local alignments with minimal recalculation, subject to the primary restriction that the different alignments should not intersect.

Smith and Waterman [27] compute an optimal local alignment by defining H_{ij} as the local similarity measure between the partial sequences $\mathbf{a} = a(1), a(2), \ldots a(i)$, and $\mathbf{b} = b(1), b(2), \ldots b(j)$. They define $H_{ij} = 0$ when either i=0 or j=0 and show that

$$H_{ij} = \max \left\{ \begin{array}{l} H_{i-1,j-1} + s(a(i), b(j)), \ \max\limits_{k \geq 1} \{H_{i,j-k} - w_k\}, \\ \max\limits_{l \geq 1} \{H_{i-l,j} - w_l\}, \ 0 \end{array} \right\}. \quad (2)$$

In this case

$$S(\mathbf{a},\mathbf{b}) = \max_{i,j} \{H_{ij}\}. \tag{3}$$

The zero boundary conditions and the critical extra zero in (2) are all that is needed to turn a global alignment algorithm into a local one. From this point on, the theoretical development will deal only with global alignment.

Both (1) and (2) are defined using the classical notion of a gap as containing inserted or deleted members in just one of the two sequences. When gaps are allowed to contain unmatched residues from both sequences, then it is straightforward to see that (1) becomes

$$S_{ij} = \max\left\{S_{i-1,j-1} + s(a(i),b(j)),\ \max_{max\{k,l\} > 0} \{S_{i-l,j-k} - w_{k+l}\}\right\}. \tag{4}$$

In equation (4) there are two terms on the right hand side:

1. This is the case where *a*(*i*) and *b*(*j*) are matched with each other in the optimal alignment. In this case, the optimal alignment score is the similarity score of aligning *a*(*i*) with *b*(*j*) plus the optimal score from aligning the remainder of the sequences.
2. In this case, alignment ends with a gap. There are inserted residues in **a** if $l > 0$ and in **b** if $k > 0$.

Both (1) and (2) present serious computational problems as they stand. For sequences of length *m* and *n*, the computation time for the alignment algorithm is $O(mn\max\{m,n\})$ if the recursions are executed as written. For the extended definition of gap the situation is even worse, since a naive application of (4) requires $O(m^2n^2)$ computer time to execute. Fortunately, these algorithms can all execute in $O(mn)$ time when gap costs are affine, as was shown by Gotoh [11]. Replacing his distance-minimizing notation with similarity maximizing notation, we can write (1) as

$$S_{ij} = \max\{S_{i-1,j-1} + s(a(i), b(j)), P_{ij}, Q_{ij}\}, \tag{5}$$

where

$$P_{ij} = \max_{k \geq 1} \{S_{i,j-k} - w_k\} \tag{6}$$

and

$$Q_{ij} = \max_{l \geq 1} \{S_{i-l,j} - w_l\}. \tag{7}$$

Note: *P* and *Q* can be called auxiliary arrays and are often used in dynamic programming.

Gotoh proved that

$$P_{ij} = max\{S_{i,j-1} - w_1, P_{i,j-1} - \beta\} \tag{8}$$

and that

$$Q_{ij} = max\{S_{i-1,j} - w_1, Q_{i-1,j} - \beta\}. \tag{9}$$

The advantage of (8) and (9) is that both P_{ij} and Q_{ij} can be computed in just two steps each, so that only a few steps are required for each *ij* pair.

When the notion of gap is generalized, equation (4) can be rewritten as

$$S_{ij} = \max\{S_{i-1,j-1} + s(a(i), b(j)), P_{ij}\}, \tag{10}$$

where

$$P_{ij} = \max_{max\{k,l\} > 0} \{S_{i-l,j-k} - w_{k+l}\}. \tag{11}$$

The computation of P_{ij} seems to require $(i-1) \times (j-1)$ steps. However, following Gotoh's reasoning, we deduce

$$\begin{aligned}
P_{ij} = & \max\left\{\begin{array}{c} S_{i-1,j} - w_1, S_{i,j-1} - w_1, \max_{max\{k-1,l\} > 0} \{S_{i-l,j-k} - w_{k+l}\}, \\ \max_{max\{k,l-1\} > 0} \{S_{i-l,j-k} - w_{k+l}\} \end{array}\right\} \\
= & \max\left\{\begin{array}{c} S_{i-1,j} - w_1, S_{i,j-1} - w_1, \max_{max\{k,l\} > 0} \{S_{i-l,j-(k+1)} - w_{(k+1)+l}\}, \\ \max_{max\{k,l\} > 0} \{S_{i-(l+1),j-k} - w_{k+(l+1)}\} \end{array}\right\} \\
= & \max\left\{\begin{array}{c} S_{i-1,j} - w_1, S_{i,j-1} - w_1, \max_{max\{k,l\} > 0} \{S_{i-l,j-1-k} - w_{k+l} - \beta\}, \\ \max_{max\{k,l\} > 0} \{S_{i-1-l,j-k} - w_{k+l} - \beta\} \end{array}\right\} \\
= & \max\{S_{i-1,j} - w_1, S_{i,j-1} - w_1, P_{i,j-1} - \beta, P_{i-1,j} - \beta\}
\end{aligned} \tag{12}$$

Thus the computation of both S_{ij} and P_{ij} requires only a few steps, making the overall algorithm execute in time ***O*(*mn*)**.

3 Standard C++ and Generic Programming

3.1 C++ Standard Library

Before we start to explain generic programming, let us have a short look at what we mean by Standard C++ Library. The diagram, Figure 1, below is meant to be a road map. It gives you an overview of the content and structure of the Standard C++ Library.

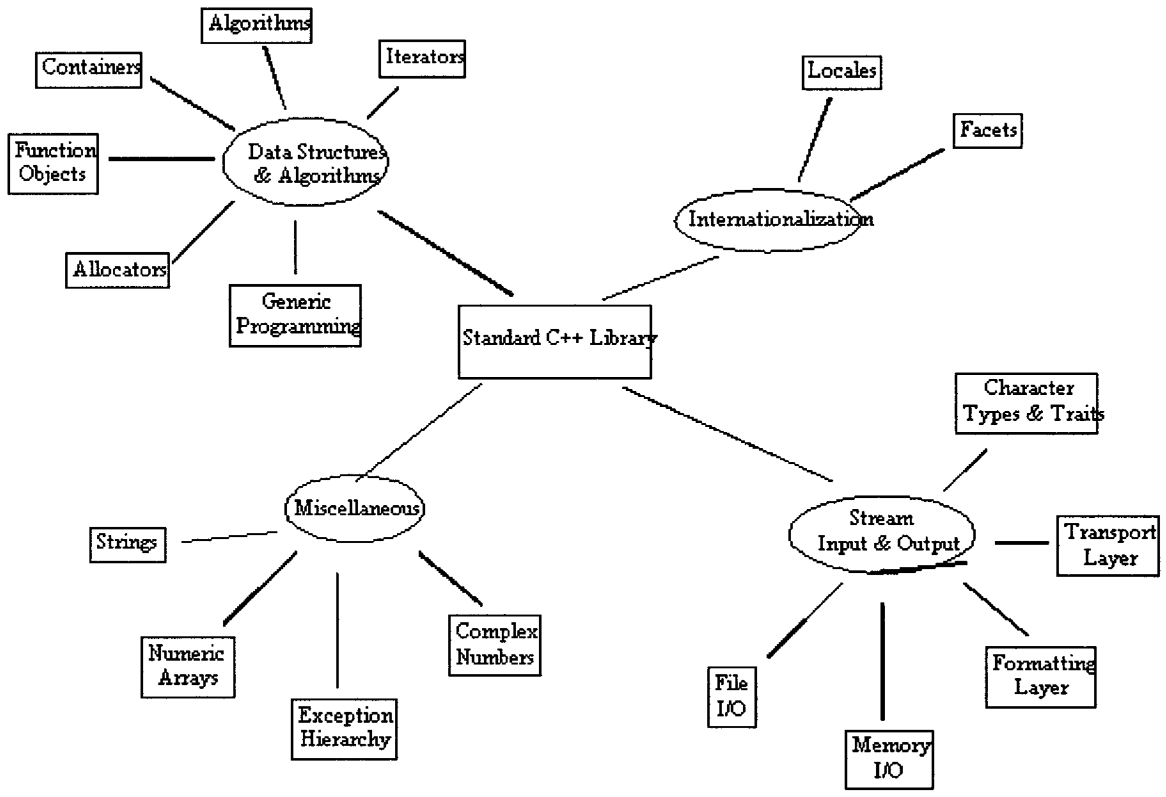


Figure 1: Overview of Standard C++ Library

Data Structures and Algorithms are based on a proposal made by Alexander Stepanov and Meng Lee of Hewlett-Packard. The proposal was submitted to the ISO/ANSI committee and accepted as part of the standard in 1994. The design of these libraries is a demonstration of generic programming, a novel programming paradigm that separates data structures from algorithms. Function objects are another category of components in the Standard C++ Library; they are used as parameters to data structures and operations and are a powerful means in generic programming. Allocators were added later on, in order to make the data structures more flexible and adaptable to different memory allocation schemes.

Internationalization is supported by the Standard C++ Library by means of the locale class and its facets. This component was designed and suggested by Nathan Myers in 1994. Major design issues were discussed and resolved until 1996.

Streams Input and Output is the next generation of IOStreams. The new standard IOStreams is a templatized and internationalized component for text and binary input and output. It aimed for staying compliant with the notion and design of the traditional IOStreams. However, fundamental changes were introduced over the time.

Miscellaneous comprises everything else. There is a string class. There is a hierarchy of exception classes, some of which are thrown by the runtime system of the language itself. Another set of classes is for numeric problems; there is a complex number class template and a numeric array. The numeric array uses sophisticated template idioms, so-called expression templates, in order to serve as a high-performance building block for matrices and multidimensional arrays.

3.2 Generic Programming

Generic programming is about generalizing software components so that they can be easily reused in a wide variety of situations. In C++, class and function templates are particularly effective mechanisms for generic programming because they make the generalization possible without sacrificing efficiency [1].

Generic programming has indeed, potentially, major advantages over "one-shot" programming, since genericity makes it possible to write programs that solve a class of problems once and for all, instead of writing new code over and over again for each different instance.

The two advantages that we stress here are the greater potential for reuse, since generic programs are natural candidates for incorporation in library form, and the increased reliability, due to the fact that generic programs are stripped of irrelevant detail, which often makes them easier to construct. Finding the right generic formulation that captures a class of related problems can be a significant challenge, whose achievement is very satisfying [7].

Thanks to parameterization, generic programming allows to abstractly represent data structures and to efficiently implement algorithms [19]. The main features of this paradigm are given by the following three statements.

- The generic expression of an algorithm only needs few hypotheses on the data it uses.
- A specialized version of an algorithm, e.g. dedicated to a particular data type, can always override the generic implementation. Nevertheless, there are no syntactic

differences between using the generic or a specialized form, and no loss in efficiency.

- The translation of an algorithm dedicated to a data type into a generic algorithm does not incur a significant overhead at run-time.

Please note that generic programming should not be confused with genericity: generic programming is an intensive use of genericity for a software architectural purpose, especially for the design of algorithm implementation, whereas the use of genericity in oriented-object programming is usually restricted to utility classes and procedures.

Generic programming highly relies on object-orientation since the notions of class, encapsulation, information hiding, inheritance, overloading and genericity are required. However, inclusion polymorphism [5] is excluded from the generic programming paradigm because, in the context of scientific numerical computing, implementation of algorithms has to be efficient and because dynamic bindings would cause an unacceptable overhead at run-time.

If, according to Meyer [16], genericity can be considered as part of the object paradigm, it is mostly used to replace macros by procedures, and to build utility classes, where a parameter usually represents a data type.

Unfortunately, the "classical" use of genericity is not well suited to numerical computing. Thus, genericity helps to write the algorithm only once, i.e., for any input type, but the use of the classical object-oriented paradigm prevents us from obtaining efficient computations.

We propose two rules that help to better define generic programming.

1. Operation polymorphism (keyword virtual in C++) is excluded because dynamic binding is too expensive. In other words, abstract methods are forbidden. As a consequence, inheritance is only used to factor method implementation and to declare attributes that can be shared by several subclasses.

2. Inclusion polymorphism is excluded. In other words, the type of a variable (static type) is exactly that of the instance it holds (dynamic type). As a consequence, each container manages exclusively objects with the same dynamic type.

These rules may seem drastic; however, C++ is a multi-paradigm language and the use of generic programming can be limited to some critical parts of code, dedicated to intensive computation, the other pieces of the software can still have a classical object-oriented design.

3.3 The Standard Template Library

The C++ STL is a powerful library intended to satisfy the vast bulk of your needs for containers and algorithms, but in a completely portable fashion. This means that not only are your programs easier to port to other platforms, but that your knowledge itself does not depend on the libraries provided by a particular compiler vendor and the STL is likely to be more tested and scrutinized than a particular vendor's library [9].

A fundamental principle of software design is that all problems can be simplified by introducing an extra level of indirection. This simplicity is achieved in the STL using iterators to perform operations on a data structure while knowing as little as possible about that structure, thus producing data structure independence. With the STL, this means that any operation that can be performed on an array of objects can also be performed on an STL container of objects and vice versa. The STL containers work just as easily with built-in types as they do with user-defined types [14].

A Container is an object that stores other objects (its elements), and that has methods for accessing its elements. In particular, every type that is a model of Container has an associated iterator type that can be used to iterate through the Container's elements [22].

The STL includes the following container classes: vector, list, deque, set, multiset, map, multimap, hash_set, hash_multiset, hash_map, and hash_multimap. For the most part, these classes and most other STL classes can be considered and used like any other library classes. If a desired functionality is needed, one needs only to find the desired class and read its documentation a bit to start using it.

The STL was originally designed around the algorithms, which are templatized functions designed to work with the containers. A concept that is used heavily in the STL algorithms is the function object or Functor. A function object has an overloaded **operator ()**, and the result is that a template function cannot tell whether you have handed it a pointer to a function or an object that has an **operator ();** all the template function knows is that it can attach an argument list to the object as if it were a pointer to a function.

Iterators are central to generic programming because they are an interface between containers and algorithms: algorithms typically take iterators as arguments, so a container need only provide a way to access its elements using iterators. This makes it possible to write a generic algorithm that operates on many different kinds of containers, even containers as different as a vector and a doubly linked list.

The STL defines several different concepts related to iterators, several predefined iterators, and a collection of types and functions for manipulating iterators. Iterators are used to traverse the elements within a container. Iterators are very similar to smart pointers and have increment and dereferencing operations. An iterator is not a general pointer, but is an abstraction of the notion of a pointer [18].

Instead of developing algorithms for a specific container, they are developed for a specific iterator category. This strategy makes it possible to use the same algorithm with a variety of different containers. STL iterators excel in performance and interchangeability, but have limitations when you are trying to maintain encapsulation or employ polymorphic behavior.

The STL addresses several problems with previous C++ container libraries in a new and innovative way. The basic goals of STL are:

1. **Flexibility** The use of generic algorithms allows algorithms to be applied to many different structures. STL's generic algorithms work on native C++ data structures such as strings and vectors.
2. **Efficiency** STL containers are very close to the efficiency of hand-coded, type-specific containers.
3. **Easy-to-learn Structure** The library is quite small owing to the high degree of generality.
4. **Theoretical foundation** The library bases its theoretical foundation on a "semi-formal" specification of the library components

STL has several disadvantages that one must learn to live with if STL is chosen:

1. Absolutely no error checking. This means that if the programmer makes an error, an error may or may not occur when the erroneous instruction is made, but it will put the program in an unstable state (e.g. Writing off the end of an array).
2. Access to the exact source that you are using may be difficult, so if you need specific information about code details, they may not be accessible

STL has several advantages:

1. Any good C++ programmer will know the syntax and typical uses of STL, meaning that code can be read by others easily

2. It is an extremely fast implementation of most of the generic containers and algorithms that a programmer would need.
3. They are debugged and ‘guaranteed’ to be correct.

4 Requirements

4.1 Requirements and analysis

This report will implement a generic application using C++ template technology for pairwise alignment algorithms. EMBOSS is a C library for a boinformatics algorithms. EMSOSS is the wide use Bioinformatics tool, in which there are several pairwise alignment algorithms that have been implemented in C. Many users are familiar with EMBOSS, so to make our tool more user friendly, our application will use the same command line parameters and formats for input and output as the EMBOSS library.

4.2 Skeleton

We propose a generic C++ implementation based on a skeleton or template for the algorithm. We categorize the basic ingredients of pairwise alignment algorithms so as to identify the main entries of the skeleton.

Output: EMBOSS has standard output for pairwise alignment algorithms, which is represented by three lines:

- The first line shows the first sequence.
- The third line shows the second sequence.
- The second line has a row of symbols.

The symbol is a vertical bar wherever characters in the two sequences match, a space wherever there is a deletion/insertion on either sequence, a colon wherever the score of characters in the two sequence are positive, otherwise a single dot. A horizontal bar may be inserted in either sequence to represent gaps.

Input: Pairwise alignment algorithms requires the following data in order to calculate the optimal alignment:

- Common input data format either the name of a file or the actual data.
- Scoring matrix file used when comparing sequences. By default it is the file 'EBLOSUM62' (for proteins) or the file 'EDNAFULL' (for nucleic sequences).
- Gap open penalty is the score taken away when a gap is created.
- Gap extension penalty is added to the standard gap penalty for each base or residue in the gap.
- Output file for the optimal alignment.

Data storage: Pairwise alignment algorithms use the same trace-back method to find the optimal solution. A Matrix is a good data structure to store score data and trace directions.

Main Procedure: There is no standard main procedure for pairwise alignment algorithm. By checking the alignment applications in EMBOSS or other alignment algorithm applications, we have observed that the main procedure looks the same for different algorithms, although different applications use different methods.

4.3 Algorithm

Pairwise alignment algorithms are optimization problems consisting of:

- Initialization function
- Gap penalty function
- Trace back function

Pairwise alignment algorithms seek the alignment with the maximal score value, called the solution.

5 Generic C++ Design and Implementations

This project presents a C++ implementation of a framework for pairwise sequence alignment and presents how to obtain an implementation of pairwise sequence alignment for a concrete problem, namely the local alignment.

5.1 Overview of design

Figure 2 shows the class diagram of the framework for pairwise sequence alignment in UML. The major classes in this framework include: the AlignmentDP, the WaterObjFunc, the Cmatrix, the Sequences, the Alignment, and the Utility.

The class AlignmentDP is a template class and is the principle engine of any pairwise sequence alignment algorithm obtained by instantiating this framework. It is responsible for initializing all the objects needed for the pairwise sequence alignment algorithm and executing the pairwise sequence alignment algorithm passed as parameterized function object. The class WaterObjFunc defines the local alignment algorithm and implements all the method required as a function object.

The class Cmatrix is a template class and defines the matrix that is used to store score data and trace information. The class Sequences defines a data structure for the raw data, which are biological sequences in the context and all operations performed on it. The class Alignment defines a data structure for output of an optimal alignment. The class Utility is a utility class, which extracts user command lines parameters and generates data necessary to instantiate above objects.

In order to design the framework to unify the different implantations of the pairwise alignments algorithms, we did a careful review of the existing alignment algorithm applications. As given in the previous chapter, we see that some entities, such as input, output, data storage, and main procedure are completely algorithm independent.

To make this framework generic, the template mechanism in C++ is used in this report to achieve a generic implementation. Moreover, a C++ parameterized component is employed to represent a pairwise alignment algorithm as a function object that the framework named generic implementation of dynamic programming should be able to accept as a parameter.

The class interfaces in the framework are described in the following subsections.

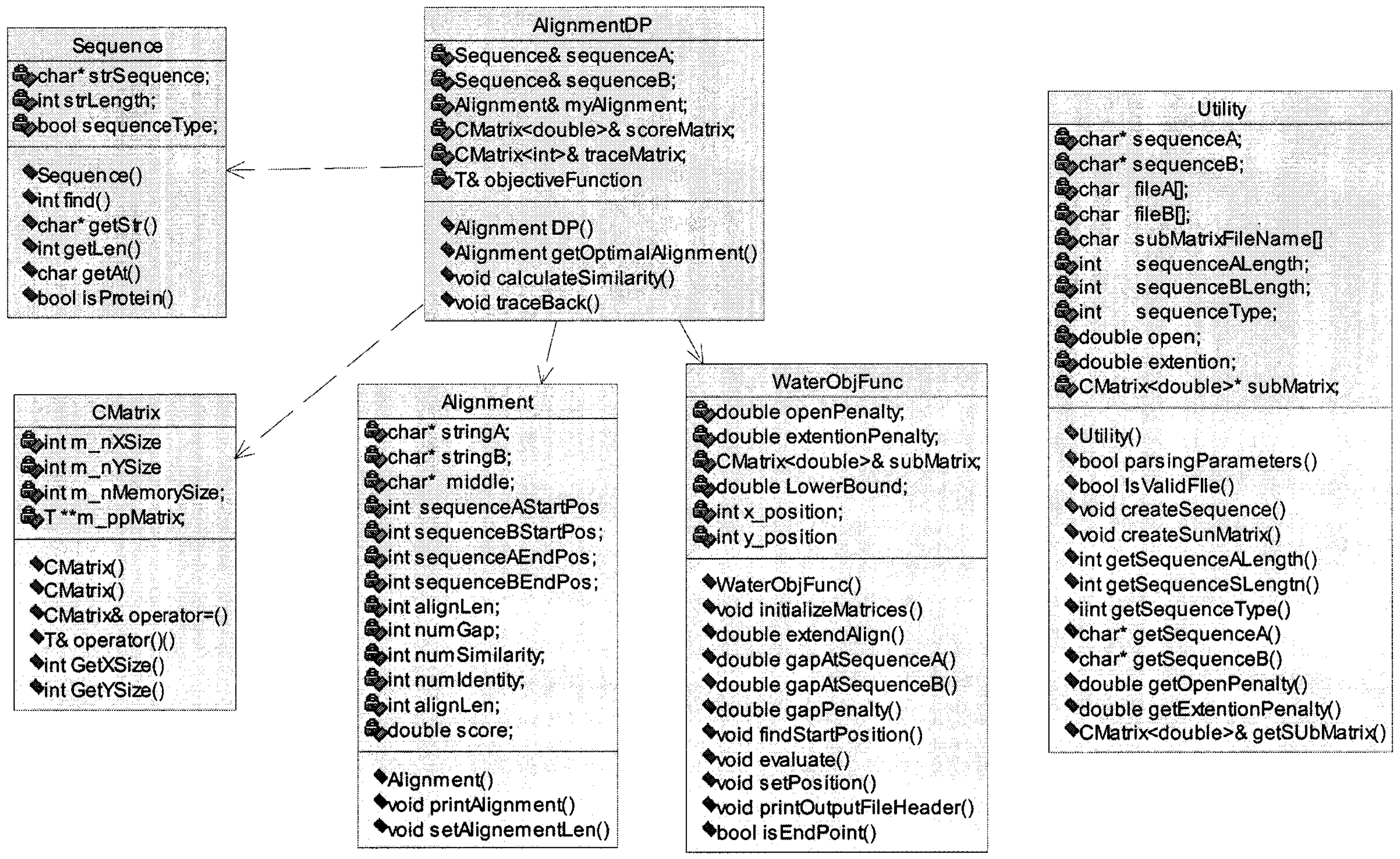


Figure 2: Class diagram in UML

5.2 Application Framework

5.2.1 Template Class AlignmentDP

The AlignemntDP is a template class, which accepts pairwise alignments algorithms as function object as well as input sequence, matrixes and output Alignment object. It provides following functionalities:

- Initialization: initialize all the objects defined in the framework.
- Execution: calls for the execution of the algorithm passed as parameterized component, which is implemented as function object.

```
template <class T>
class AlignmentDP{
private:
        Alignment               myAlignment;
        Sequence&               sequenceA;
        Sequence&               sequenceB;
        CMatrix<double>&        scoreMatrix;
        CMatrix<int>&           traceMatrix;
        T&                      objectiveFunction;
public:
        void getOptimalAlignment(Alignment& myAlignment);
        void calculateSimilarity(Alignment& myAlignment);
        void traceBack(Alignment& myAlignment);
};
```

Figure 3: The AlignementDP class

Since we need a framework in this application, to which any pairwise alignment algorithm could fit in, we choose the parameterized object function as the key abstraction of the generic component. This component is going to be the principle engine of any alignment algorithm obtained by instantiating this framework. In such a design process we have to abstract from existing algorithm and we came up with the class AlignmentDP.

The AlignmentDP class in Figure 3 implements three methods:

- getOptimalAlignment : initializes Score matrix, fills each cell in Score and Trace matrix as well as calculates similarity between both sequences. All the objective functions are called in this function.
- calculateSimilarity : creates the symbols string and calculates gap number, similarity number and identity number based on the match/mismatch between two sequences.
- traceBack : creates alignment from the trace matrix.

5.2.2 Class WaterObjFunc

The WaterObjFunc class is an object function class for the local algorithm to be demonstrated in this framework, which is a solution of a problem instance for any pairwise alignment algorithm such as global alignment algorithm, semi-global alignment algorithm and local alignment algorithm. Any pairwise alignment algorithm is defined as a function object in this framework. The users who employ this framework must provide a complete definition and implementation of their function object.

A simplified affine local alignment algorithm is defined and implemented in this report. The Figure 4 shows the detailed local alignment algorithm implemented:

```
Initialization:

        S[0][0] = 0;

        S[i][0] = S[0][j] = 0;   (0 < i < = length of sequence A, 0 < I <= length of
        sequence B)

Assignment: assign similarity score for each cell in the matrix.

Iteration:

for(int i = 1; i<= length of sequence B; i++) {
  for (int j = 1; j<= length of sequence A; j++) {
     S[i][j] =  max{ S[i-1][j-1]+ S[i][j];
                    max{ $S_{i-1,j}$ – open Penalty;$S_{i-1,j}$ – extension penalty };
                    max{$S_{i,j-1}$  - open Penalty; $S_{i,j-}$ extension penalty};
                    0;
}
```

Figure 4: Local Alignment Algorithm implemented in this report

The class WaterObjFunc in the Figure 5 is used as a private data member in the class AlignmentDP.

```
class WaterObjFunc{
private:
        double                  openPenalty;
        double                  extentionPenalty;
        CMatrix<double>&        subMatrix;
        double                  LowerBound
        int                     x_position;
        int                     y_position;
public:
        WaterObjFunc(double open, double extention,CMatrix<double>& sub)
        void initializeMatrices(CMatrix<double>& scoreMatrix,CMatrix<int>& traceMatrix,
                                Sequence& a, Sequence& b)
        double extendAlign(CMatrix<double>& scoreMatrix)
        double gapAtSequenceA(CMatrix<double>& scoreMatrix,CMatrix<int>& traceMatrix)
        double gapAtSequenceB(CMatrix<double>& scoreMatrix,CMatrix<int>& traceMatrix)
        double gapPenalty(int gapLen)
        void findStartPosition(CMatrix<double>& score,Alignment& myAlignment);
        void evaluate(CMatrix<double>& scoreMatrix,CMatrix<int>& traceMatrix,double case1,
                      double case2, double case3);
        void setPosition(int x, int y);
        void printOutputFileHeader()
        bool isEndPoint(CMatrix<double>& scoreMatrix, int x_index, int y_index)
};
```

Figure 5: The WaterObjFunc class

The WaterObjFunc class implements the following methods:

- initilaizeMatrices : Initialize Score Matrix and Trace Matrix.
- extendAlign : calculate the score value if moving to diagonal cell.
- gapAtSequenceB : calculate the score value if moving to left cell.
- gapAtSequenceA : calculate the score value if moving to up cell.

- evaluate : evaluate above three moving case value to determine the next moving direction.
- setPosition : set the x_position and y_position of the cell in the matrix.
- findStartPosition : return the start index in sequence string.
- isEndPoint : return the end index in sequence string.
- printOutputFileHeader : print the algorithm name, gap penalty value used, extention penalty value used, and output file name.

5.2.3 Template class Cmatrix

The Cmatrix class is a template class, which is designed to store score data and trace information. In this framework, a double type score Matrix holds double values and an integer trace matrix holds integer values are instantiated. Applying a move to the next cell may result in an improvement or a deterioration of the objective function value – the score value in our case. Without additional control, such a process can cause a locally optimal solution to be re-visited immediately after moving to the next cell.

5.2.4 Class Sequence

The Sequence class keeps raw data -- Proteins or DNA/RNA character string of biology molecular, which will be aligned in this framework. The Sequence class provides five methods:

- find : find the index of given character in the given string.
- getLen : return the length of character string.
- getAt : return the character in given index.
- isProtein : return TRUE if string is Proteins.

- GetStr: get raw data of the biological sequence in string format.

5.2.5 Class Alignementt

The Alignment class keeps two character strings, which contain insertion and deletion information inside sequences, as well as a string of symbols, which presents insertion and deletion, similarity and identity information. It also provides functionality to print out optimal alignment.

The Alignment class implements two methods:

- setAlignmentLen : set the length of the alignment based on the sequence length and insertions/deletions.
- printAlignment : print out the optimal alignment.

5.2.6 Class Utility

The Utility class provides the basic functionalities to extract user command line parameters such as input biology sequences, substitution Matrix and gap penalty in order to generate data necessary to instantiate above classes.

5.3 Procedure for instantiation of a pairwise alignment algorithm

Any pairwise alignment algorithm can be defined as a function object, which will be passed as a parameter to thus framework. The following demonstrates the generic procedure of how to instantiate a pairwise alignment algorithm in this framework. In this report a local alignment algorithm is instantiated and served as an example, see Figure 5.

```
.....
#include "Utility.h"
#include "WaterObjFunc.h"
//#include "NeedleObjFunc.h"
#include "ALignmentDP.h"

void main(int argc, char **argv){

        Utility myUtility(argc, argv);
        if(!myUtility.parsingParameters()){
                printf("Invalid command lines parameters, please repeat operation again!\n");
                return;
        }
         //initialize all the date structure:

        double    openPenal = myUtility.getOpenPenalty();
        double    extentionPenal = myUtility.getExtentionPenalty();
        Sequence   sequenceA(myUtility.getSequenceA(),myUtility.getSequenceType());
        Sequence   sequenceB(myUtility.getSequenceB(),myUtility.getSequenceType());
        Alignment  myAlignment(myUtility.getSequenceA(),myUtility.getSequenceB());
        CMatrix<double> scoreMatrix(myUtility.getSequenceBLength()+1,
                                         myUtility.getSequenceALength()+1);
        CMatrix<int> traceMatrix(myUtility.getSequenceBLength()+1,
                                  myUtility.getSequenceALength()+1);

        //Initialize objective function

        WaterObjFunc myObjectFunction (myUtility.getOpenPenalty(),myUtility.getExtentionPenalty(),
        myUtility.getSubMatrix());
        //NeedleObjFunc
        myObjectFunction(myUtility.getOpenPenalty(),myUtility.getExtentionPenalty(),myUtility.getSubMatrix());

        //create Alignment instance

        AlignmentDP<WaterObjFunc> myInstance(sequenceA, sequenceB, scoreMatrix,
                                                    traceMatrix, myObjectFunction, myAlignment);
        //AlignmentDP<NeedleObjFunc>  myInstance(sequenceA, sequenceB, scoreMatrix,
                                                    traceMatrix,myObjectFunction, myAlignment);

        myInstance.getOptimalAlignment(myAlignment);
        myObjectFunction.printOutputFileHeader();
        myAlignment.printAlignment();
        return;
}
```

Figure 6: Instantiation Procedure

First, a pairwise alignment algorithm needs to be defined and implemented as an object function, that is, all the methods requested as a function object must be implemented as well as its unique methods.

Second, the framework takes the newly defined function object as a parameter pass to the template version of the AlignmentDP application.

In the source code level, the following three steps need to follow to run a new pairwise alignment algorithm:

1. Include its header file for a new pairwise alignment algorithm
2. Create an objective function for a new pairwise alignment algorithm by providing the type of object
3. Create an alignment instance for a new pairwise alignment algorithm

6 Test Results

6.1 Introduction

The design of these tests is to show the correctness of the implementation of the local alignment algorithm. We present three pairs of test data for validating the implementation in this report. The first two *Hemoglobin Alpha chain* (hba_human)/*Hemoglobin beta chain* (hbb_human) and *Rattus Norvegicus cxc4 protein sequence* (Rattus Norvegicus)/ *Rattus Norvegicus cxc4 protein sequence* (Rattus Norvegicus) are pairs of protein sequences and the other *Dasypus Novemcinctus / Didelphis Virginiana* is a pair of nucleic acid sequences

For the pairs of protein sequences, the scoring function has open-gap-penalty =10, extension-gap-penalty = 0.5, and the substitution matrix is EBLOSUM62.

For the pair of nucleic acid sequences, the scoring function has open-gap-penalty = 10, extension-gap-penalty = 0.5, and the substitution matrix is EDNAFULL.

6.2 Test Data

Protein sequence 1: hba_human:

```
VLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHFDLSHGSAQVKGHG
KKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLPAEFT
PAVHASLDKFLASVSTVLTSKYR
```

Protein sequence 2: hbb_human:

```
VHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPK
VKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHF
GKEFTPPVQAAYQKVVAGVANALAHKYH
```

Protein sequence 3: Rattus Norvegicus

```
MEIYTSDNYSEEVGSGDYDSNKEPCFRDENENFNRIFLPTIYFIIFLTGIVGNGLVILV
MGYQKKLRSMTDKYRLHLSVADLLFVITLPFWAVDAMADWYFGKFLCKAVHIIYTVNLY
SSVLILAFISLDRYLAIVHATNSQSARKLLAEKAVYVGVWIPALLLTIPDIIFADVSQG
DGRYICDRLYPDSLWMVVFQFQHIMVGLILPGIVILSCYCIIISKLSHSKGHQKRKALK
TTVILILAFFACWLPYYVGISIDSFILLEVIKQGCEFESVVHKWISITEALAFFHCCLN
PILYAFLGAKFKSSAQHALNSMSRGSSLKILSKGKRGGHSSVSTESESSSFHSS
```

Protein sequence 4: Cyprinus Carpio

```
MEFYDHIFFDNSSDSGSGDFDFDELCDLKVSNDFQKIFLPVVYGIIFVLGIIGNGLVVL
VMGFQKKSKNMTDKYRLHLSIADLLFVLTLPFWAVDAASGWHFGGFLCVTVNMIYTLNL
YSSVLILAFISLDRYLAVVRATNSQNFRRVLAEKVIYLGVWLPASLLTVPDLVFAKVHD
TGMNTICELTYPLQGNTVWKAVFRFQHIFVGFLLPGLIILTCYCIIISKLSKNSKGQAL
KRKALKTTVILILCFFICWLPYCAGILVDTLVMLNVISHTCFLEQGLEKWIFFTEALAY
FHCCLNPILYAFLGVKFSKSARNALSISSRSSHKMLTKKRGPISSVSTESESSSVLSS
```

Nucleic acid sequence 1: Didelphis Virginiana

```
GCAAGTTTCCGCTACCCAGTGAGAATGCCCTTTAAGTCTTATAAATTAAGCAAAAGGAG
CTGGTATCAGGCACACAAAATGTAGCCGATAACACCTTGCTTTACCACACCCCCACGGG
AGACAGCAGTGATTAAAATTAAGCAATAAACGAAAGTTTGACTAAGTCATAATTTACAT
TAGGGTTGGTCAATTTCGTGCCAGCCACCGCGGTCATACGATTAACCCAAATTAATAAA
TAACGGCGTAAAGAGTGTTTAAGTTATATACAAAAATAAAGTTAATAATTAACTAAACT
GTAGCACGTTCTAGTTAATATTAAAATACATAATAAAAATGACTTTAATATCACCGACT
ACACGAAAACTAAGACACAAACTGGGATTAGATACCCCACTATGCTTAGTAATAAACTA
AAATAATTTAACAAACAAAATTATTCGCCAGAGAACTACTAGCAATTGCTTAAAACTCA
AAGGACTTGGCGGTGCCCTAAACCCACCTAGAGGAGCCTGTTCTATAATCGATAAACCC
CGATAAACCAGACCTTATCTTGCCAATACAGCCTATATACCGCCATCGTCAGCTAACCT
TTAAAAAGAATTACAGTAAGCAAAATCATACAACATAAAAACGTTAGGTCAAGGTGTAG
CATATGATAAGGAAAGTAATGGGCTACATTCTCTACTATAGAGCATAACGAATCATATT
ATGAAACTAAAATGCTTGAAGGAGGATTTAGTAGTAAATTAAGAATAGAGAGCTTAATT
G
```

Nucleic acid sequence 2: Dasypus Novemcinctus

```
GCAAGTATCAGCACACCAGTGAGAATGCCCTCTAACTCTTATAGATCAAAAGGAGCAAG
CATCAAGTACACACAGCCCTTACAGTAGCTCATAACCGAAAGCTTGACTAAGTTATGTT
ATTATAAGGGTTGGTAAATTTCGTGCCAGCAACCGCGGTCATACGATTAACCCAAATTA
ATAGTTATCGGCGTAAAGCGTGTTTAAGACACCTAGACAATAGAGTTAAACCCTTACTA
CGCTGTAAAAAGCCTTAGTAGGACCATAAACCCTTCAACGAAAGTGACTCTAATTTATC
TGACTACACGATAGCTAGGACCCAAACTGGGATTAGATACCCCACTATGCCTAGCCCTA
AACTAAAACAGTTCACAAACAAAACTGTTCGCCAGAGTACTACTAGCAACAGCTTAAAA
CTCAAAGGACTTGGCGGTGCTTTACATCCTTCTAGAGGAGCCTGTTCTATAATCGATAA
ACCCCGATATACCTCACCACCCCTTGCTAATACAGCCTATATACCGCCATCTTCAGCAG
ACCCTAGTAAGGCACCACAGTGAGCACAATAACATACATAAAGACGTTAGGTCAAGGTG
TAGCTTATGGGGTGGGAAGAAATGGGCTACATTTTCTAATAAAGAGCAAATACAAAAAA
```

```
CTTAATGAAACAATTTAAGACTAAGGTGGATTTAGTAGTAAGCTAAAAATAGAGAGTTT
AGCTG
```

6.3 Test Result

The following results are obtained by running the generic **AlignmentDP** application implemented in this report.

6.3.1 Test Output for aligning human protein sequences

```
Smith-Waterman local alignment.
Gap opening Penalty  : 10.0
Gap extension Penalty: 0.5
Output file name     : Alignment_output.txt
Alignment length:   145
Gaps:                 8
Identity:          63/145 (43.4%)
Similarity:        88/145 (60.7%)
score     :        293.5


sequence 1:      2 LSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHF-DLS-     49
                   |:|.:|:.|.|.||||  :..|.|.|||.|:.:.:|.|:.:|..| |||
sequence 2:      3 LTPEEKSAVTALWGKV--NVDEVGGEALGRLLVVYPWTQRFFESFGDLST     50

sequence 1:     50 ----HGSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDP     95
                       .|:.:||.|||||..|.::.:||:|::....:.||:||..||.|||
sequence 2:     51 PDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDP    100

sequence 1:     96 VNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKY         140
                   .||:||.:.|:..||.|...||||.|.|:..|.:|.|:..|..||
sequence 2:    101 ENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKY         145
```

6.3.2 Test Output for aligning Rattus Norvegicus and Cyprinus Carpio

```
Smith-Waterman local alignment.
Gap opening Penalty  : 10.0
Gap extension Penalty: 0.5
Output file name     : Alignment_output.txt
Alignment length:   357
Gaps:                12
Identity:          225/357 (63.0%)
Similarity:        276/357 (77.3%)
score     :        1117.0


sequence 1:      1 MEIYTS---DNYSEEVGSGDYDSNKEPCFRDENENFNRIFLPTIYFIIFL     47
                   ||.|..   || |.:.||||:|.: |.|....:.:|.:||||.:|.|||:
sequence 2:      1 MEFYDHIFFDN-SSDSGSGDFDFD-ELCDLKVSNDFQKIFLPVVYGIIFV     48

sequence 1:     48 TGIVGNGLVILVMGYQKKLRSMTDKYRLHLSVADLLFVITLPFWAVDAMA     97
                   .||:|||||:||||:|||.::||||||||||:||||||:|||||||||.:
sequence 2:     49 LGIIGNGLVVLVMGFQKKSKNMTDKYRLHLSIADLLFVLTLPFWAVDAAS     98
sequence 1:    98 DWYFGKFLCKAVHIIYTVNLYSSVLILAFISLDRYLAIVHATNSQSARKL    147
                  .|:||.|||..|::|||:|||||||||||||||||||:|.|||||:.|::
sequence 2:    99 GWHFGGFLCVTVNMIYTLNLYSSVLILAFISLDRYLAVVRATNSQNFRRV    148

sequence 1:   148 LAEKAVYVGVWIPALLLTIPDIIFADVSQGDGRYICDRLYP---DSLWMV    194
                  ||||.:|:|||:||.|||:||::||.|.......||:..||   :::|..
sequence 2:   149 LAEKVIYLGVWLPASLLTVPDLVFAKVHDTGMNTICELTYPLQGNTVWKA    198

sequence 1:   195 VFQFQHIMVGLILPGIVILSCYCIIISKLS-HSKGHQ-KRKALKTTVILI    242
                  ||:||||.||.:|||::||:|||||||||| :|||.. ||||||||||||
sequence 2:   199 VFRFQHIFVGFLLPGLIILTCYCIIISKLSKNSKGQALKRKALKTTVILI    248

sequence 1:   243 LAFFACWLPYYVGISIDSFILLEVIKQGCEFESVVHKWISITEALAFFHC    292
                  |.||.|||||..||.:|:.::|.||...|..|..:.|||..|||||:|||
sequence 2:   249 LCFFICWLPYCAGILVDTLVMLNVISHTCFLEQGLEKWIFFTEALAYFHC    298

sequence 1:   293 CLNPILYAFLGAKFKSSAQHALNSMSRGSSLKILSKGKRGGHSSVSTESE    342
                  |||||||||||.||..||::||:..|| ||.|:|:| |||..||||||||
sequence 2:   299 CLNPILYAFLGVKFSKSARNALSISSR-SSHKMLTK-KRGPISSVSTESE    346

sequence 1:   343 SSSFHSS                                               349
                  |||..||
sequence 2:   347 SSSVLSS                                               353
```

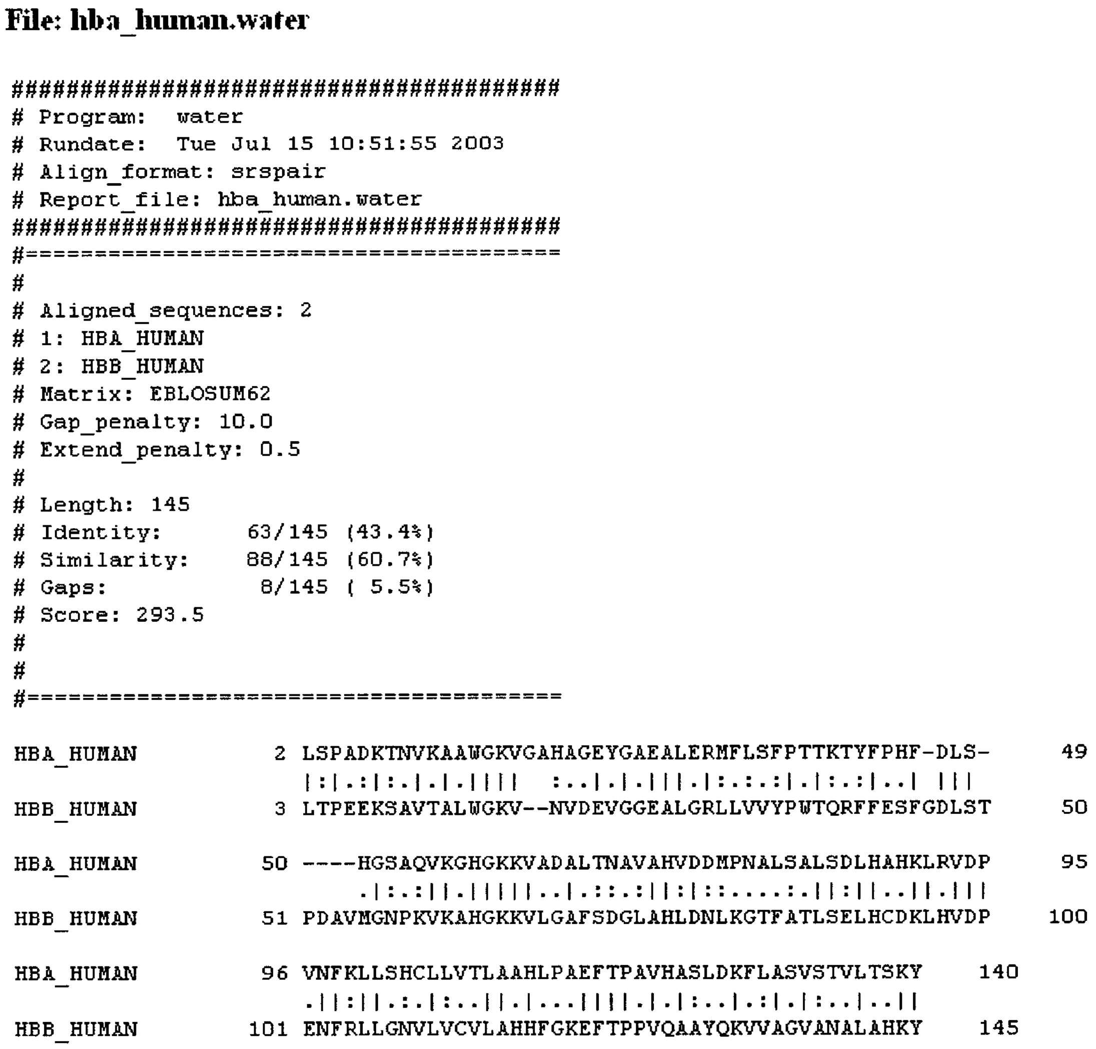
6.3.3 Test Output for aligning Didelphis Virginiana and Dasypus Novemcinctus

```
Smith-Waterman local alignment.
Gap opening Penalty  : 10.0
Gap extension Penalty: 0.5
Output file name     : Alignment_output.txt
Alignment length:   781
Gaps:               81
Identity:          568/781 (72.7%)
Similarity:        568/781 (72.7%)
score     :        2072.0


sequence 1:     1 GCAAGTTTCCGCTAC-CCAGTGAGAATGCCCTTTAAGTCTTATAAATTAA     49
                  ||||||.||.|| || ||||||||||||||||.|||.|||||||.|
sequence 2:     1 GCAAGTATCAGC-ACACCAGTGAGAATGCCCTCTAACTCTTATAGA----     45

sequence 1:    50 GCAAAAGGAGCTGGTATCAGGCACACAAAATGTAGCCGATAACACCTTGC     99
                  .||||||||||..|.||||.|.||||
sequence 2:    46 TCAAAAGGAGCAAGCATCAAGTACAC------------------------     71

sequence 1:   100 TTTACCACACCCCCACGGAGACAGCAGTGATTAAAATTAAGC-AATAAA    148
                                      |||||    ..|.|.|.| ||| .||||.
sequence 2:    72 --------------------ACAGC----CCTTACAGT-AGCTCATAAC     95

sequence 1:   149 CGAAAGTTTGACTAAGTCATAATTTA-CATTAGGGTTGGTCAATTTCGTG    197
                  ||||||.||||||||||  ||..||| .||.||||||||.|||||||||
sequence 2:    96 CGAAAGCTTGACTAAGT--TATGTTATTATAAGGGTTGGTAAATTTCGTG    143

sequence 1:   198 CCAGCCACCGCGGTCATACGATTAACCCAAATTAATAAATAACGGCGTAA    247
                  |||||.|||||||||||||||||||||||||||||..||.||||||||
sequence 2:   144 CCAGCAACCGCGGTCATACGATTAACCCAAATTAATAGTTATCGGCGTAA    193

sequence 1:   248 AGAGTGTTTAAGTTATATACAAAAATAAAGTTAATAATTAACTAAACTGT    297
                  ||.|||||||||..|..|| .|.||||.||||||....|.||||..||||
sequence 2:   194 AGCGTGTTTAAGACACCTA-GACAATAGAGTTAAACCCTTACTACGCTGT    242
sequence 1:   298 AGCACGTTCTAGTTAATATTA-AAATACAT-AATAAAAATGACTTTAATA    345
                  |..|.|...||| ||..|..| |||..|.| ||..|||.|||||.||||.
sequence 2:   243 AAAAAGCCTTAG-TAGGACCATAAACCCTTCAACGAAAGTGACTCTAATT    291

sequence 1:   346 TCACCGACTACACGAAAACTAAGACACAAACTGGGATTAGATACCCCACT    395
                  |..|.||||||||||.|.|||.|||.||||||||||||||||||||||||
sequence 2:   292 TATCTGACTACACGATAGCTAGGACCCAAACTGGGATTAGATACCCCACT    341

sequence 1:   396 ATGCTTAGTAATAAACTAAAATAATTTAACAAACAAAATTATTCGCCAGA    445
                  ||||.|||...|||||||||| .|.||.||||||||||.|.|||||||||
sequence 2:   342 ATGCCTAGCCCTAAACTAAAA-CAGTTCACAAACAAAACTGTTCGCCAGA    390

sequence 1:   446 GAACTACTAGCAATTGCTTAAAACTCAAAGGACTTGGCGGTGCCCTAAAC    495
                  |.|||||||||||..||||||||||||||||||||||||||||..||.|.
sequence 2:   391 GTACTACTAGCAACAGCTTAAAACTCAAAGGACTTGGCGGTGCTTTACAT    440

sequence 1:   496 CCACCTAGAGGAGCCTGTTCTATAATCGATAAACCCCGATAAACCAGACC    545
                  ||..|||||||||||||||||||||||||||||||||||||.|||..|||
sequence 2:   441 CCTTCTAGAGGAGCCTGTTCTATAATCGATAAACCCCGATATACCTCACC    490

sequence 1:   546 TTATCTTGCCAATACAGCCTATATACCGCCATCGTCAGCTAACCTTTAAA    595
                  ....|||||.|||||||||||||||||||||||.|||||..|||.|...|
sequence 2:   491 ACCCCTTGCTAATACAGCCTATATACCGCCATCTTCAGCAGACCCTAGTA    540

sequence 1:   596 AAGAATTACAGTAAGCAAAAT--CATACAACATAAAAACGTTAGGTCAAG    643
                  |.|.|..|||||.||||.|||  |||||   |||||.|||||||||||||
sequence 2:   541 AGGCACCACAGTGAGCACAATAACATAC---ATAAAGACGTTAGGTCAAG    587

sequence 1:   644 GTGTAGCATATGATAAGGAAAGTAATGGGCTACATTCTCTACTATAGAGC    693
                  |||||||.||||....||.|||.|||||||||||||.||||.||.|||||
sequence 2:   588 GTGTAGCTTATGGGGTGGGAAGAAATGGGCTACATTTTCTAATAAAGAGC    637

sequence 1:   694 ATA-ACGAATCATATTATGAAACTAAAATGCTT--GA---AGGAGGATTT    737
                  |.| ||.||..|..|.|||||||   |||  ||  ||   |||.||||||
sequence 2:   638 AAATACAAAAAACTTAATGAAAC---AAT--TTAAGACTAAGGTGGATTT    682

sequence 1:   738 AGTAGTAAATTAAGAATAGAGAGCTTAATTG                       768
                  ||||||||..|||.|||||||||.|||..||
sequence 2:   683 AGTAGTAAGCTAAAAATAGAGAGTTTAGCTG                       713
```

6.4 Test Result From Water application in EMBOSS

The following results are obtained by running the online application Water in EMBOSS at the link: http://csc-fserve.hh.med.ic.ac.uk/emboss/water.html.

6.4.1 Test Output for aligning human protein sequences



File: hba_human.water

```
########################################
# Program:  water
# Rundate:  Tue Jul 15 10:51:55 2003
# Align_format: srspair
# Report_file: hba_human.water
########################################
#=======================================
#
# Aligned_sequences: 2
# 1: HBA_HUMAN
# 2: HBB_HUMAN
# Matrix: EBLOSUM62
# Gap_penalty: 10.0
# Extend_penalty: 0.5
#
# Length: 145
# Identity:      63/145 (43.4%)
# Similarity:    88/145 (60.7%)
# Gaps:           8/145 ( 5.5%)
# Score: 293.5
#
#
#=======================================

HBA_HUMAN          2 LSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHF-DLS-     49
                     |:|.:|:.|.|.||||  :..|.|.|||.|:.:.:|.|:.:|..| |||
HBB_HUMAN          3 LTPEEKSAVTALWGKV--NVDEVGGEALGRLLVVYPWTQRFFESFGDLST     50

HBA_HUMAN         50 ----HGSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDP     95
                         .|:.:||.|||||..|.::.:||:|::....:.||:||..||.|||
HBB_HUMAN         51 PDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDP    100

HBA_HUMAN         96 VNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKY     140
                     .||:||.:.|:..||.|...||||.|.|:..|.:|.|:..|..||
HBB_HUMAN        101 ENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKY     145
```

Figure 7: An output file for Water application in EMBOSS

6.4.2 Test Output for aligning Rattus Norvegicus and Cyprinus Carpio

```
Local: , vs ,
Score: 1117.00

,               1         MEFYDHIFFDN.SSDSGSGDFDFD.ELCDLKVSNDFQKIFLPVVY 43
                          || |     || | : ||||:| : | |    : :| :|||| :|
,               1         MEIYTS...DNYSEEVGSGDYDSNKEPCFRDENENFNRIFLPTIY 42

,               44        GIIFVLGIIGNGLVVLVMGFQKKSKNMTDKYRLHLSIADLLFVLT 88
                           |||: ||:|||||:||||:||| ::||||||||||:||||||:|
,               43        FIIFLTGIVGNGLVILVMGYQKKLRSMTDKYRLHLSVADLLFVIT 87

,               89        LPFWAVDAASGWHFGGFLCVTVNMIYTLNLYSSVLILAFISLDRY 133
                          |||||||| : |:|| |||  |::|||:|||||||||||||||||
,               88        LPFWAVDAMADWYFGKFLCKAVHIIYTVNLYSSVLILAFISLDRY 132

,               134       LAVVRATNSQNFRRVLAEKVIYLGVWLPASLLTVPDLVFAKVHDT 178
                          ||:| |||||: |::|||| :|:|||:|| |||:||::|| |
,               133       LAIVHATNSQSARKLLAEKAVYVGVWIPALLLTIPDIIFADVSQG 177

,               179       GMNTICELTYPLQGNTVWKAVFRFQHIFVGFLLPGLIILTCYCII 223
                              ||:  ||   :::|  ||:||||  || :|||::||:|||||
,               178       DGRYICDRLYP...DSLWMVVFQFQHIMVGLILPGIVILSCYCII 219

,               224       ISKLSKNSKGQALKRKALKTTVILILCFFICWLPYCAGILVDTLV 268
                          ||||| :|||   ||||||||||||| || |||||  || :|: :
,               220       ISKLS.HSKGHQ.KRKALKTTVILILAFFACWLPYYVGISIDSFI 262

,               269       MLNVISHTCFLEQGLEKWIFFTEALAYFHCCLNPILYAFLGVKFS 313
                          :| ||   |  |  : |||  |||||:|||||||||||||| ||
,               263       LLEVIKQGCEFESVVHKWISITEALAFFHCCLNPILYAFLGAKFK 307

,               314       KSARNALSISSR.SSHKMLTK.KRGPISSVSTESESSSVLSS    353
                           ||::||:  || || |:|:| |||  |||||||||||  ||
,               308       SSAQHALNSMSRGSSLKILSKGKRGGHSSVSTESESSSFHSS    349

%id = 65.22                         %similarity = 80.00
Overall %id = 63.74                 Overall %similarity = 78.19
```

6.4.3 Test Output for aligning Didelphis Virginiana and Dasypus Novemcinctus

```
Local: , vs ,
Score: 2120.00

,               1         GCAAGTTTCCGCTAC.CCAGTGAGAATGCCCTTTAAGTCTTATAA 44
                          |||||| || || || |||||||||||||||| ||| |||||||
,               1         GCAAGTATCAGC.ACACCAGTGAGAATGCCCTCTAACTCTTATAG 44

,               45        ATTAAGCAAAAGGAGCTGGTATCAGGCACACAAAATGTAGCCGAT 89
                          ||    ||||||||||  | |||| | |||
,               45        AT....CAAAAGGAGCAAGCATCAAGTACA............... 70

,               90        AACACCTTGCTTTACCACACCCCCACGGGAGACAGCAGTGATTAA 134
                                         |||| |||                  |||
,               71        ...............CACAGCCC..................TTAC 82

,               135       AATTAAGC.AATAAACGAAAGTTTGACTAAGTCA..TAATTTACA 176
                          | |  |||  |||| |||||| |||||||||| |  | |||   |
,               83        AGT..AGCTCATAACCGAAAGCTTGACTAAGTTATGTTATT...A 122
,          177    TTAGGGTTGGTCAATTTCGTGCCAGCCACCGCGGTCATACGATTA 221
                  | ||||||||| ||||||||||||||| ||||||||||||||||||
,          123    TAAGGGTTGGTAAATTTCGTGCCAGCAACCGCGGTCATACGATTA 167

,          222    ACCCAAATTAATAAATAACGGCGTAAAGAGTGTTTAAGTTATATA 266
                  |||||||||||||  || |||||||||| |||||||||      |
,          168    ACCCAAATTAATAGTTATCGGCGTAAAGCGTGTTTAAG......A 206

,          267    CA.....AAAATAAAGTTAATAATTAACTAAACTGTAGCACGTTC 306
                  ||     | |||| ||||||    | ||||  |||||  |
,          207    CACCTAGACAATAGAGTTAAACCCTTACTACGCTGTAAAA..... 246

,          307    TAG..TTAATATTAAAATACA.....TAATAAAAATGACTTTAAT 344
                   ||  ||| ||  |  ||| |      ||  ||| ||||| ||||
,          247    .AGCCTTAGTAGGACCATAAACCCTTCAACGAAAGTGACTCTAAT 290

,          345    ATCACCGACTACACGAAAACTAAGACACAAACTGGGATTAGATAC 389
                   |  | |||||||||| | ||| ||| ||||||||||||||||||
,          291    TTATCTGACTACACGATAGCTAGGACCCAAACTGGGATTAGATAC 335

,          390    CCCACTATGCTTAGTAATAAACTAAAATAATTTAACAAACAAAAT 434
                  |||||||||| |||   ||||||||||  | || ||||||||||
,          336    CCCACTATGCCTAGCCCTAAACTAAAA.CAGTTCACAAACAAAAC 379

,          435    TATTCGCCAGAGAACTACTAGCAATTGCTTAAAACTCAAAGGACT 479
                  | |||||||||| |||||||||||  |||||||||||||||||||
,          380    TGTTCGCCAGAGTACTACTAGCAACAGCTTAAAACTCAAAGGACT 424

,          480    TGGCGGTGCCCTAAACCCACCTAGAGGAGCCTGTTCTATAATCGA 524
                  |||||||||  || | ||  |||||||||||||||||||||||||
,          425    TGGCGGTGCTTTACATCCTTCTAGAGGAGCCTGTTCTATAATCGA 469

,          525    TAAACCCCGATAAACCAGACCTTATCTTGCCAATACAGCCTATAT 569
                  |||||||||||| |||  |||    ||||| ||||||||||||||
,          470    TAAACCCCGATATACCTCACCACCCCTTGCTAATACAGCCTATAT 514

,          570    ACCGCCATCGTCAGCTAACCTTTAAAAAGAATTACAGTAAGCAAA 614
                  ||||||||| |||||  ||| |   || | |  ||||| |||| |
,          515    ACCGCCATCTTCAGCAGACCCTAGTAAGGCACCACAGTGAGCACA 559

,          615    AT..CATACAACATAAAAACGTTAGGTCAAGGTGTAGCATATGAT 657
                  ||  |||   ||||||| |||||||||||||||||||| ||||
,          560    ATAACAT...ACATAAAGACGTTAGGTCAAGGTGTAGCTTATGGG 601

,          658    AAGGAAAGTAATGGGCTACATTCTCTACTATAGAGCATA.ACGAA 701
                    || ||| ||||||||||||| |||| || |||||| | || ||
,          602    GTGGGAAGAAATGGGCTACATTTTCTAATAAAGAGCAAATACAAA 646

,          702    TCATATTATGAAAC....TAAAATGCTTGAAGGAGGATTTAGTAG 742
                    |  | |||||||    ||| |  ||  |||| |||||||||||
,          647    AAACTTAATGAAACAATTTAAGA..CT..AAGGTGGATTTAGTAG 687

,          743    TAAATTAAGAATAGAGAGCTTAATTG                    768
                  |||  ||| ||||||||| |||  ||
,          688    TAAGCTAAAAATAGAGAGTTTAGCTG                    713

%id = 82.61                        %similarity = 82.61
Overall %id = 74.22                Overall %similarity = 74.22
```

6.4.4 Discussion and Summary

From the test results presented in this section, we observed that the test results produced by the generic AlignmentDP application implemented in this report aligned to the result generated by the online application Water in EMBOSS. But the accuracy of the local alignment algorithm instantiated in this report for protein sequences is better than it for nuclei acid sequence. Because we only implement the fundamental local alignment algorithm, which however is different from the local algorithm for the application Water in EMBOSS.

The reason why the algorithm implemented in this report is not the same as the one implemented in Water application in EMBOSS:

- There is no requirement for this report to implement the same algorithm as that of Water application in EMBOSS.
- The detail implementations of the pairwise local alignment algorithm in this report is different from them in EMBOSS, which determines our output will be different from the one generated by water application in EMBOSS. The major difference in these two algorithms is:
    1. In this application, we only implements a simplified affine local algorithm, the gap penalty is calculated based on the open-penalty and the score and trace value in the direct left/right proceeding cell respectively, that is, only one step proceeding gaps is calculated as extension penalty.

2. In EMBOSS, Water application is the really affine local algorithm, it keeps scoring, tracing as well as two additional score, insertion in x-axis (Ix) and insertion in y-axis (Iy), which are then fed into the scoring matrix to take account of the additional types of gap, all the gap penalty is being added to a preceding gap (as opposed to a gap being added to a preceding base), so, the cell with the minimal gap penalty is selected in the corresponding row or column.

- Applications in EMBOSS are evolving. The EMBOSS package when we started our report was the release 2.5.0, but the current release is 2.8.0. There is about one year gap in between. We did not reflect the changes in the pairwise local alignment algorithm instantiated in this report to align to the current Water application in EMBOSS. For example, by comparing the result below generated from the Water application in EMBOSS on November 20th, 2003 with the one in Figure 7, we can see that the Water application is evolving because the similarity and identity are different.

```
Local: , vs ,
Score: 293.50

,              3        LTPEEKSAVTALWGKV..NVDEVGGEALGRLLVVYPWTQRFFESF 45
                        |:| :|: | | ||||  :  | | ||| |: : :| |: :|  |
,              2        LSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHF 46

,              46       GDLSTPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSE 90
                         |||      |: :|| |||||  | :: :||:|::    : ||:
,              47       .DLS.....HGSAQVKGHGKKVADALTNAVAHVDDMPNALSALSD 85

,              91       LHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVA 135
                        ||  || ||| ||:|| : |:  || |   |||| | |:  | :|
,              86       LHAHKLRVDPVNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLA 130

,              136      GVANALAHKY                                    145
                         |:  |  ||
,              131      SVSTVLTSKY                                    140

%id = 45.99                       %similarity = 64.23
Overall %id = 43.15               Overall %similarity = 60.27
```

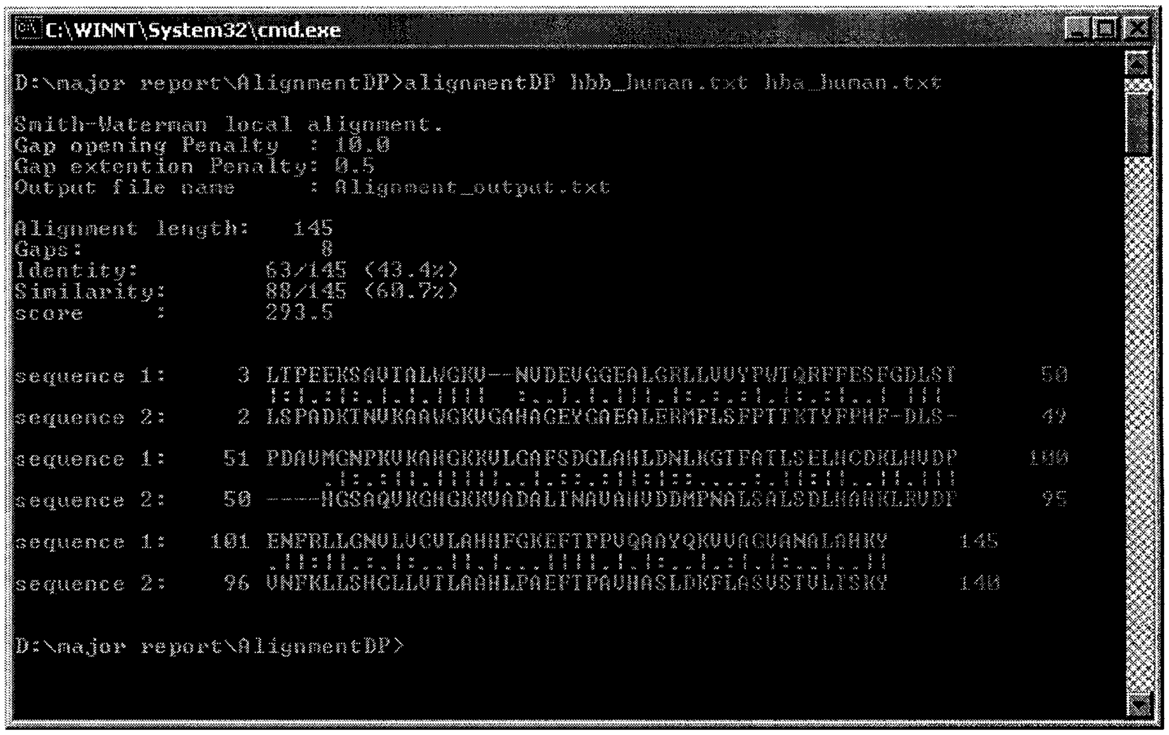
Although the pairwise local alignment algorithm implemented in this report is not the exactly same as the one implemented by the Water application in EMBOSS, the main object of this report is to develop a generic application, defined as a framework, to unify the different implementations of any pariware alignment algorithms and to provide the developer with the capability to develop a pairwise alignment algorithm with little effort. The purpose that we put their experimental result here is to demonstrate:

- The framework AlignmentDP works correctly with the pairwise local alignment algorithm instantiated in this report.
- How easier to instantiate a pairwise local alignment algorithm using this generic application.
- The user can use the same input and output formats as they are in EMBOSS, which property will make the user accept this application easily.

The framework AlignmentDP has also been instantiated by a global algorithm reported in [30]. The test results can be found from section 5 in [30].

6.5 Application Usage

Figure 4 presents a sample display of the implementation of the local alignment algorithm.



```
C:\WINNT\System32\cmd.exe

D:\major report\AlignmentDP>alignmentDP hbb_human.txt hba_human.txt

Smith-Waterman local alignment.
Gap opening Penalty  : 10.0
Gap extention Penalty: 0.5
Output file name     : Alignment_output.txt

Alignment length:   145
Gaps:                 8
Identity:         63/145 (43.4%)
Similarity:       88/145 (60.7%)
score     :       293.5


sequence 1:     3 LTPEEKSAVTALWGKV--NVDEVGGEALGRLLVVYPWTQRFFESFGDLST     50
                  |:|.:|:.|.|.||||  :..|.|.|||.|:.:.:|.|:.:|..| |||
sequence 2:     2 LSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHF-DLS-     49

sequence 1:    51 PDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHVDP    100
                      .|:.:||.|||||..|.::.:||:|::....:.||:||..||.|||
sequence 2:    50 ----HGSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDP     95

sequence 1:   101 ENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKY    145
                  .||:||.:.|:..||.|...||||.|.|:..|.:|.|:..|..||
sequence 2:    96 VNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKY    140


D:\major report\AlignmentDP>
```

Figure 8: A sample session with local algorithm

Now we brief the usage of the application:

- Type the application name alignmentDP at the DOS > prompt, then followed by two text files that each contains a biological sequence in the FASTA format.
- Display one-line description of the application.
- Prompt you for input information: Gap opening Penalty, Gap extension Penalty and Output file name.

6.6 Mandatory Parameters

Many users are familiar with EMBOSS, so to make our tool more user friendly, our application will use the same command line parameters and formats for input and output as the EMBOSS library. The following parameters are needed for running the application at the command line:

- **[-asequence]** sequence Required text file for Sequence A in the FASTA format which should be in the same folder of executable file.
- **[-seqall]** sequence Required text file for Sequence B in the FASTA format which should be in the same folder of executable file.
- **-gapopen** double The gap open penalty is the score taken away when a gap is created. The best value depends on the choice of comparison matrix. The default value is 10.0
- **-gapextend** double The gap extension, penalty is added to the standard gap penalty for each base or residue in the gap. This is how long gaps are penalized. Usually you will expect a few long gaps rather than many short gaps, so the gap extension penalty should be lower than the gap penalty. An exception is where one or both sequences are single reads with possible sequencing errors in which case you would expect many single base gaps. You can get this result by setting the gap open penalty to zero (or very low) and using the gap extension penalty to control gap scoring.
- **-datafile** CMatrix<double> This is the scoring matrix file used when comparing sequences. This file should be in the same folder of executable file. By default it is the file 'EBLOSUM62'.

7 Conclusions and Future work

7.1 General Conclusion

This application is a tool, which generates optimal solution for a simplified affine gap-penalty local alignment algorithm, and with little modification, this application can fit in with any pairwise alignment algorithm.

Its generic design and implementation offers the possibility for the future users to instantiate any alignment algorithm with little effort and only basic knowledge of the C++ language. The implementation provides both robustness and reusability.

This framework is instantiated by a local alignment algorithm in this project. The main advantage of the design is its high reusability. It provides two levels of code reusability:

- Input, output, data storage Matrix and main procedure are defined as object class, so the user does not need to redefine and re-implement them for their algorithm.
- The command interface for pairwise alignment algorithm has been identified and a local alignment algorithm has been instantiate. Most of the methods in this class can be re-used when defining and implementing methods for other alignment algorithms.

7.2 Future Work

The AlignmentDP application is easy to implement, understand and reuse. To make AlignmentDP a more powerful framework, the following jobs need to be carried out for improving the use of the application program:

1. Input: It is better for the application to accept more kinds of data format as input. So far, only text files with raw sequence data are accepted. If it is necessary for the application to integrate with the Emboss environment, it will need to accept some formats, which are generated by other application of Emboss.
2. Interface: For user friendliness, a graphical user interface may be needed.
3. Output: so far AlignmentDP application only supports one optimal solution; some modification of the Alignment class and trace back functionality is needed to support the output of all optimal solutions.
4. Database connectivity: For local alignment, the functionality of connecting to a database should be provided in the future.
5. The algorithm implemented is not exactly the Smith – Waterman algorithm as the one in the EMBOSS library. Function *printOutputFileHeader()* in WaterObjFunc class may need to be modified to prevent confusion.

References

1. H. Austern, Generic Programming and the STL: Using and Extending the C++ Standard Template Library. Addison-Wesley Publishing, 1998.
2. R. Bellman, Dynamic Programming. Princeton Univesity Press, 1957.
3. B. Bergeron, Bioinformatics Computing. Prentice Hall PTR, 2002.
4. D. Bertsekas, Dynamic Programming. Prentice Hall, 1987.
5. L. Cardelli and P. Wegner, On understanding types, data abstraction, and polymorphism. Computing Surveys, 17(4): 471–522, 1985.
6. A. Cockburn and L. Williams, The Costs and Benefits of Pair Programming in Extreme Programming Examined. Addison Wesley, 2001.
7. K. Czarnecki and U. Eisenecker, Generative Programming: Methods, Tools, and Applications. Addison-Wesley Publishing, 2000.
8. R. Durbin, S. Eddy, A. Krogh, and G. Mitchison, Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids. Cambridge, 1999.
9. B. Eckel, Thinking in C++, Volume 1: Introduction to Standard C++. 2000.
10. G. B. Fogel and David W. Corne, Evolutionary Computation In Bioinformatics. Morgan Kaufmann, 2002.
11. O. Gotoh, An improved algorithm for matching biological sequences. J. Mol. Biol., 162, 705-708, 1982.
12. D. Gusfield, Algorithms on Strings, Trees, and Sequences. Cambridge University Press, 1997.
13. S. Henikoff and J. G. Henikoff, Amino acid substitution matrices from protein blocks. Proc. Nat. Acad. Sci. U.S.A, 89: 10915-10919, 1992.

14. N. M. Josuttis, The C++ Standard Library: A Tutorial and Reference. Addison-Wesley, 1999.

15. D. E. Krane and M. L. Raymer, Fundamental Concepts of Bioinformatics. Pearson Benjamin Cummings, 2002.

16. B. Meyer, Object-oriented software construction. Prentice Hall, 1997.

17. W. Miller and E.W. Myers, Sequence comparison with concave weighting functions. Bull. Math. Biol., p. 97--120, 50-2, 1988.

18. D. R. Musser and A. Saini, STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library. Addison-Wesley Publishing, 2001.

19. D. R. Musser and A. A. Stepanov, Algorithm-oriented generic libraries. Software: Practice and Experience, 24(7): 632–642, 1994.

20. E. W. Myers and W. Miller, Optimal alignments in linear-space. Comput. Appl. Biosci. 4, 11-17, 1988.

21. S. B. Needleman and C. D. Wunsch, A general method applicable to the search for similarities in the amino acid sequences of two proteins. J. Mol. Biol. 48, 443-453, 1970.

22. M. Nelson, C++ Programmer's Guide to the Standard Template Library. IDG Books Worldwide, 1995.

23. P. A. Pevzner, Computational molecular biology: an algorithmic approach. MIT Press, 2000.

24. D. Sankoff and J. B. Kruskal, Time warps, string edits and macromolecules: the theory and practice of sequence comparison. Addison Wesley, 1983.

25. R.M. Schwartz and M.O. Dayhoff, Matrices for detecting distant relationships. Atlas of Protein Sequence and Structure. 5 suppl. 3:353-358, 1978.

26. P. H. Sellers, On the theory and computation of evolutionary distances. SIAM J. Appl. Math. 26, 787-793, 1974.

27. T.F. Smith and M.S. Waterman, Identification of Common Molecular Subsequences. J. Mol. Biol., 147, pp. 195-197, 1981.

28. M. S. Waterman and M. Eggert, A new algorithm for best subsequence alignments with application to tRNA-rRNA comparisons. J. Mol. Biol. 197, 723–728, 1987.

29. M. S. Waterman, T. F. Smith and W. A. Beyer, Some biological sequence metrics. Advances in Mathematics, 20, 367-387, 1976.

30. Y. Zhang, Generic C++ Implementation of Pairwise Sequence Alignment: Instantiation for Global Alignment, Major Report, Department of Computer Science, Concordia University, 2003.