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

Yan Zhang

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: 0-612-83924-9
Our file Notre référence
ISBN: 0-612-83924-9

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the major report prepared

By: **Yan Zhang**

Entitled: **Generic C++ Implementation of Pairwise Sequence Alignment: Instantiation for Global Alignment**

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

Clement Lam ______ Examiner

[illegible] ______ Supervisor

Approved [illegible]
Chair of Department or Graduate Program Director

SEP 25 2003 20 [illegible]
Dr Nabil Esmail, Dean
Faculty of Engineering and Computer Science

ABSTRACT

Generic C++ Implementation of Pairwise Sequence Alignment: instantiation for global alignment

Yan Zhang

Sequence comparison and alignment is a central problem in computational biology. Pairwise sequence alignment of protein or nucleic acid sequences is the foundation upon which most bioinformatics tools are built. EMBOSS has several pairwise alignment algorithms implemented in C, but they are quite dependable on the algorithms.

Recently, generic programming has emerged as a programming paradigm capable of providing high levels of performance and re-usability in the presence of programming abstractions. Generic programming is about representing domains as collections of highly general and abstract components, which can be combined in vast numbers of ways to yield very efficient concrete programs.

Pairwise alignment algorithms are optimization problems; from the view of a design, all the pairwise alignment shares the most common entities.
This report designs and implements an application for pairwise Sequence Alignment using a generic programming approach in C++. Object-oriented programming principles

are used for design; generic parameter and parameterized components mechanism in the C++ lanuage are used for implementing this application, from which user can derive instantiations for any pairwise alignment algorithms of interest, this part can be considered as a framework in this application. Semi-global alignment algorithm is instantiated in this project.

This implementation offers the possibility for the programmer to instantiate any kind of pairwise alignment algorithm with little efforts and basic knowledge of the C++ language. The implementation provides both robustness and re-usability properties.

ACKNOWLEDGEMENTS

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

I am pleased to express my gratitude to Prof. Greg Butler, who encouraged my interest in the subject of generic C++ implementation of Bioinformatics Algorithms, for acceptance to be my supervisor, his technical advise, constant help and critical remarks through this work. He patiently guided me with his knowledge about software technology for complex software systems, especially on software architectures, object-oriented design, and software reuse.

I am greatly indebted to my partner, Xiao Yang, in making our shared major report into reality. Without her input, persistence, help and collaboration, this would not have been possible. This report is dedicated to my family for their help, love and encouragement to achieve excellence. The author's sincere thanks are due to the support by my company, Motorola Software Center, Montreal, Quebec, Canada.

Contents

LIST OF FIGURES .... **VII**

LIST OF TABLES .... **VIII**

1 INTRODUCTION **1**

1.1 BACKGROUND 1
1.2 OBJECTIVE OF STUDY 3
1.3 PROJECT SCOPE 4
1.4 JOINT EFFORT 5
1.5 OUTLINE OF THE REPORT 5

2 PAIRWISE SEQUENCE ALIGNMENT **6**

2.1 SEQUENCE COMPARISON FUNDAMENTALS 6
2.1.1 Introduction *6*
2.1.2 Sequence Alignment vs. Similarity Searching *7*
2.1.3 Homology vs. Similarity *7*
2.1.4 Similarity vs. Distance *8*
2.2 WEIGHT MATRICES FOR SEQUENCE SIMILARITY SCORING 9
2.2.1 The Scoring Model *9*
2.2.2 The Substitution Matrices *10*
2.3 REALISTIC GAP MODELS 14
2.4 DYNAMIC PROGRAMMING 16
2.4.1 Overview of Dynamic Programming *16*
2.4.2 Characteristics of Dynamic Programming Problems *18*
2.5 VARIATIONS OF PAIRWISE ALIGNMENT 19
2.5.1 Pairwise Alignment Algorithm *19*
2.5.2 Global Alignment: Needleman – Wunsch algorithm *20*
2.5.3 Semi-global Alignment Algorithm implemented in the report *24*

3 GENERIC PROGRAMMING **26**

3.1 GENERIC PROGRAMMING AND STL 26
3.1.1 Algorithms *27*
3.1.2 Iterators *27*
3.1.3 Adaptors *29*
3.1.4 Functors *29*
3.2 GENERICITY AND POLYMORPHISM 29
3.2.1 Parametric polymorphism *29*
3.2.2 Subtype polymorphism *30*
3.2.3 C++ support for both polymophism *31*
3.3 USING GENERIC PROGRAMMING FOR APPLICATION DEVELOPMENT 31

4 GENERIC C++ IMPLEMENTATION **37**

4.1 SYSTEM DESIGN IN UML 38
4.2 DESIGN AND C++ IMPLEMENTATION 39

4.3 AN INSTANTIATION: SEMI-GLOBAL ALIGNMENT ALGORITHM 42

5 EXPERIMENTAL RESULTS 45

5.1 OUTLINE OF THE EXPERIMENTS 45
5.2 EXPERIMENT DATA 45
5.3 RESULT PRESENTATION 46
5.3.1 Output: Hba_human and Hba_human 46
5.3.2 Output: Rattus and Cyprinus 47
5.3.3 Output: Didelphis and Dasypus 47
5.4 RESULT COMPARISON WITH NEEDLE APPLICATION IN EMBOSS 49
5.4.1 Output: Hba_human and Hba_human 49
5.4.2 Output: Rattus and Cyprinus 49
5.4.3 Output: Didelphis and Dasypus 50

6 USER MANUAL 52

6.1 FUNCTION USAGE 52

7 CONCLUSIONS AND RECOMMENDATION 54

7.1 CONCLUSION 54
7.2 RECOMMENDATIONS FOR FUTURE WORKS: 55

REFERENCES 57

LIST OF FIGURES

FIGURE 1: FINDING THE SCORE OF EACH CELL IN THE MATRIX 21
FIGURE 2: AN EXAMPLE OF NEEDLEMAN – WUNSCH ALGORITHM 22
FIGURE 3: DESIGN AN APPLICATION AS A GENERIC PROGRAM 33
FIGURE 4: DESIGN AN APPLICATION AS AN OBJECT-ORIENTED PROGRAM THAT USE THE GENERIC COMPONENTS (2A) 35
FIGURE 5: DESIGN AN APPLICATION AS AN OBJECT-ORIENTED PROGRAM THAT USE THE GENERIC COMPONENTS (2B) 36
FIGURE 6: CLASS DIAGRAM IN UML 38
FIGURE 7: A SAMPLE OUTPUT FROM NEEDLE APPLICATION IN EMBOSS 49
FIGURE 8: A SAMPLE SESSION WITH NEEDLEMAN-WUNSCH ALGORITHM 52

List of Tables

Table 1 The log odds matrix for PAM250 (multiplied by 10) 13
Table 2 The log odds matrix for BLOSUM62 14

1 Introduction

1.1 Background

Bioinformatics has never been as popular as today. The genomic revolution is closely linked with advances in computer science. Each genome project results in a rapid collection of data in digital form, whose meaning is left for later interpretation. Computers not only store the data, but also are essential for their interpretation. Problems such as gene identification and expression, RNA and protein folding and structure determination, and metabolic pathway analysis carry their own computational demands [9].

Today bioinformatics is an applied science. We use computer programs to make inferences from the data archives of modern molecular biology, to make connections among them, and to derive useful and interesting predictions.

Finding differences between sequences is often equivalent to finding similarities between these sequences. For example, if edit operations are limited to insertions and deletions (no substitutions), the edit distance problem is equivalent to the Longest Common Subsequence (LCS) Problem. Although the LCS Problem captures most algorithmic aspects of sequence comparison, biologists prefer using alignments for DNA and protein sequence comparison [17].

Sequence alignment is a method to inferring homology and function. In a very real sense, any alignment between two or more nucleotide or amino acid sequences represents an explicit hypothesis regarding the evolutionary history of those sequences. As a direct result, comparisons of related protein and nucleotide sequences have facilitated many recent advances in the understanding of the information content and function of genetic sequences. For this reason, techniques for aligning and comparing sequences, and for searching sequence databases for similar sequences, have become cornerstones of bioinformatics [12].

In the early papers on sequence alignment, scientists attempted to find the similarity between entire strings V and W, i.e., global alignment. Needleman and Wunsch [15] developed the most basic algorithm to align two sequences in 1970. The algorithm is a simple and beautiful way to find an alignment that maximizes a particular score. This is meaningful for comparisons between members of the same protein family. In many biological applications, the score of alignment between sub-strings of V and W may be larger than the score of alignment between the entire strings. This problem is known as the local alignment problem. Smith and Waterman [18] proposed a clever modification of dynamic programming that solves the local alignment problem.

A wide range of software including both programs directly related to compute sequence alignments of biological sequences and utility programs is developed in recent years and available on the Internet. Many suites of software are now available that present integrated sets of tools for sequence analysis. GCG is used by molecular biologists worldwide for comprehensive sequence analysis, which is based on published algorithms from the fields of mathematical and computational biology. But GCG is commercial and very expensive. Alternative software to GCG, EMBOSS (The European Molecular Biology Open Software Suite) is a new, free Open Source software analysis package specially developed for the needs of the molecular biology user community. The software automatically copes with data in a variety of formats and even allows for transparent retrieval of sequence data from the web. Also, as extensive libraries are provided with the package, it is a platform to allow other scientists to develop and release software in the true open source spirit. EMBOSS also integrates a range of currently available packages and tools for sequence analysis into a seamless whole. EMBOSS breaks the historical trend towards commercial software packages.

1.2 Objective of Study

Although, there are already several pairwise alignment algorithm have been implemented in C in EMBOSS, all of them are quite dependable on the algorithm. These approaches have, at least two drawbacks. First, one has to implement method from scratch for any pairwise alignment algorithm of interest. Second, it is difficult to introduce even small changes in the code since it would require the modification of most of the implementation.

This project designs and implements a generic framework in C++ for pairwise alignments. The motivation for this project is two fold: first, to enable the user to instantiate this framework for any kind of pairwise alignment algorithm with little efforts by re-using several components, second, to give flexibility at implementing the methods. To achieve this property, this application is carefully designed by identifying the common entities of pairwise alignment method and, of course, due to generic programming technology.

1.3 Project Scope

Prof. G. Butler supervised the research work for this project. The work-study was started in September 2002. The procedure to develop this project is structured in the following way:

1. What is bioinformatics? Especially in understanding the basic concepts of this new applied science.
2. Focus on Pairwise sequence alignment algorithms. Searching and reading papers and books on this area.
3. Understanding the generic programming.
4. Analyze and refine the basic requirement for this project. Exploring the EMBOSS libraries.
5. Project design, source coding in C++ by using template to with special emphasis on generic library classes.
6. Integrate the program and do system testing.

7. Benchmark testing and result analysis.
8. A deliverable project package with manual, sample protein sequence file and test results.
9. Make a conclusion for this research work and provide recommendations for future works.

1.4 Joint Effort

This project is a joint project, is completed by cooperating with Xiao Yang. We share the understanding of Biochemistic domain technology and discussion on the methodology of object-oriented programming and generic programming. We worked together in following components of this project:

- Framework design and implementation
- Objective function common interface design

Following contents may overlap in our major report:

- Design and implementation of Framework
- Experiment data
- Part of conclusion and recommendation for the future job

We implement global alignment algorithm and local alignment separately.

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 mechanism used in this report. Chapter 4 covers design and implementation of this application with object-oriented design methods and C++ generic mechanisms. Chapter 5 presents the experimental results. Chapter 6 briefly describes this application function usage. Finally, Chapter 7 presents the conclusion of the report and suggests future works.

2 Pairwise Sequence Alignment

2.1 Sequence Comparison Fundamentals

2.1.1 Introduction

In molecular biology, proteins and DNA can be similar with respect to their function, their structure, or their primary sequence of amino or nucleic acids. The general rule is that sequence determines shape, and shape determines function. So when we study sequence similarity, we eventually hope to 1) discover or validate similarity in shape and function (assign functions to unknown proteins), 2) determine relatedness of organisms, 3) identify structurally, functionally, and evolutionally similarities and 4) make predictions about the 3D structure. Since similarity may be an indicator of homology, it provides some insight into function or gene identification.

Sequence comparison and alignment is a central problem in computational biology. The most basic sequence analysis task is to ask if two sequences are related. For example, it's generally accepted that if two sequences are in alignment, part or the entire pattern of nucleotides or amino acids match, then they are similar and may be homologous. Another heuristic is that if the sequence of a protein or other molecule significantly matches the sequence of a protein with a known structure and function, then the molecules may share structure and function.

2.1.2 Sequence Alignment vs. Similarity Searching

Sequence alignment and similarity searching are widely employed in the sequence analysis task, but they focus on the different aspects of sequence analysis. An alignment between two sequences is simply a pairwise match between the characters of each sequence. A true alignment of nucleotides or amino acid sequences is one that reflects the evolutionary relationship between two or more sequences that share a common ancestor. So Sequence alignment is a search for a consensus sequence. As a result, it produces a score, but the desired result is the sequence. On the other hand, similarity searching such as BLAST, FASTA and SSEARCH, is to search for homologies in order to elucidate the function of an unknown protein. At the end, it produces alignments, but the desired result is the score.

2.1.3 Homology vs. Similarity

All of the patterns we are trying to find in biology domain are based on evolution. 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.

Similarity is an observable quantity that may be expressed as degrees of identity or some other measure. Similarity has both a quantitative and a qualitative aspect: A similarity measure gives a quantitative answer, saying that two sequences show a certain degree of similarity. Sequence similarity is a repeatable and objective measurement.

Homology is not a matter of degree, at any given position in an alignment, sequences and individual positions either share a common ancestor or they do not. In contrast, the overall similarity between two sequences can be described as a fractional value. We also want the sequence similarity measurement to contribute to homology; the sequence similarity should measure the maximum extent of sequence conservation or the minimum extent of sequence divergence.

An alignment is a mutual arrangement of two sequences, which is a sort of qualitative answer; it exhibits where the two sequences are similar, and where they differ. An optimal alignment, of course, is one that exhibits the most correspondences, and the least differences.

2.1.4 Similarity vs. Distance

Two ways are used to quantify similarity of two sequences: A similarity measure is a function that associates a numeric value with a pair of sequences, with the idea that a higher value indicates greater similarity. Beyond this, similarity measures vary widely, and care must be taken when interpreting similarity measures.

The notion of distance is somewhat dual to similarity. It treats sequences as points in a metric space. A distance measure is a function that also associates a numeric value with a pair of sequences, but with the idea that the larger the distance, the smaller the similarity, and vice versa. Distance measures usually satisfy the mathematical axioms of a metric. In particular, distance values are never negative.

In most cases, distance and similarity measures are interchangeable in the sense that a small distance means high similarity, and vice versa. Sometimes similarity measures are a little more flexible.

This scheme is known as the Levenshtein Distance [13], also called unit cost model. Its predominant virtue is its simplicity. In general, more sophisticated cost models must be used. For example, replacing an amino acid by a biochemical similar one should weight less than a replacement by an amino acid with totally different properties.

2.2 Weight Matrices for Sequence Similarity Scoring

2.2.1 The Scoring Model

The key issues of aligning the sequences are: (1) what sorts of alignment should be considered; (2) the scoring system used to rank alignments; (3) the algorithm used to find optimal scoring alignments; and (4) the statistical methods used to evaluated the significance of an alignment score. So we must give careful thought to the scoring system we use to evaluate alignments [8].

When we compare sequences, we are looking for evidence that they have diverged from a common ancestor by a process of mutation and selection. The basic mutational processes that are considered are substitutions, which change residues in a sequence, and insertions and deletions, which add or remove residues. Insertions and deletions are together referred to as gaps. Natural selection has an effect on this process by screening the mutations, so that some sorts of change may be seen more than others.

The total score we assign to an alignment will be a sum of terms for each aligned pair of residues, plus terms for each gap. In the probabilistic interpretation, this will correspond to the logarithm of the relative likelihood that the sequences are related, compared to being unrelated. Informally, we expect identities and conservative substitutions to be likely in alignment than expected by chance, and contributes positive score terms; and non-conservative changes are expected to be observed less frequently in real alignments then expected by chance, and so these contribute negative score terms.

Using a scoring scheme corresponds to an assumption that we can consider mutations at different sites in a sequence to have occurred independently (treating a gap of arbitrary length as a single mutation). All the sequence alignment algorithms for finding optimal alignments depend on such a scoring scheme.

Scoring matrices or substitution matrices appear in all analysis involving sequence comparison. The choice of matrix can strongly influence the outcome of the analysis. Scoring matrices implicitly represent a particular theory of evolution. Understanding theories underlying a given scoring matrix can aid in making a proper choice.

2.2.2 The Substitution Matrices

We need to score terms for each aligned residue pair. Given a pair of aligned sequences, we want to assign a score to the alignment that gives a measure of the relative likelihood that the sequence are related as opposed to being unrelated.

Protein structure and function are surprisingly resistant to polypeptide substitution, to the degree that the substitutions don't alter the chemistry of the protein. Substitutions are common over large expanses of time and from one species to the next. In many cases, the substitution of polypeptides through evolution can be predicted. In this way, a matrix of likely polypeptide substitutions can be constructed. The amino acids are listed across the top and side of a matrix, typically using the amino acid code letters. At each intersection, the matrix is filled with a score that reflects how often one polypeptide would have been paired with the other in an alignment of related protein sequences. This is known as a score matrix or a substitution matrix. An underlying assumption is that this association is symmetrical, in that either polypeptide can be substituted for the other [4].

Several criteria can be considered when devising a scoring matrix for amino acid sequence alignments. Two of the most common are based on observed chemical/physical similarity and observed substitution frequencies.

A more common method for deriving scoring matrices is to observe the actual substitution rates among the various amino acid residues in nature. If a substitution between two particular amino acids is observed frequently, then positions in which these two residues that are aligned are scored favourably. Likewise alignments between residues are not observed to interchange frequently in natural evolution are penalized. One commonly used scoring matrix based on observed substitution rates is the Point Accepted Mutation (PAM) matrix [7]. The scores in a PAM matrix are computed by observing the substitutions that occur in alignments between similar sequences. First, an alignment is constructed between sequences with very high (usually > 85 %) identity. Next, the **relative mutability**, m_j, for each amino acid, j, is computed. The relative mutability is simply the number of times the amino acid was substituted by any other amino acid. Next, A_{ij}, the number of times amino acid j was replaced by amino acid i, is tallied for each amino acid pair i and j. Finally, the substitution tallies (the A_{ij} values) are divided by the relative mutability values, normalized by the frequency of occurrence of each amino acid, and the log of each resulting value is used to compute the entries, R_{ij}, in the PAM – 1 matrix. The resulting matrix is sometimes referred to as a **log odds matrix**, since the entries are based on the log of the substitution probability for each amino acid [4].

The normalization of each matrix entry is done such that the PAM matrix represents substitution probabilities over a fixed unit of evolutionary change. For PAM – 1, this unit is 1 substitution (accepted point mutation) per 100 residues, or one **PAM unit**. The particular PAM matrix is most appropriate for a given sequence alignment depends on the length of the sequences and on how closely the sequences are believed to be related. It is most appropriate to use the PAM – 1 matrix to compare sequences that are closely related, whereas the PAM – 1000 matrix might be used to compare sequences with very distant relationships. In practice, the PAM – 250 matrixes is a commonly used compromise (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

Another popular scoring matrix, the BLOSUM matrix [11], is also derived by observing substitution rates among similar protein sequences. For BLOSUM, un-gapped alignments of related proteins are grouped using statistical clustering techniques, and substitution rates between the clusters are calculated. This clustering approach helps to avoid some statistical problems that can occur when the observed substitution rate is very low for a particular pair of amino acids. Like the PAM matrices, various BLOSUM matrices can be constructed to compare sequences with different degrees of relatedness. The significance of the numbering for BLOSUM matrices, however, can be thought of as the inverse of the PAM numbers. In other words, lower numbered PAM matrices are appropriate for comparing more closely related sequences; while lower numbered BLOSUM matrices are used for more distantly relate sequences. As a rule of thumb, a BLOSUM – 62 matrix is appropriate for comparing sequences of approximately 62 % sequence similarity, while a

BLOSUM – 80 matrix is more appropriate for sequences of about 80 % similarity (see Table 2).

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

First, these matrices are static; furthermore, these matrices aren't mere mathematical constructs designed simply to facilitate computational sequence alignment, but they reflect the biology of the molecules represented by the sequences [4].

2.3 Realistic Gap Models

Sometimes, from an evolutionary point of view, it is more realistic to assume that nature frequently deletes or inserts entire sub-strings as a unit, as opposed to deleting or inserting individual nucleotides. A gap in an alignment is defined as a continuous sequence of spaces in one of the rows. (We have treated the gap symbol "-" as yet another character, denoting an individual insertion or deletion.) It is natural to assume that the score of a gap consisting of x spaces is not just the sum of scores of x index, but rather a more general function.

In the simple case, where no internal gaps are allowed, aligning two sequences is simply a matter of choosing the starting point for the shorter sequence. (Sometimes we want no-gap alignments. For example, in a family of proteins there may be a strongly conserved subunit, which is the site of some protein-protein interaction. Any deletion/insertion in the chain of amino acids would be likely to destroy its biochemical function. Such regions we want to align using matches/replacements only.) But when we add gaps into compared sequences to reflect the occurrence of insertion and deletion, the possibility of insertion and deletion events significantly complicates sequence alignments by vastly increasing the number of possible alignments between two or more sequences. In scoring an alignment that includes gaps, the gap penalty must be included in the scoring function [8].

We expect to penalize gaps. The standard cost associated with a gap of length $\boldsymbol{g}$ is given either by a linear score

$$\gamma(\boldsymbol{g}) = -\boldsymbol{gd}$$

or an affine score

$$\gamma(\boldsymbol{g}) = -\boldsymbol{d} - (\boldsymbol{g} - 1)\boldsymbol{e}$$

where $\boldsymbol{d}$ is called the ***gap-open*** penalty and $\boldsymbol{e}$ is called the ***gap-extension*** penalty. The gap-extension penalty $\boldsymbol{e}$ is usually set to something less than the gap-open penalty $\boldsymbol{d}$. It means that we charge a certain set-up cost for introducing a new gap, whereas extending an existing gap is less expensive. This is desirable when gaps of a few residues are expected almost as frequently as gaps of a single residue.

2.4 Dynamic Programming

2.4.1 Overview of Dynamic Programming

One way to be certain that the solution to a sequence alignment is the best alignment possible is to try every alignment, introducing one or more gaps at every position, and computing an alignment score based on aligned character pairs and inexact matches. However, the computational overhead of evaluating all possible alignments of one sequence against another grows exponentially with the length of the two sequences. For example, consider two modest-sized sequences of 100 and 95 nucleotides. If we were to devise an algorithm that computed and scored all possible alignments, our program would have to test ~55 million possible alignments, just to consider the case where exactly five gaps are inserted into the shorter sequence. As the lengths of the sequences grow, the number of possible alignments to search quickly becomes intractable, or impossible to compute in a reasonable amount of time [12].

We can overcome this problem by using dynamic programming, a method of breaking a problem apart into reasonably sized sub-problems, and using these partial results to compute the final answer.

Dynamic Programming is a useful mathematical technique for making a sequence of interrelated decisions. The mathematical theory of dynamic programming as a mean of solving dynamic optimization problems dated to the early contributions of Bellman [3] and Bertsekas [5]. It provides a systematic, recursive procedure for determining the optimal combination of decisions. In contrast to linear programming, there does not exist a standard mathematical formulation of the dynamic programming program. Rather, dynamic programming is a general approach to problem solving. A certain degree of ingenuity and insight are necessary to develop the appropriate form of the recursion.

Dynamic Programming is a very general programming technique. Dynamic programming, in fact, is a divide-and-conquer method. The three parts to dynamic programming are:

- Recurrence relation - establishes the recursive relation between a problem and smaller instances of the problem. For any recursive relation, the base condition(s) must be specified.
- Tabular computation - use the recurrence relations to compute all partial solutions for the sub-problems. Find the optimal partial solution in each sub-problem.
- Traceback - Find the optimal overall solution by tracing back the path that gave the optimal partial solutions.

S. Needleman and C. Wunsch [15] were the first to apply a dynamic programming approach to the problem of sequence alignment. To bring the power of dynamic programming into the realm of pairwise sequence alignment, consider MaxValue to be the alignment score for pairwise alignment of two sequences. MaxValue takes into account gaps penalties, correct alignments, and imperfect alignment. After the matrix is filled in using the alignment score to determine MaxValue, the highest scoring path is followed back to the beginning of the alignment to define the best alignment of elements in the sequence, including gaps.

Dynamic programming is processor- and RAM- intensive, but the technique of storing intermediate values in a matrix can transform an otherwise intractable problem requiring immense computational capabilities into one that is computationally feasible. Instead of solving one complex CPU- and RAM- intensive problem, the task is decomposed into hundreds or thousands of easily and quickly solved problems.

2.4.2 Characteristics of Dynamic Programming Problems

- The problem can be divided into stages with a policy decision required at each stage.
- Each stage has a number of states associated with it. The number of states may be finite or infinite.
- The effect of the policy decision at each stage is to transform the current state into a state associated with the next stage. The transformation may be deterministic or stochastic.
- Given a current state, an optimal policy for the remaining stages is independent of the policy adopted in the previous stages. This is the Principle of Optimality or the Markovian Property.

- The solution procedure typically begins by finding the optimal policy for each state of the last stage, i.e., we use a backward solution technique. Sometimes a forward procedure makes more sense.
- A recursive relationship identifies the optimal policy for each state at stage n, given the optimal policy for each state in stage n+1 is available. The form of the recursion is problem dependent. Therein lies the "art" of dynamic programming.
- Using the recursive relationship, the solution procedure moves backward (in a backward formulation) stage by stage - each time finding the optimal policy for that stage.

2.5 Variations of Pairwise Alignment

2.5.1 Pairwise Alignment Algorithm

Given a scoring system, we need to have an algorithm for finding an optimal alignment for a pair of sequences, where the sequences have the same length n, there is only one possible global alignment of the complete sequences, but things become more complicated once gaps allowed. There are

$$\binom{2n}{n} = \frac{(2n)!}{(n!)^2} \simeq \frac{2^{2n}}{\sqrt{\pi n}}$$

possible global alignments between two sequences of length n. It is clearly not computationally feasible to enumerate all these, even for moderate values of n. But dynamic programming algorithms are guaranteed to find the optimal scoring alignment or set of alignments. In most cases heuristic methods have also been developed to perform the same type of search. These can be very fast, but they make additional assumptions and will miss the best match for some sequence pairs [8].

Because we introduced the scoring system as a log-odds ratio, better alignments will have higher scores, and so we want to maximize the score to find the optimal alignment. Sometimes score are assigned by other means and interpreted as costs or edit distances, in which case we would seek to minimize the cost of an alignment. Both approaches have been used in the biological sequence comparison literature. Dynamic programming algorithms apply to either case; the differences are trivial exchanges of 'min' for 'max'.

2.5.2 Global Alignment: Needleman – Wunsch algorithm

The problem this project is considering is that of obtaining the optimal global alignment between two sequences, allowing gaps. The dynamic programming algorithm for solving this problem is known in biological sequence analysis as the Needleman-Wunsch algorithm [15], but the more efficient version that we describe was introduced by Gotoh [10].

The idea is to build up an optimal alignment using previous solutions for optimal alignments of smaller subsequences. We construct a matrix F indexed by I and j, one index for each sequence, where the value F(i,j) is the score of the best alignment between the initial segment $x_{1...i}$ of x up to x_i and the initial segment $y_{1...j}$ of y up to y_j. We can build F(i,j) recursively. We begin by initializing F(0, 0) = 0. We then proceed to fill the matrix from top left to bottom right. If F(i-1,j-1), F(i-1,j) and F(i,j-1) are known, it is possible to calculate F(i,j). There are three possible ways that the best score F(i,j) of an alignment up to x_i, y_j could be obtained: x_i could be aligned to y_j, in which case

$F(i,j) = F(i-1,j-1) + s(x_i, y_j)$; or x_i is aligned to gap, in which case $F(i,j) = F(i-1, j) - d$; or

y_j is aligned to gap, in which case F(i,j) = F(i, j-1) – d. The best score up to (i,j) will be the largest of these three options.

Therefore, we have

$$\mathbf{F(i, j) = Max} \begin{cases} \mathbf{F(i\text{-}1,j\text{-}1) + s(x_i, y_j),} \\ \mathbf{F(i\text{-}1, j) - d,} \\ \mathbf{F(i, j\text{-}1) - d} \end{cases} \qquad \mathbf{2.1}$$

This equation is applied repeatedly to fill the matrix of F(i,j) values, calculating the value in the bottom right-hand corner of each square of four cells from one the other three values as in the figure 1.

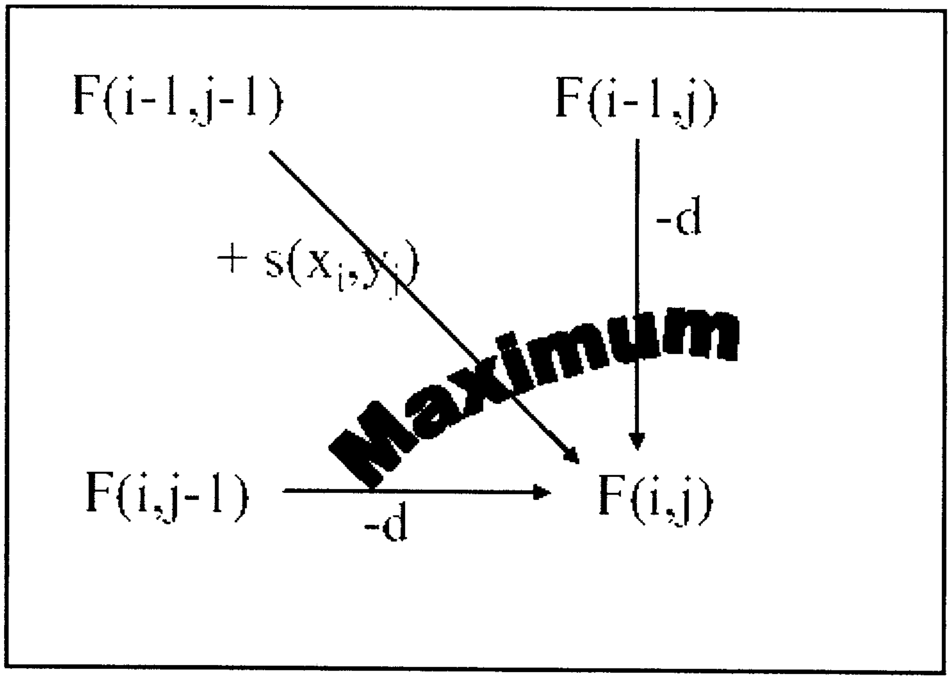
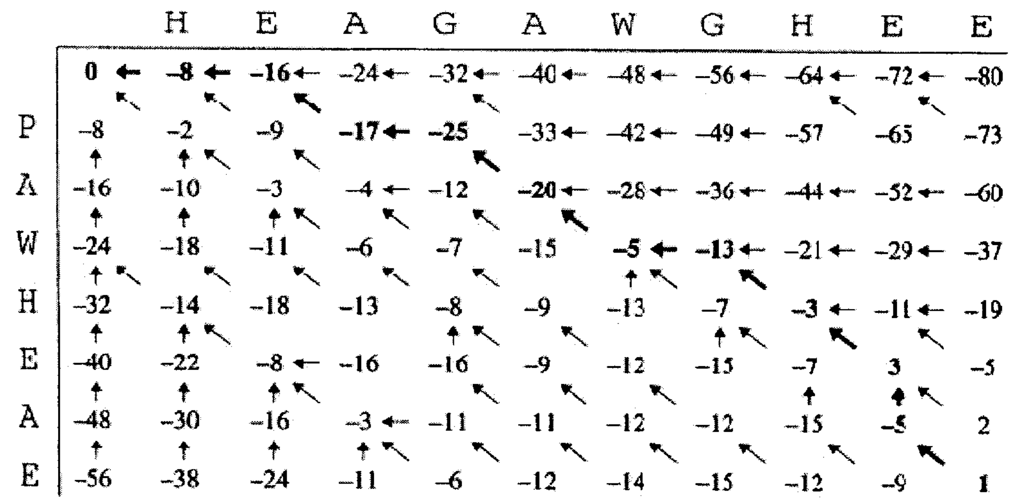


Figure 1: Finding the score of each cell in the matrix

As we fill in the F(i,j) values, we also keep a pointer in each cell back to the cell from which its F(i,j) was derived, as shown in the example of the full dynamic programming matrix in Figure 2.



```
HEAGAWGHE-E
--P-AW-HEAE
```

Figure 2: An example of Needleman – Wunsch algorithm

Note for the figure 2:

- Above, the global dynamic programming matrix for our example sequences, with arrows indicating traceback pointers; values on the optimal alignment path are shown in bold.
- Below, a corresponding optimal alignment, which has total score 1.

To complete our specification of the algorithm of the algorithm, we must deal with some boundary conditions. Along the top row, where j = 0, the values F(i-1, j) and F(i-1, j-1) are not defined so the values F(i,0) must be handled specially. The values F(i,0) represent alignments of a prefix of x to all gaps in y, so we can define F(i,0) = -id. Likewise down the left column F(0,j) = -jd.

The value in the final cell of the matrix, F(n, m), is by definition the best score for an alignment of $x_{1\ldots n}$, $y_{1\ldots m}$, which is what we want: the score of the best global alignment of x to y. To find the alignment itself, we must find the path of choices from the above (2.1) that led to this final value. The procedure for doing this is known as a *traceback*. It works by building the alignment in reverse, starting from the final cell, and following the pointers that we stored when building the matrix. At each step the traceback process we move back from the current cell F (i, j) to the one of the cells (i-1, j-1), (i-1, j) or (i, j-1) from which the value F (i, j) was derived. At the same time, we add a pair of symbols onto the front of the current alignment: x_i and y_j if the step was to (i-1, j-1), x_i and the gap character '-' if the step was to (i-1, j), or '-' and y_j if the step was to (i, j-1). At the end we will reach the start of the matrix, I = j = 0. An example of this procedure is shown in the Figure 2.

Note that in fact the traceback procedure described here finds just one alignment with the optimal score; if at any point two of the derivations are equal, an arbitrary choice is made between equal options. The traceback algorithm is easily modified to recover more than one equal-scoring optimal alignment. The set of all possible optimal alignments can be described fairly concisely using a sequence graph structure [1].

It is useful to know how an algorithm's performance in CPU time and required memory storage will scale with the size of the problem. From the algorithm above, we see that we are storing (n+1) x (m+1) numbers, and each number costs us a constant number of calculations to compute three sums and a max. \We say that the algorithm takes O(nm) time and O(nm) memory, where n and m are the length of the sequences. Since n and m are usually comparable, the algorithm is usually said to be $O(n^2)$. The larger the exponent of n, the less practical the method becomes for long sequences. With biological sequences and standard computers, $O(n^2)$. Algorithms are feasible but a little slow, while $O(n^3)$ algorithm are only feasible for very short sequences.

2.5.3 Semi-global Alignment Algorithm implemented in the report

The objective of global alignment is to maximize a similarity score to give a maximum match between two-compared sequences, where maximum match means the largest number of residues of one sequence that can be matched with another allowing for all possible deletions. In above sections, we have presented a global alignment algorithm with constant penalty (line penalty). In this report, semi-global alignment algorithm with affine gap penalty will be implemented.

Three Main Steps

- Assign similarity scores: A numerical value (score) is assigned to every cell in the array depending on the substitution matrix selected by the user.
- Score pathways through matrix: For each cell want to know the maximum possible score for an alignment ending at that point; Cumulative score by adding in a path through the matrix; searches subrow and subcolumn for the highest score; Gap penalty dependent of the length of the gap; the best match is the pathway with the highest score.

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 ,;
```

,

3 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.

The term generic programming has at least four different meanings:

- Programming with generic parameters.
- Programming by abstracting from concrete types
- Programming by abstracting from concrete types
- Programming with parameterized components programming method based in finding the most abstract representation of efficient algorithms.

The first meaning is the most common: generic parameters are type or value parameters of types. With generic parameters, code duplication can be avoided. [6].

The design of STL is a demonstration of generic programming, a novel programming paradigm that separates data structures from algorithms.

3.1 Generic programming and STL

The data structures and algorithms in the Standard C++ Library, and the STL [2, 14, 16, 20], are a demonstration of a programming paradigm called generic programming. The key abstraction of the generic components in the Standard C++ Library are algorithm, iterators, adapter and functors. Data abstractions are data types and sets of operations on them. C++ provides templates as the necessary constructs for data abstractions. Templates provide a uniform interface and implementation abstractions for different data types. For instance, a template stack class can be instantiated to a stack of integers, doubles, or any user-defined type. Thus, for *N* data types only one template container class is provided which can be instantiated *N* ways.

3.1.1 Algorithms

Generic algorithmic abstractions are families of data abstractions with a common set of algorithms. In order to make algorithms generic they are designed to work on iterators (see below) that are exported by containers. For instance, a sort algorithm could work on a linked list or a vector data abstraction if the list and vector collection classes provide iterator objects that mark the beginning and end of the container. Algorithms are implemented as template functions in STL, typically parameterized over iterators or structural abstractions.

3.1.2 Iterators

Iterators are implementations of structural abstractions and are data type templates exported by container classes. Iterators are generalizations of array pointers for generic containers and provide operators to traverse the range of data they point to and also operators to reference the element they point to. Typically, the pointer arithmetic operators like ++ (auto-increment), - (auto-decrement), +*n* (jump *n* positions forward), and -*n* (jump *n* positions backward), are overloaded to provide traversal implementations. They also overload the comparison operators (==, <, >,) and the assignment operator (=) to compare iterator positions and allow iterator assignments, respectively. The C++ * operator is overloaded to reference the element at the position pointed to by the iterator. Algorithms work over iterators rather than directly over containers. Therefore the same algorithm can work for different container types as long as they export appropriate types of iterators. An additional advantage of having algorithms work on iterators instead of on containers is that algorithms can be used to work on a partial range of elements in a container.

An iterator can be of one of the following kinds - input, output, forward, bidirectional, or random-access. Input iterators are data sources (e.g. the *cin* standard input object in C++), and output operators are data sinks (e.g. *cout* in C++). Forward iterators satisfy properties of both input and output iterators. Forward iterators can be traversed one position at a time only in the forward direction, hence they support only the ++ operator. Bidirectional iterators satisfy properties of forward iterators, can be traversed in both forward and reverse directions one step at a time, and support the - operator. Random access iterators are bidirectional iterators, which can make non-unit jumps in the forward or reverse direction. They support the +*n* and -*n* operators too.

Most container classes export member functions called *begin()* and *end()* which return iterators that point to the first element and past the last element, respectively, of the container object. Starting with these functions, and using the iterator traversal operators, users can construct iterators pointing to a subrange of the elements in a container.

3.1.3 Adaptors

Generic representational abstractions are mappings from one structural abstraction to another. Called adaptors in STL, these abstractions are casting wrappers that change the appearance of a container (building a stack from a list), or an iterator (converting a bidirectional iterator to a reverse iterator). STL also has adaptors to convert C++ I/O streams and arrays to STL-style containers.

3.1.4 Functors

STL also defines function objects (or functors) which are basically template function pointers wrapped in template classes. These classes provide a '()' operator which is used for invoking the function. STL also provides adaptors to convert normal C++ function pointers to function objects.

3.2 Genericity and polymorphism

The ability of a code to work with different type is referred to as *polymophism*, which can be achived through two different ways: subtyping and parametric. Both ways can archive: generic parameters, abstracting from concrete types, parameterized components. Here we will explain how C++ supports these two ways of polymophism and the advantage of parametric polymophism.

3.2.1 Parametric polymorphism

Generic paramerters: almost all the algorithm in STL using generic parameters so that it can be used with different types. For example, sorting routine where the element type is declared as a generic parameters. This way, same sorting code for different types is avoided.

Abstracting from concrete types: Data abstractions are data types and sets of operations on them. C++ provides templates as the necessary constructs for data abstractions. Templates provide a uniform interface and implementation abstractions for different data types. For instance, a template stack class can be instantiated to a stack of integers, doubles, or any user-defined type. Thus, for *N* data types only one template container class is provided which can be instantiated *N* ways. the container data structures and algorithms in STL are desingned and implemented as generic by finding the most abstract representation of efficient algorithms, so that the abstraction process should not compromise efficiency – the abstract algorithm can be instantiated back for a concrete case, and the result is as efficient as the original concrete algorithm. The focus on efficiency has led to using type parameterization as the main implementation technique.

Parameterized components: In STL, *template method* provides a way to vary computation steps while keeping the algorithm structure constant, and the *strategy* provides a way to vary the algorithm used by a component, such an object is refered as an function object. To implement strategy as static parameteration, strategy can be encapsulated as an object and be passed to the sorting routine as an extra parameter.

3.2.2 Subtype polymorphism

Generic parameters: which use an abstact type – abstract class in C++ -- in the place of the element type with dynamic binding. The abctract type should be defined to include the comparison operation , and the rountine works for all subtypes of the abstract type. Programming paradigm prescribes four kinds of abstractions: data abstractions, algorithmic abstractions, structural abstractions, and representational abstractions.

Abstracing from concrete types: list the operation called on a given variable in the code we want to abstract and to use an abstract type including these operations as the type of the variable.

Parameterized components: the function object is made to pass code that varies at runtime, i.e. virtual function in C++.

3.2.3 C++ support for both polymophism

C++ supports subtyping polymorphism through virtual functions. The target component implementing the method may vary at runtime. C++ supports static polymophism through template, typedef, inline, etc.

The main advantages of using static polymophism are: better efficency and smaller executables. Parameter components are instantiated at compile time, and only the necessary functional code to be executated avoiding dynamic binding overhead.

3.3 Using Generic Programming for Application Development

Usually, genericity can be achieved by means of a design idea, that is, separation of data structures from algorithms, and by the use of programming techniques supported by the programming language, i.e. C++ class and templates [19].

Although generic programming uses classical object-oriented C++ language features such as class (template) declarations, it is not object-oriented. In object-oriented programs abstractions are expressed by means of base classes.

Object-oriented programming and generic programming are fundamentally different. Object-oriented design methods could not model a generic program: because classes in generic programming are mostly unrelated. There is no inheritance among them; in generic programming, many relationships are expressed in terms of implicit requirements to template arguments, which can be modeled by object-oriented design method.
Let us consider following scenarios in application development:

Scenario 1: Design an application as a generic program.
To apply the generic programming paradigm to the entire application, i.e. to design an application as a generic program that consists of various generic components that can be plugged together.

The C++ standard already defines a set of generic components - the data structures and algorithms from the Standard C++ Library. It would certainly be wise to make your abstractions compatible to the standard framework. Now, think of the main elements of generic programming in the Standard C++ Library - containers, algorithms, and iterators. Adopting the generic programming paradigm from the Standard C++ Library for an entire application basically means that you have to categorize all significant elements of your software to be either an algorithm, or an iterator, or a container.

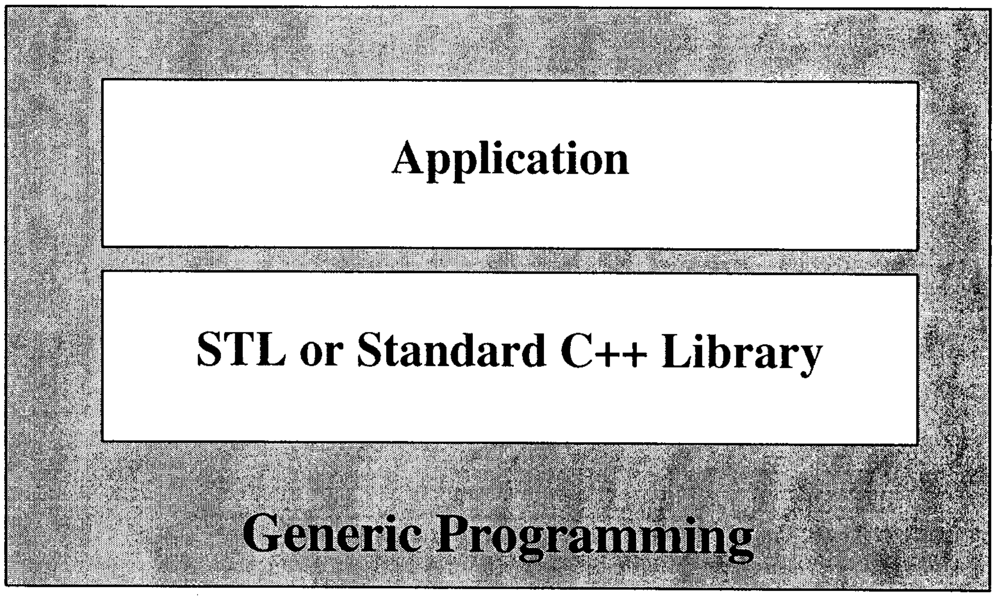


Figure 3: Design an application as a generic program

The notion of these main elements can be broadened to some extent. Take the abstraction of a container for instance. A container in essence is an object that can provide iterators to let algorithms operate on itself. Hence, a container need not necessarily be a typical 'container' data structure such as a list or a queue.

Also, the generic programming paradigm does not impose any limitations regarding the type or number of abstractions. You are free to invent novel generic abstractions. Allocators are an example of this; they were introduced to the STL in the process of making the STL part of the Standard C++ Library. Allocators added another dimension of genericity to containers. You can also contrive a novel set of requirements to algorithms; you might want to add algorithms that accept containers as arguments rather than iterator ranges.

In principle, it is conceivable that to use the ideas of generic programming in an application design, broaden the standard abstractions and add new generic elements.

Scenario 2: Design your application as an object-oriented program that uses the generic components.

In this case you will apply classical object-oriented design techniques, and then build this object-oriented application on top of the generic components from the Standard C++ Library or the STL. In principle there are two possibilities for doing this:

- You can either use the generic components directly, or
- You introduce a middle layer of object-oriented base components that are built on top of the generic components from the Standard C++ Library or the STL.

If you use the generic components directly, i.e. without a middle layer, these generic components will show up in your object-oriented design model. One reasonable way of integrating them is to model the standard containers as classes, and all standard algorithms that can be applied to a container as methods of the respective container class. Basically you would fake an object-oriented container model.

In your object-oriented design the application domain objects will have use- and containment-relationships to the faked object-oriented container classes. However, for purpose of implementation, your application domain classes you will ultimately have to use the generic components as they are: as containers with separate algorithms.

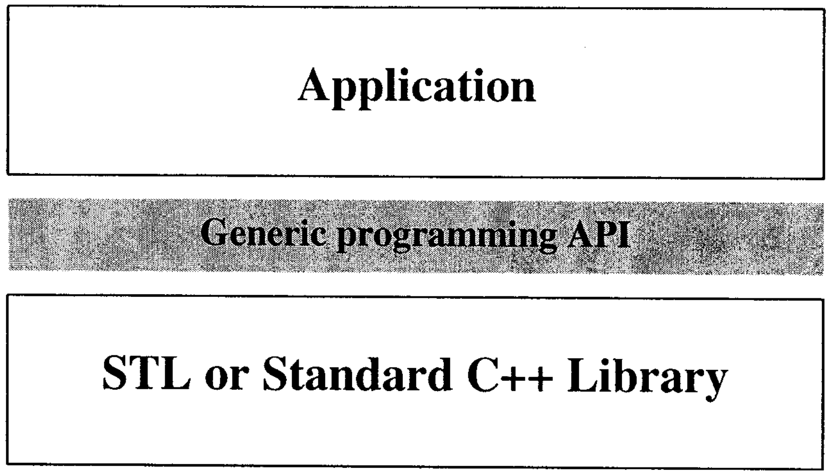


Figure 4: Design an application as an Object-Oriented program that use the generic components (2a)

An alternative possibility is to really implement the object-oriented containers from our design model above, i.e. you will implement a middle layer on top of the generic components.

In your object-oriented design the application domain objects will have use- and containment-relationships to the object-oriented container classes from the middle layer. Different from the approach suggested above, you will also use these object-oriented containers in your implementation. You will not have to switch back to generic programming then.

The generic components from the Standard C++ Library will be hidden behind an object-oriented wrapper and will serve solely as a portability layer. They will be invisible both in design and implementation.

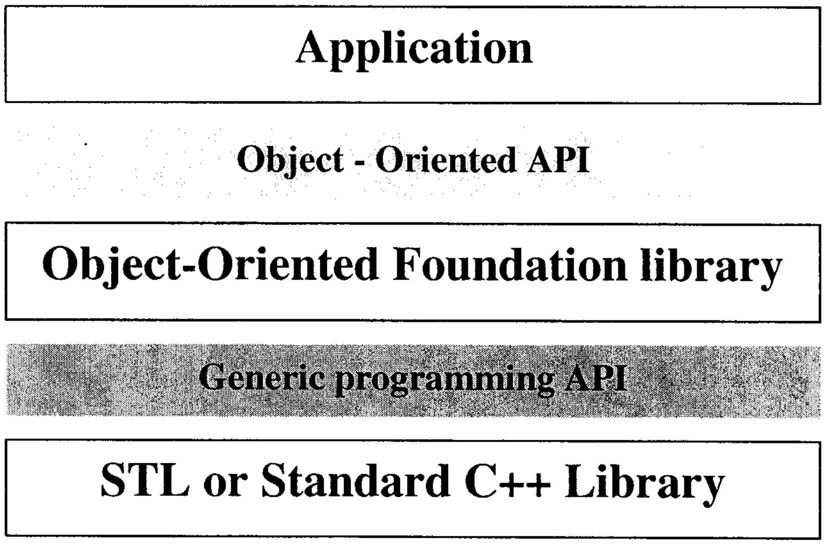

Application
Object - Oriented API
Object-Oriented Foundation library
Generic programming API
STL or Standard C++ Library

Figure 5: Design an application as an Object-Oriented program that use the generic components (2b)

There is a mismatch between object-oriented design / object-oriented programming on the one hand and generic programming / use of generic components on the other hand. You cannot appropriately express generic programming in an object-oriented design. Either you refrain from doing object-oriented design (scenario 1) and stick to the generic programming paradigm. Or you do object-oriented design, fake an object-oriented container model for that purpose, and accept that your implementation does not exactly match the design (scenario 2a). Or you hide the generic components under an object-oriented layer that you can include both in your design and implementation (scenario 2b). All three alternatives are viable approaches.

4 Generic C++ Implementation

This chapter presents a generic application for pairwise alignment algorithms implemented with C++ and how to obtain an implementation for a concrete problem, namely semi-global alignment algorithm based on the framework of generic application and components reuse.

This chapter is organized at follows:

- Section 1: the class design in UML
- Section 2: design and implementation for both framework and function object – Globalalgorithm.
- Section 3: show how to instantiate this application for semi-global alignment algorithm.

4.1 System Design in UML

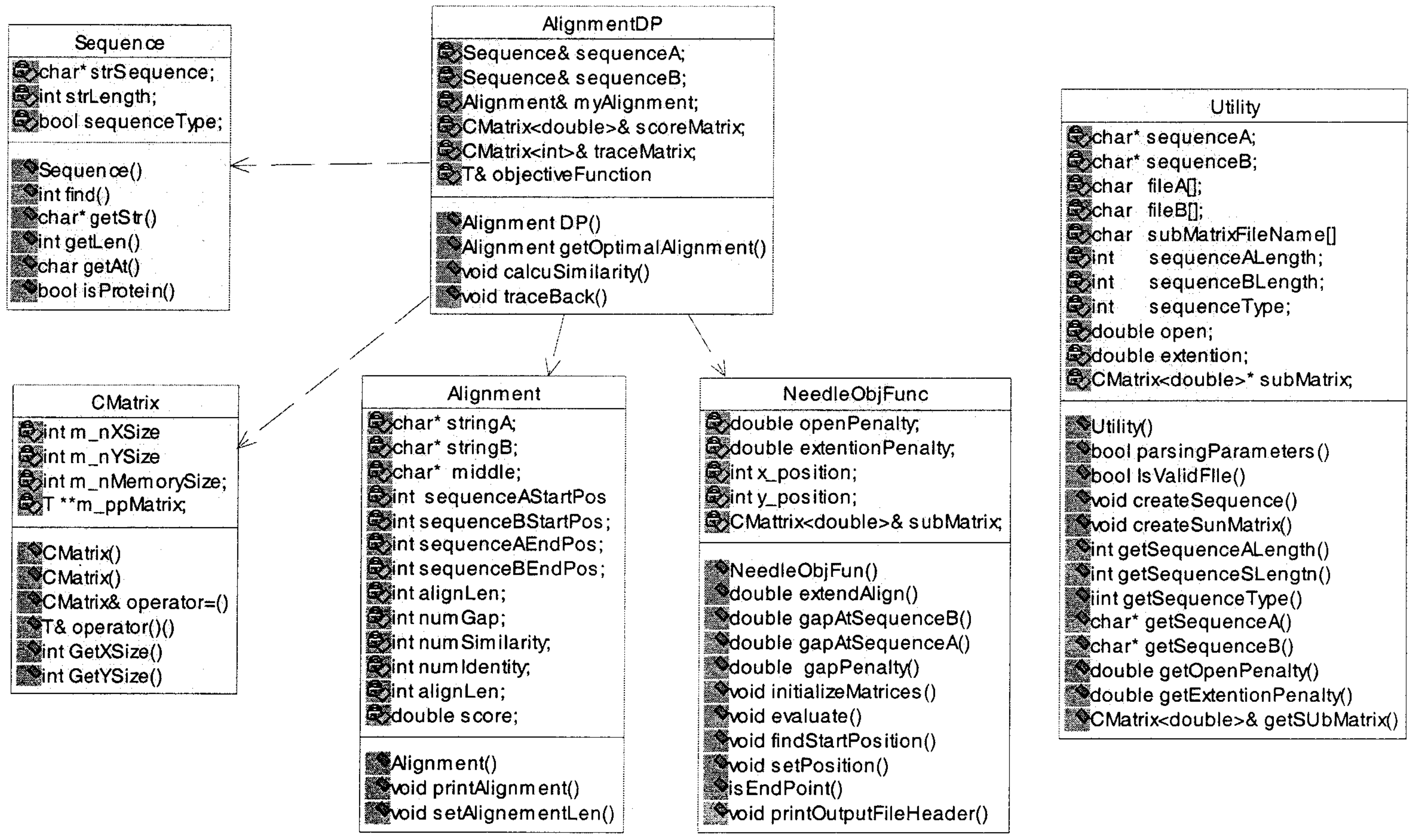


Figure 6: Class diagram in UML

4.2 Design and C++ implementation

This section will show the main points of the design and implementation in C++ of the framework in this application and instantiate a function object for Globalalgorithm.

Performance is the main criteria in comparing sequences. Static binding leads to run-time efficiency since the compiler can optimize code before executing it. The design was driven by the goal of programming with generic parameters and parameterized components so that the generic process should not compromise efficiency. The focus in efficiency has led this project using type parameterization as the main implementation technique.

The idea is that the compile will automatically generate an appropriate concrete function for each parameterized type – alignment algorithm when an application is run.

This project is designed as an object-oriented program that use the generic components. We will apply classical object-oriented design techniques, and then build this object – oriented application on the top of the generic components, which is used directly, as we described in previous chapter, these generic components will show up in object-oriented model, which is modeled as a class.

Based on the revision of several pairwise alignment algorithms, we see that

- There are entities, such as score matrix, trace matrix, Sequence, Alignment; they are not dependent on the problem at hand, in that, they can be implemented in a generic way.
- There is another entity, alignment algorithm, which is completely problem dependent.

Hence, we can divide the components in this application into two distinguished parts:

- A "public" part consists of several classes have been implemented in this project.
- A "private" part consists of generic component class, which needs to be filled in by the users who will instantiate this framework for a given algorithm. In this project, global algorithm has been defined and implemented.

Following are the "public" components in ADP application so as to identify common entities of this application:

Template class AlignmentDP is a template class, which can accept parameterized objective function – pairwise alignment algorithm as well as input sequence, score Matrix, trace Matrix as well as output Alignment object.

This application is a generic application, since the objective function is made a parameter and thus this application can be adapted to work with any pairwise alignment algorithm.

The key abstraction of the generic component in ADP is alignment objective function, which implement alignment algorithm of interest.

Template class Cmatrix is a kind of container. Like many class libraries, the ADP includes *container* class: classes whose purpose is to contain other objects. The ADP includes the class Matrix, which is a template class, and can be instantiated to contain any type of object, In this application, there are two types matrix are instantiated, int and double type, i.e. traceMatrix<int>, which's cell keeps the direction to the start point of alignment; Cmatrix<double>Score Matrix, which's cell keeps maximal score of alignment end at that cell;

Class Sequences is a data structure that keeps sequence raw data, sequence type information as well as find function, which accepts a characters and return the index in substitution Matrix.

Class Alignment is a data structure for output of application, 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 score value of characters in two sequence are greater than 0, otherwise a single dot.

Horizontal bar may be inserted in either sequence to represent gaps.

Class Utility is a utility class, which extracts user command lines parameters and generates data necessary to instantiate above objects.

Objective Function is the only "private" component in this application. **manObjFunc** is an objective function class for Global algorithm to exemplify this application, which will implement global alignment algorithm. Implementation detail, refer chapter 2, section 2.5.

The implementation of this application using parameterized component mechanism:

```
Template <class T>
Class AlignmentDP{
public:
        ATbT: eTtDP(SequeTce& a,
                        SequeTce& T,
                        CMatrix<douTE>& score,
                        CMatrix<iTt>& trace,
                        T& pOTjectiTeFuTctioT,
                        ATbT: eTt& : yATbT: eTt);
.....
}
```

4.3 An Instantiation: semi-global alignment algorithm

This section will show how to implement a function object for a pairwise alignment algorithm and instantiation procedure. In this report, global alignment with affined gap penalty, global algorithm is instantiated.

To be a parameterized objective function of this application, the following function must be implemented for an algorithm:

initializeMatrices ():

initialize the score and trace matrices which are passed by references.

gapPenalty():

function to calculate gap penalty and return the value

extendAlign():

extend alignment, return score value of a certain cell which is coming from upper diagonal cell plus match value.

gapAtSequenceB():

maximal alignment end as sc(a_i, -), return score value comes from upper cell's score value minus gap penalty

gapAtSequenceA():

maximal alignment end as sc(-, b_j), return score value comes from left cell's score value minus gap penalty

evaluation():

evaluate all the possibilities, with global algorithm, is function will return the maximal value of above three cases.

PrintOutputFileName():

display the algorithm name and applicable properties of an algorithm to output file. For global algorithm, it will display Globalas application name as well as display gap opening penalty value and gap extension penalty value.

IsEndPoint() :

implement the criteria for terminating an alignment. With global alignment algorithm, alignment will be end at the beginning of two sequences.

Note that the type parameters allow us to ensure proper typing at compile time. If an algorithm's objective function which is passed not having above functions, a compile time error would be generated.

Follows are the steps to instantiate the application for Globalalgorithm:

- A new class definition should be created for a new alignment algorithm, and all the functions mentioned above should be defined as well. For example, NeedleObjFunc.h in this project.
- A new class implementation file should be created for implementing all the functions; it is file NeedleObjFunc.cpp in this project.
- Modified AlignmentDP.cpp: Include objective function header file for certain alignment algorithm; instantiate an instance of function object just defined. For example: **NeedleObjFunc** NeedleObjectFunction (parameters); instantiate an instance of Alignment application with parameterized object created in step3. For example:

AlignmentDP<**NeedleObjFunc**> myInstance(parameters, NeedleObjectFunction, etc);

5 Experimental Results

5.1 Outline of the Experiments

There are two sets of experiments in report. The first set is a pair Protein sequences and the other one pair is nucleic acid sequences.

- Two protein sequences with open-gap-penalty =10, extension-gap-penalty = 0.5 and substitution matrix EBLOSUM62 for protein sequences.
- Two Nucleic acid sequences with open-gap-penalty = 10, extension-gap-penalty = 0.5 and substitution matrix EDNAFULL for nucleic acid sequences.

5.2 Experiment Data

Protein sequence1: hba_human:

VLSPADKTNV KAAWGKVGAH AGEYGAEALE RMFLSFPTTK TYFPHFDLSH GSAQVKGHGK
KVADALTNAV AHVDDMPNAL SALSDLHAHK LRVDPVNFKL LSHCLLVTLA AHLPAEFTPA
VHASLDKFLA SVSTVLTSKY R

Protein sequence2: Hba_human:

VHLTPEEKSA VTALWGKVNV DEVGGEALGR LLVVYPWTQR FFESFGDLST PDAVMGNPKV
KAHGKKVLGA FSDGLAHLDN LKGTFATLSE LHCDKLHVDP ENFRLLGNVL VCVLAHHFGK
EFTPPVQAAY QKVVAGVANA LAHKYH

Protein sequence3: Rattus norvegicus cxc4 protein sequence

MEIYTSDNYSEEVGSGDYDSNKEPCFRDENENFNRIFLPTIYFIIFLTGIVGNG
LVILVMGYQKKLRSMTDKYRLHLSVADLLFVITLPFWAVDAMADWYFGKFLCKAVHIIYT
VNLYSSVLILAFISLDRYLAIVHATNSQSARKLLAEKAVYVGVWIPALLLTIPDIIFADV
SQGDGRYICDRLYPDSLWMVVFQFQHIMVGLILPGIVILSCYCIIISKLSHSKGH
QKRKALKTTVILILAFFACWLPYYVGISIDSFILLEVIKQGCEFESVVHKWISITEALAF
FHCCLNPILYAFLGAKFKSSAQHALNSMSRGSSLKILSKGKRGGHSSVSTESESSSFHSS

Protein sequence4: Cyprinus Cyprinus carpio cxc4 protein sequence

MEFYDHIFFDNSSDSGSGDFDFDELCDLKVSNDFQKIFLPVVYGIIFVLGIIGNG
LVVLVMGFQKKSKNMTDKYRLHLSIADLLFVLTLPFWAVDAASGWHFGGFLCVTVNMIYT
LNLYSSVLILAFISLDRYLAVVRATNSQNFRRVLAEKVIYLGVWLPASLLTVPDLVFAKV
HDTGMNTICELTYPLQGNTVWKAVFRFQHIFVGFLLPGLIILTCYCIIISKLSKNSKGQA
LKRKALKTTVILILCFFICWLPYCAGILVDTLVMLNVISHTCFLEQGLEKWIFFTEALAY

```
FHCCLNPILYAFLGVKFSKSARNALSISSRSSHKMLTKKRGPISSVSTESESSSVLSS
```

Nucleic acid sequence 1: Didelphis

```
GCAAGTTTCCGCTACCCAGTGAGAATGCCCTTTAAGTCTTATAAATTAAGCAAAAGGAGCTGGTATCAGGC
ACACAAAATGTAGCCGATAACACCTTGCTTTACCACACCCCCACGGGAGACAGCAGTGATTAAAATTAAGC
AATAAACGAAAGTTTGACTAAGTCATAATTTACATTAGGGTTGGTCAATTTCGTGCCAGCCACCGCGGTCA
TACGATTAACCCAAATTAATAAATAACGGCGTAAAGAGTGTTTAAGTTATATACAAAAATAAAGTTAATAA
TTAACTAAACTGTAGCACGTTCTAGTTAATATTAAAATACATAATAAAAATGACTTTAATATCACCGACTA
CACGAAAACTAAGACACAAACTGGGATTAGATACCCCACTATGCTTAGTAATAAACTAAAATAATTTAACA
AACAAAATTATTCGCCAGAGAACTACTAGCAATTGCTTAAAACTCAAAGGACTTGGCGGTGCCCTAAACCC
ACCTAGAGGAGCCTGTTCTATAATCGATAAACCCCGATAAACCAGACCTTATCTTGCCAATACAGCCTATA
TACCGCCATCGTCAGCTAACCTTTAAAAAGAATTACAGTAAGCAAAATCATACAACATAAAAACGTTAGGT
CAAGGTGTAGCATATGATAAGGAAAGTAATGGGCTACATTCTCTACTATAGAGCATAACGAATCATATTAT
GAAACTAAAATGCTTGAAGGAGGATTTAGTAGTAAATTAAGAATAGAGAGCTTAATTG
```

Nucleic acid sequence 2: Dasypus

```
GCAAGTATCAGCACACCAGTGAGAATGCCCTCTAACTCTTATAGATCAAAAGGAGCAAGCATCAAGTACAC
ACAGCCCTTACAGTAGCTCATAACCGAAAGCTTGACTAAGTTATGTTATTATAAGGGTTGGTAAATTTCGT
GCCAGCAACCGCGGTCATACGATTAACCCAAATTAATAGTTATCGGCGTAAAGCGTGTTTAAGACACCTAG
ACAATAGAGTTAAACCCTTACTACGCTGTAAAAAGCCTTAGTAGGACCATAAACCCTTCAACGAAAGTGAC
TCTAATTTATCTGACTACACGATAGCTAGGACCCAAACTGGGATTAGATACCCCACTATGCCTAGCCC
TAAACTAAAACAGTTCACAAACAAAACTGTTCGCCAGAGTACTACTAGCAACAGCTTAAAACTCAAAGGAC
TTGGCGGTGCTTTACATCCTTCTAGAGGAGCCTGTTCTATAATCGATAAACCCCGATATACCTCACCACCC
CTTGCTAATACAGCCTATATACCGCCATCTTCAGCAGACCCTAGTAAGGCACCACAGTGAGCACAATAACA
TACATAAAGACGTTAGGTCAAGGTGTAGCTTATGGGGTGGGAAGAAATGGGCTACATTTTCTAATAAAGAG
CAAATACAAAAAACTTAATGAAACAATTTAAGACTAAGGTGGATTTAGTAGTAAGCTAAAAATAGAGAGTT
TAGCTG
```

5.3 Result Presentation

5.3.1 Output: Hba_human and Hba_human

```
Needleman-Wunsch global alignment.
Gap opening Penalty  : 10.000000
Gap extention Penalty: 0.5
Output file name     :  A
Alignment length:   148
Gaps:                 9
Identity:          63/148 (42.6%)
Similarity:        88/148 (59.5%)
score     :        290.5


sequence 1:       1 -VLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHF-DL      48
                     .|:|.:|:.|.|.||||  :..|.|.|||.|:.:.:|.|:.:|..| ||
sequence 2:       1 VHLTPEEKSAVTALWGKV--NVDEVGGEALGRLLVVYPWTQRFFESFGDL      48

sequence 1:      49 S-----HGSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKLRV      93
                    |     .|:.:||.|||||..|.::.:||:|::....:.||:||..||.|
sequence 2:      49 STPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHV      98

sequence 1:      94 DPVNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKYR      141
                    ||.||:||.:.|:..||.|...||||.|.|:..|.:|.|:..|..||.
```

```
sequence 2:     99 DPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH      146
```

5.3.2 Output: Rattus and Cyprinus

```
Needleman-Wunsch global alignment.
Gap opening Penalty  : 10.000000
Gap extention Penalty: 0.5
Output file name     :  A
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
```

5.3.3 Output: Didelphis and Dasypus

```
Needleman-Wunsch global alignment.
Gap opening Penalty  : 10.000000
Gap extention Penalty: 0.5
Output file name     :  A
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 TTTACCACACCCCCACGGGAGACAGCAGTGATTAAAATTAAGC-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
```

5.4 Result comparison with Needle application in EMBOSS

5.4.1 Output: Hba_human and Hba_human

File: hba_human.needle

```
########################################
# Program:  needle
# Rundate:  Tue Jul 15 10:46:54 2003
# Align_format: srspair
# Report_file: hba_human.needle
########################################
#=======================================
#
# Aligned_sequences: 2
# 1: HBA_HUMAN
# 2: HBB_HUMAN
# Matrix: EBLOSUM62
# Gap_penalty: 10.0
# Extend_penalty: 0.5
#
# Length: 148
# Identity:      63/148 (42.6%)
# Similarity:    88/148 (59.5%)
# Gaps:           9/148 ( 6.1%)
# Score: 290.5
#
#
#=======================================

HBA_HUMAN          1 VLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHF-DL     48
                     .|:|.:|:.|.|.||||  :..|.|.|||.|:.:.:|.|:.:|..| ||
HBB_HUMAN          1 VHLTPEEKSAVTALWGKV--NVDEVGGEALGRLLVVYPWTQRFFESFGDL     48

HBA_HUMAN         49 S-----HGSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKLRV     93
                     |     .|:.:||.|||||..|.::.:||:|::....:.||:||..||.|
HBB_HUMAN         49 STPDAVMGNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKLHV     98

HBA_HUMAN         94 DPVNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKYR    141
                     ||.||:||.:.|:..||.|...||||.|.|:..|.:|.|:..|..||.
HBB_HUMAN         99 DPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH    146
```

Figure 7: A sample output from Needle application in EMBOSS

This result is getting from:
http://www.hgmp.mrc.ac.uk/Software/EMBOSS/Apps/needle.html#output.1

5.4.2 Output: Rattus and Cyprinus

This result is got by running Needle application on line: http://csc-fserve.hh.med.ic.ac.uk/emboss/needle.html

```
Global: , 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
```

5.4.3 Output: Didelphis and Dasypus

This result is got by running Needle application on line:

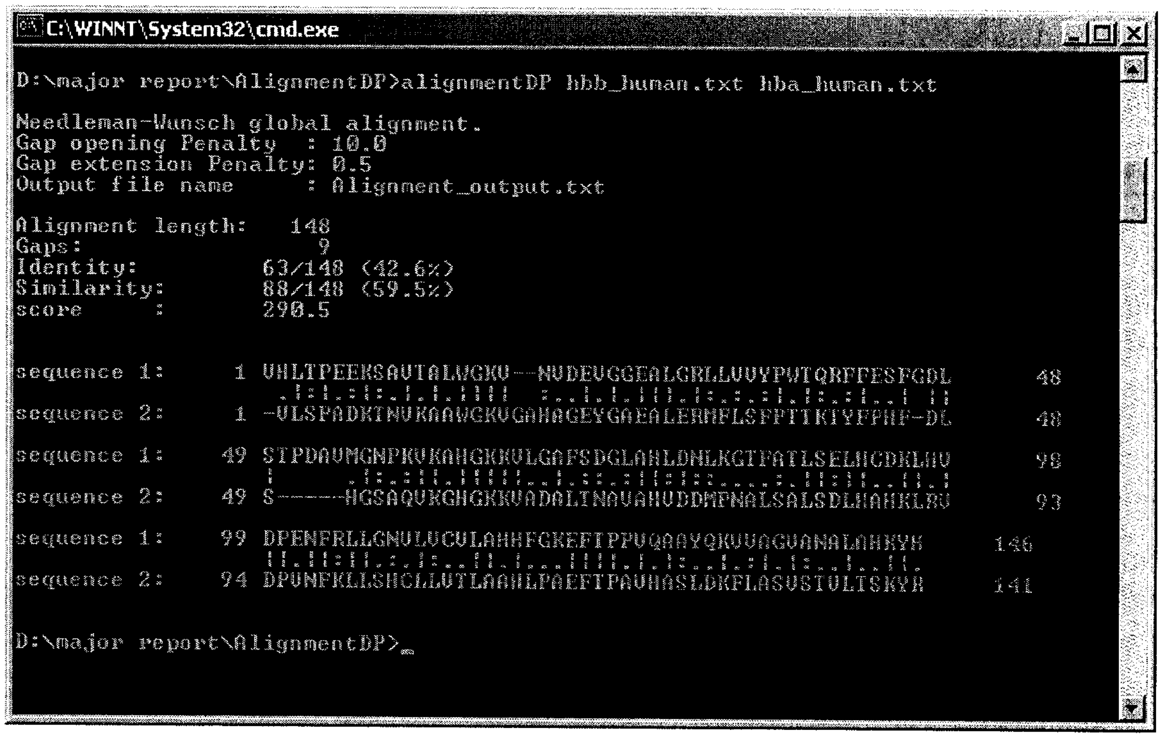
http://csc-fserve.hh.med.ic.ac.uk/emboss/needle.html `Output for`

```
Global: , 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 User Manual

6.1 Function Usage

Here is a sample session with Alignment application instantiated with semi-global algorithm.



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

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

Needleman-Wunsch global alignment.
Gap opening Penalty  : 10.0
Gap extension Penalty: 0.5
Output file name     : Alignment_output.txt

Alignment length:   148
Gaps:                 9
Identity:          63/148 (42.6%)
Similarity:        88/148 (59.5%)
score      :       290.5


sequence 1:      1 UHLTPEEKSAUTALWGKU--NUDEUGGEALGRLLUUYPWTQRFFESFGDL       48
                    .|:|.:|:.|.|.||||  :..|.|.|||.|:.:.:|.|:.:|..| ||
sequence 2:      1 -ULSPADKTNUKAAWGKUGAHAGEYGAEALERMFLSFPTTKTYFPHF-DL       48

sequence 1:     49 STPDAUMGNPKUKAHGKKULGAFSDGLAHLDNLKGTFATLSELHCDKLHU       98
                   |      .|:.:||.|||||..|.::.:||:|::....:.||:||..||.|
sequence 2:     49 S------HGSAQUKGHGKKUADALTNAUAHUDDMPNALSALSDLHAHKLRU       93

sequence 1:     99 DPENFRLLGNULUCULAHHFGKEFTPPUQAAYQKUUAGUANALAHKYH     146
                   ||.||:||.:.|:..||.|...||||.|.|:..|.:|.|:..|..||.
sequence 2:     94 DPUNFKLLSHCLLUTLAAHLPAEFTPAUHASLDKFLASUSTULTSKYR     141

D:\major report\AlignmentDP>_
```

Figure 8: A sample session with Global algorithm

Mandatory qualifiers:

[-asequence]	sequence	**Required** text file for Sequence A, which should be in the same folder of executable file.
[-seqall]	Sequence	**Required** text file for Sequence B, 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 Recommendation

7.1 Conclusion

This project proposes a generic C++ implementation based on a skeleton design for the pairwise alignments. This implementation offers the possibility for the user to instantiate the ADP for any pairwise alignment algorithm with little efforts and basic knowledge of C++ language. The implementation provides both robustness and re-usability properties. This project instantiated the skeleton on one pairwise alignment algorithm, namely semi-global sequence alignment (Globalalignment) so as to evidence the claimed properties of the skeleton.

Different pairwise alignment algorithm can be generated by varying the definition of the data member, the gap penalty, filling Matrix procedure and evaluation criteria etc, so it would be quite interesting to generate new implementation for an algorithm from existing ones with as few changes as possible.

In spite of easy use, the flexibility, the robustness as well as a considerable savings in time, this project has shown that even though being generic, this project implementation achieves comparable results to that of other specific implementations, i.e. Needle in EMBOSS.

In this project, there are two level of re-usability: in the application itself and in the instantiations. The reusability of the application is due to the input and output 's standardization of the of pairwise alignment algorithms. Sequence, Alignment, Matrix and Utility components are directly reused from the ADP skeleton without any effort for the user. The second level refers to the instantiations already dine. There exist many shared methods that can be implemented in the same way for many pairwise alignment algorithms, so a unique implementation can be reused for all of them. Furthermore, by doing small changes in an instantiation, and reusing the rest of it, different implementation of the same algorithm can be obtained.

7.2 Recommendations for Future Works:

Alignment application is easy to understand and reuse. To make Alignment application to be a powerful framework, following detail job need to be carried out for improving it reusability:

1. Input: It is better for Alignment application to accept more kinds of format data as input. So far, this application can only accept text file with raw sequence data. If it is necessary for this application to integrate to Emboss environment, it may need to accept some formats of data, which are generated by other application of Emboss.
2. Interface: For user friendliness, a graphic user interface may need.
3. Output: so far this application only supports one optimal solution; some modification of Alignment class and trace back functionality is needed to support multiple alignment output.

4. In this report, simply template class Matrix is used to store matrix data, simply Sequence is used to store sequence raw data. For the performance and application's extendibility, we can replace those simple classes with container, iterators in STL and The Matrix Template Library (MTL). This application is designed following object-oriented mechanism; it is easy to make standard library's components plugged together.
5. The algorithm implemented in this report is not exactly Needleman – Wunsch algorithm, Function *printOutputFileHeader()* in NeedleObjFunc class may need to modify to prevent confusion.

References

1. S. Altschul and B. W. Erickson, Optimal sequence alignment using affine gap costs. J. Mol. Biol., 48:603 -616, 1986
2. H. Austern, Generic Programming and the STL: Using and Extending the C++ Standard Template Library, Addison-Wesley Publishing, 1998
3. R. Bellman, Dynamic Programming. Princeton Univesity Press, 1957
4. B. Bergeron, Bioinformatics Computing, Prentice Hall PTR, 2002Matthew
5. D. Bertsekas, Dynamic Programming. Prentice Hall, 1987
6. K. Czarnecki, U. Eisenecker and K. Czarnecki, Generative Programming: Methods, Tools, and Applications, Addison-Wesley Publishing, 2000
7. M. Dayhoff, R. M. Schwartz, and B.C. Orcutt, A model of evolutionary change in protein. Atlas of Protein Sequence and Structure, Vol. 5, 345-352, 1978
8. R. Durbin, S. Eddy, A. Krogh and G. Mitchison, Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids, Cambridge, 1999
9. G. B. Fogel and D. W. Corne, Evolutionary Computation In Bioinformatics, Morgan Kaufmann, 2002
10. O. Gotoh, An improved algorithm for matching biological sequences. J. Mol. Biol., 162, 705-708, 1982
11. S. Henikoff and J. G. Henikoff, Amino acid substitution matrices from protein blocks, Proc. Nat. Acad. Sci. U.S.A, 89: 10915-10919, 1992
12. D. E. Krane and M. L. Raymer, Fundamental Concepts of Bioinformatics, Pearson Benjamin Cummings, 2002

13. V.I. Levenshtein, Binary codes capable of correcting deletions insertions and reversals. Soviet Physics-Doklandy, 10(8): 707-710. 1966

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

15. 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

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

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

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

19. Combining OO Design and Generic Programming, C++ Report, March 1997, http://www.langer.camelot.de/Articles/C++Report/OOPvsGP/Introduction.htm

20. Standard Template Library Programmer's Guide, http://www.sgi.com/tech/stl/