AN INTERACTIVE SYSTEM FOR TREE ALIGNMENT AND RECONSTRUCTION

AN INTERACTIVE SYSTEM FOR TREE ALIGNMENT AND RECONSTRUCTION

By

FENG LIU, B.ENG.

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfillment of the Requirements

for the Degree

Master of Science

McMaster University

MASTER OF SCIENCE (1998) MCMASTER UNIVERSITY

(Computation) Hamilton, Ontario

TITLE: An Interactive System for Tree Alignment and Reconstruction

AUTHOR: Feng Liu

B.Eng. (EE, Huazhong University of Science and Technology, China)

SUPERVISOR: Professor Tao Jiang

NUMBER OF PAGES: 116, viii

Abstract

This application software system of Tree Alignment and Reconstruction (TAAR) is an X Window based interactive system developed mainly for sequence analysis. It provides a user-friendly graphical interface and convenient operations for performing Pairwise Alignment, 3-Star Alignment, Phylogeny Reconstruction and Generalized Tree Alignment algorithms. The algorithms for Phylogeny Reconstruction and Generalized Tree Alignment are designed based on heuristic stepwise addition and internal node sequence alignment induction methods. All above algorithms are space-efficiently implemented. For each algorithm, both fast and optimal versions are provided for user's convenience.

TAAR can be used for DNA, RNA and PROTEIN sequences analysis. In general, as long as the sequence characters are in the range of a—z and A—Z, and the score matrix alphabet contains the sequence alphabet, the sequence would be accepted by this application and all calculations could be carried out.

This software system is programmed in C, Motif and X Window Library functions. It is intended to be running in a UNIX and X Window environment.

Acknowledgments

I would like to express my thanks to Professor Tao Jiang for his continually encouragement, support and guidance to me. He is always available for help and has contributed greatly to my interest in the research. My thanks also go to Professor William F. Smyth and Professor Sanzheng Qiao for their constructive suggestions.

I also like to thank Mr. Xian Zhang, Mr. Bin Wu and Miss Yufang Hua very much for their valuable suggestions on software testing and for the sharing of computer resources.

I wish to thank the professors, staff and my fellow graduate students in the department, who made my study at McMaster University an extremely enriching experience.

I would especially thank my wife, Wen Qing Xie, and my son, Daniel, for their constant love, understanding, encouragement and patient cooperation.

Table of Contents

Appendices

Table of Figures

Chapter 1

Introduction

The molecular sequence (DNA, RNA and Protein) analysis plays an important role in computational biology. It is also an important part in the study of evolution of life. The large amount of information and computation involved in the analysis has brought many challenging computational problems to mathematicians and computer scientists. Multiple Sequence Alignment, including Tree Alignment, is among the most important topics in sequence analysis. Since such problems are usually computationally hard (NP-hard in terms of computer science), optimal solutions are not available or not practical. Therefore heuristic and approximate methods are applied and new proposals continue to emerge. For a new algorithm, although it is important that the algorithm is good in the sense of complexity theory, it will be more convincing if a systematic performance analysis is conducted on the computed results, the required time and resources. This demand gives a reason for the design and development of this Tree Alignment and Reconstruction (TAAR) software system.

Before TAAR was developed, another similar application called Sequence Analysis Tool (SAT) was designed and developed by Chen [3] under the supervision of Professor Tao Jiang in 1994. Chen's SAT system has been tested and shown to be very unreliable (sometimes crashed the workstation that SAT works on). In addition to this, it was also implemented on an outdated platform, Openwin, and only few operations (i.e., FAST & OPT Tree Alignments) were implemented in it. Since a much more reliable

system with more functions, such as Phylogeny Reconstruction and Generalized Tree Alignment, is needed in the computational biology research project at McMaster University, I designed this TAAR system based on some of the ideas provided in SAT. Several new functions and improvements are added and implemented in TAAR.

TAAR is an X Window based interactive system developed mainly for providing a platform so that the performance of molecular sequence analysis algorithms developed at McMaster University can be conducted easily. It provides a user-friendly graphical interface and convenient operations for performing Pairwise Alignment, Tree Alignment, Phylogeny Reconstruction and Generalized Tree Alignment algorithms. All above algorithms are space-efficiently implemented, and both fast and optimal versions of these algorithms are provided for the user's convenience.

TAAR can be used for DNA, RNA and PROTEIN sequence analysis. In general, as long as the sequence characters are in the range of a—z and A—Z, and the score matrix alphabet contains the sequence alphabet, the sequence would be accepted by this application and all calculations could be carried out. TAAR allows the user to load, save, view and print out FASTA [20] formatted sequences, custom formatted score matrices and tree graphs. It includes text editors for editing sequence and score matrix, and a graph editor that provides basic editing functions such as *create*, *move*, *cut*, *root* and *undo* for graphs as well as *label* and *unlabel* operations for sequence assignment to tree nodes. The user can use the above graph editing functions to draw any tree directly on the canvas window and assign any loaded sequence onto the tree node; can edit any loaded sequence and score matrix in the edit window; can input and select various scoring

systems and do any of total eight kinds of calculations for Tree Alignment, Phylogeny Reconstruction and Generalized Tree Alignment algorithms. When a calculation is done, the user can view, save and print out the result of multiple sequence alignment and phylogenetic tree structure.

Among the above implemented algorithms, I designed the Phylogeny Reconstruction and Generalized Tree Alignment algorithms for TAAR. The Phylogeny Reconstruction algorithm applies tree alignment and some heuristic stepwise addition methods, that work as follows: from the basic star tree, trying to add a branch on every tree edge and settle down with the most optimal tree cost, and then repeating these steps to grow up the tree until all input sequences are considered and added to the leaf nodes. The Generalized Tree Alignment algorithm works by inferring a phylogeny first and then using the alignment relationship of all node sequences of the phylogenetic tree to induce a multiple alignment of all leaf node sequences. A detailed explanation about these algorithms will be shown in the next chapter.

TAAR is developed with C and Motif (V1.2) languages and Xlib and X Toolkit Intrinsics (R5) functions, and tested in UNIX, X Window Manager and Motif Window Manager operating systems on a SUN SPARCstation (Ultra-2).

This thesis contains eight chapters and four appendices. Chapter 2 gives a general survey of sequence comparison and discusses the Tree Alignment and Phylogeny Reconstruction problems in detail. Dynamic programming is a major approach for solving alignment problems. Chapter 3 reviews the user interface style and some development issues. A brief introduction to the X Window System is also included.

Chapter 4 discusses the development and implementation techniques used in this project. Main data structures and functions for implementing various sequence alignment algorithms are presented. Chapter 5 is a user's guide for the main features of TAAR. Chapter 6 demonstrates some test results of main TAAR calculations. Chapter 7 gives some suggestions for further improvement. Appendix A is a sample sequence file for testing purpose. Appendix B is a sample score matrix file. Appendix C is the test result of multiple sequence alignment of the above sample data. Appendix D is the test result of some real molecular sequences.

Chapter 2

Molecular Sequence Analysis

One of the basic problems of biology is to understand inheritance. Modern Molecular Biology indicates that DeoxyriboNucleic Acid (DNA) is the primary genetic material and it contains information for inheritance. DNA is a molecule composed of four nucleotides: Adenine (A), Cytosine (C), Guanine (G), and Thymine (T), which, conceptually, are linked linearly to form long chains with double helical structure. These chains are called DNA sequences. There is a large class of molecules in cells. These large molecules, known as macromolecules, are most interesting to us. They are DNA, RNA, and Proteins. Proteins are sequences made from 20 amino acids. The information necessary to build a protein or RNA found in an organism is encoded in its DNA molecules. The study of these sequences results in important insights into human biochemistry, physiology and disease processes.

An important method for studying these biology sequences is sequence comparison, which is necessary for the detection of common structure and function as well as for the study of evolutionary relationship. The basic idea of most sequence comparison algorithms is to obtain a measure of similarity (or distance) among a collection of sequences. Alignments are usually constructed in terms of maximizing the measure of similarity (or minimizing distance) between sequences. Since there exist

various clever alignment techniques and algorithms, comparative sequence analysis is an active and fruitful area in the application of computation to biological problems.

To obtain an optimal (or quasi-optimal) alignment, dynamic programming is usually applied. Dynamic programming algorithms are theoretically important and beautiful. However, these methods may require exhaustive computation resources (especially when doing dynamic computation for long sequences or the alignment of many sequences), which may become impractical in many cases because of the limit of computing time and memory resource. The question of how to improve various algorithms to save computer storage resource and to increase computation speed is a very important research topic in biocomputing.

In this Chapter, we will consider some important problems in sequence alignment, in particular, pairwise alignment, 3-star alignment, tree alignment with a given phylogeny, phylogeny reconstruction and generalized tree alignment, and present our algorithmic solutions for each of the problems. These functions are what the TAAR system is able to handle at the time being.

2.1 Pairwise Alignment

Pairwise alignment is the basis of all other sequence comparison methods. In this section, we formally define the terminology and notations needed in alignment.

The *space* is denoted by the symbol " - ".

An *alphabet* Σ is a finite set of symbols including the space.

An *element* is a member of the alphabet Σ.

A *letter* is an element other than the *space*.

A *sequence (*or *string*) is a finite string of letters.

A *padded sequence* (or *padded string*) is a finite string of elements.

Given an alphabet Σ, a scoring function $w : \Sigma \times \Sigma \rightarrow$ **Real** can be defined to measure distance or similarity between any two elements. In this project, we use distance-measuring scoring functions. Therefore the following discussion is based on the distance criteria.

So given two sequences A and B, a *pairwise alignment* of A and B is expressed as A' and B':

$$\begin{cases} A = a_1\, a_2\, a_3 \cdots a_m \\ B = b_1\, b_2\, b_3 \cdots b_n \end{cases} \quad \Rightarrow \quad \begin{cases} A' = a'_1\, a'_2\, a'_3 \cdots a'_l \\ B' = b'_1\, b'_2\, b'_3 \cdots b'_l \end{cases}$$

where the two rows are padded sequences obtained with the insertion of spaces into A and B respectively, such that no column contains two spaces.

To determine the quality of an alignment, one needs some scoring and optimization criterion defined for alignments. For the above alignment, it is natural to consider the sum

$$\sum_{i=1}^{l} C(a'_i,\ b'_i)$$

For example, let $\Sigma = \{a,\ b,\ c,\ d\}$ and let the pairwise scores be defined in the following matrix:

s	*a*	*b*	*c*	*d*	-
a	1	-1	-2	0	-1
b		3	-2	-1	0
c			0	-4	-2
d				3	-1
-					0

Let the two sequences be A = *cacdbd* and B = *cabbdb*. Then the alignment of A and B will be

c a c - d b d
c a b b d b -

and it has a maximum total score sum: $0 + 1 - 2 + 0 + 3 + 3 - 1 = 4$.

However, to get a biologically reasonable alignment, additional costs are usually charged for *gaps*, which are maximal strings of adjacent spaces in a sequence aligned with letters in the other. Therefore we can say the cost (or score) of a pairwise alignment is the sum of the cost of all aligned pairs of elements and the cost of all gaps.

An optimal alignment of strings A and B is one that minimizes the cost over all possible alignments. For any two sequences, there may exist many optimal alignments. The sequence alignment problem is to find one or more optimal alignments and the optimal cost. The standard method uses *dynamic programming* on variants of the following recurrence relations.

Let $A = a_1 a_2 a_3 \cdots a_m$ and $B = b_1 b_2 \cdots b_n$. Define a cost matrix CC such that $CC[i, j]$ denotes the minimum cost of aligning $a_1 a_2 \cdots a_i$ and $b_1 b_2 \cdots b_j$. When gap costs are not charged, we have the recurrence relation:

$$CC[0, 0] = 0,\ \mathrm{CC}[0, j] = \sum_{k=1}^{j} C(-, b_k),\ CC[i, 0] = \sum_{k=1}^{i} C(a_k, -),$$

and

$$CC[i, j] = \min \begin{cases} CC[i-1, j] + C(a_i, -) \\ CC[i-1, j-1] + C(a_i, b_j) \\ CC[i, j-1] + C(-, b_j) \end{cases}$$

Where $0 \le i \le m$ and $0 \le j \le n$.

A direct implementation of this relation needs O(mn) space and time. Dynamic programming with this kind of recurrence relations has been studied extensively in the past. Lots of time or space saving strategies have been proposed and actually applied [5, 8, 30].

It is important to realize that an optimal alignment is optimal only for a particular scoring system and gap costs. To make significant biological application of these mathematical models, we have to consider how to choose the scoring function w on Σ, and also the gap cost function. In practice, the primary scoring system used for DNA sequences is the identity matrix. For protein sequences, the most common choice for measuring similarity is the Dayhoff mutation matrices (PAM matrices) [4].

When gap penalties are considered, let g_k be the gap cost for a gap of k spaces. Then we have the recurrence relation:

$$\mathrm{CC}[0, 0] = 0, \quad \mathrm{CC}[0, \mathrm{j}] = g_j, \quad \mathrm{CC}[\mathrm{i}, 0] = g_i,$$

and

$$CC[i, j] = \min \begin{cases} CC[i-1, j-1] + C(a_i, b_j) \\ \min_{1 \le k \le j} \{CC[i, j-k] + g_k\} \\ \min_{1 \le k \le i} \{CC[i-k, j] + g_k\} \end{cases}$$

A direct dynamic programming implementation of the above relation requires $O(mn)$ space and $O(mn(m+n))$ time complexity. Among possible gap penalties, the simplest and most commonly used are *affine gap penalties*, which charges a fixed amount for each additional space. Because they have been taken to subsume space costs, they are usually expressed as

$$g_k = a + bk$$

where a is called the *open gap penalty* and b the *gap extension penalty*. It is known that pairwise alignment with affine gap penalties can be computed in $O(mn)$ space and time [8].

Gap penalties have a large effect on an alignment and it is wise to sample a wide range of penalty values in order to find the most interesting optimal alignment. The following example is taken from the book [7], which shows how gap penalties influence sensible alignments. Two alignments of human pancreatic hormone and chicken pancreatic hormone are shown.

An optimal alignment without gap penalties:

```
Human   ALLLQPLLGAQGAPLEPVYPGDNATP-EQMAQ-YAAD-LRRYINMLTRPRYGKRHKEDTLAF
Chicken G----P--S-Q--P--T-YPGDDA-PVEDLIRFY--DNLQQYLNVVT------RHR-----Y
```

An optimal alignment with gap penalty of 1.0 + 0.1 × (gap length):

```
Human   ALLLQPLLGAQGAPLEPVYPGDNATPEQMAQYAADLRRYINMLTRPRYGKRHKEDTLAF
Chicken ------------GPSQPTYPGDDAPVEDLIRFYDNLQQYLNVVTRHRY-----------
```

2.2 Star Alignment

A *star-alignment* is a special case of tree alignment in which the tree has only one internal node. Here, we are especially interested in star-alignment of three strings, which is a problem stated as: *Given three strings A, B and C, construct a new string D and optimally align D with each of A, B and C. The sum of costs of the three pairwise alignments is defined as the cost of the star-alignment.* This problem will be denoted as *star-alignment* (*A, B, C*). An optimal star-alignment (*A, B, C*) is one attaining the minimum cost among all possible star-alignment (*A, B, C*). The string *D* newly constructed in an *optimal* star-alignment (*A, B, C*) is called an optimal *center-string* of (*A, B, C*).

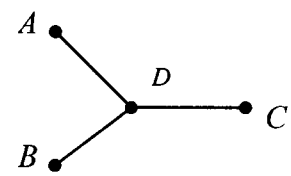


Figure 2.1: A Star Tree

In the following, we present an algorithm using the dynamic programming method to find a center-string of (*A, B, C*) with given strings $A = a_1a_2 \ldots a_M$, $B = b_1b_2 \ldots b_N$ and $C = c_1c_2 \ldots c_P$.

Let A_i denote the i-symbol prefix $a_1a_2 \ldots a_i$, of A, B_j the j-symbol prefix $b_1b_2 \ldots b_j$ of B and C_k the prefix $c_1c_2 \ldots c_k$ of C. Define

$$Cost(i, j, k) = \text{the cost of an optimal star-alignment}(A_i, B_j, C_k).$$

Then we can obtain the following recurrence relation:

$$Cost(i, j, k) = \min_{\delta \neq (0)} [\, Cost(i - \delta_1, j - \delta_2, k - \delta_3) + \min_{x \in \Sigma} \{ w(\delta_1 a_i, x) + w(\delta_2 b_j, x) + w(\delta_3 c_k, x) \} \,]$$

where $\delta_1, \delta_2, \delta_3 \in \{ 0, 1 \}$ and $\delta = (\delta_1, \delta_2, \delta_3)$. Thus we have $1a = a$ and $0a = -$. There are seven different values of δ with $\delta \neq (0)$ (here $\delta \neq (0)$ means $\delta \neq (0, 0, 0)$). The element x where the minimum is attained is the last element (maybe "-") of the constructed center-string of (A_i, B_j, C_k).

In the implementation of this recurrence relation, we first preprocess the part

$$\min_{x \in \Sigma} \{ w(\delta_1 a_i, x) + w(\delta_2 b_j, x) + w(\delta_3 c_k, x) \}.$$

Define a relation *lookup*: $\Sigma \times \Sigma \times \Sigma \rightarrow \Sigma \times$ **Real,** such that, for any $e_1, e_2, e_3 \in \Sigma$, *lookup*(e_1, e_2, e_3) has two fields, one is denoted as *cost* and represents the minimum sum, and the other is denoted as *letter* and stores the letter x such that the minimum sum is attained at x.

Once the *Cost*(i, j, k) matrix is computed, to obtain a center-string of (A, B, C), one can use the straightforward *backtraking technique.* That is, we can set up a 3-dimensional matrix *TraceMat* by computing the recurrence relation of *Cost*(i, j, k) and store in the (i, j, k) cell of *TraceMat*, the element x and a pointer pointing to $(i - \delta_1, j - \delta_2, k - \delta_3)$ such that x and $\delta = (\delta_1, \delta_2, \delta_3)$ lead to the minimum value *Cost*(i, j, k). With the

information stored in the matrix *TraceMat*, we can then simply start from the (M, N, P) cell of *TraceMat*. By following the pointers in *TraceMat*, a linked list is established and a center sequence is obtained in the reverse order. Obviously, an implementation based on this technique needs $O(MNP)$ space. In practice, this space requirement often limits the method's applicability.

If we are only interested in the cost of an optimal star-alignment (A, B, C), the space requirement can be reduced dramatically. More specifically, for each fixed $1 \leq i \leq M$, we use *level* (i) to represent the 2-dimensional array consisting of $\{Cost(i, j, k) \mid 1 \leq j \leq N, 1 \leq k \leq P\}$. The recurrence relation shows that $Cost(i, j, k)$ depends only on seven values in *level* (i - 1) and *level* (i). Therefore, two matrices of size $N * P$ are adequate to compute successive levels. In fact, with a little care, one matrix of size $N * P$ and one vector of size $P + 1$ suffice. Suppose $Cost(i, j', k')$ needs to be computed and values preceding it have already been obtained. Then we can define a matrix CC_{N*P} and a vector CB of size $P + 1$ represented as follows:

$$CC(j, k) = \begin{cases} Cost(i, j, k) & \text{if } j < j' \text{ and } k < k' \\ Cost(i - 1, j, k) & \text{otherwise} \end{cases}$$

$$CB(k) = \begin{cases} Cost(i - 1, j, k - 1) & \text{if } k < k' \\ Cost(i - 1, j - 1, k) & \text{otherwise} \end{cases}$$

$$CB(P + 1) = Cost(i\text{-}1, j'\text{-} 1, k'\text{-} 1)$$

With this loop-invariant condition, we can present an algorithm as shown in Figure 2.2 for calculating the cost of an optimal star-alignment (A, B, C) using $O(N*P)$ space.

```
Algorithm Costonly_star_align(A, B, C, CC)
 Input: A, B, and C (string of size M, N and P).
        {We assume the lookup relation has been computed}
 Output: CC (a matrix of size N * P) and cost where CC(j, k) represents the cost
         of an optimal 3-star-align(A_M, B_i, C_j).

 var array CC[0...N][0...P], CB[0...(P+1)].
 begin
   CC(0, 0) = 0
   for i = 0 to M do
     for j = 0 to N do
       for k = 0 to P do
          begin
            if ( i -1, j - 1, k -1) is valid then
               value(0) = CB(P + 1) + lookup(a_i, b_j, c_k).cost;
            if ( i -1, j - 1, k ) is valid then
               value(1) = CB(k) + lookup(a_i, b_j, -).cost;
            if ( i -1, j, k -1) is valid then
               value(2) = CB(k - 1) + lookup(a_i, -, c_k).cost;
            if ( i -1, j, k) is valid then
               value(3) = CC(j, k) + w(a_i, -);
            if ( i, j - 1, k -1) is valid then
               value(4) = CC(j - 1, k - 1) + lookup(-, b_j, c_k).cost;
            if ( i, j - 1, k) is valid then
               value(5) = CC(j - 1, k) + w(b_j, -);
            if ( i, j, k -1) is valid then
               value(6) = CC(j, k - 1) + w(c_k, -);
            {update CB and CC as follows:}
            if (i > 0) and (j > 0) then
               CB(P + 1) = CB(k);
            if (i > 0) then
               CB(k) = CC(j, k);
               CC(j, k) = min of {value(i) | 0 ≤ i ≤ 6};
          end
   cost = CC(N, P);
   Output cost and CC;
 end
```

Figure 2.2: Algorithm **Costonly_star_align**.

To make the algorithm shorter and easy to read, we include the testing of the boundary situation in the main body. But in our implementation, we deal with the boundary cases separately to eliminate the "if" instructions.

To actually produce a center-string of a star-alignment, we generalize the recursive divide-and-conquer technique of Hirschberg [13] and Myers and Miller [18] so that we obtain an algorithm with $O(N * P + \log M)$ space requirement. The central idea is to find the "midpoints" of an optimal star alignment of three strings by using a "forward" and "backword" application of the quadratic space *Costonly_star_align* algorithm. Then a center-string can be obtained by recursively determining optimal star alignments on both sides of the midpoints.

For a sequence X, let $rev(X)$ denote the reverse of X and let X_i^T denote the suffix $x_{i+1}x_{i+2}\cdots x_M$ of X. Given three sequences A, B and C of sizes M, N and P respectively, applying the algorithm *Costonly_star_align* to $rev(A)$, $rev(B)$ and $rev(C)$, we obtain a matrix RR such that the entry $RR(N - j, P - k)$ represents the cost of an optimal star-alignment (A, B_j^T, C_k^T).

Now we are in a position to explain our algorithm of delivering a center string of a 3-star alignment. Again, suppose three strings A, B and C are of non-zero length M, N and P respectively. Let $i^* = \lfloor M/2 \rfloor$, then $level(i^*)$ bisects the cube associated with the recursive *Cost* function (defined on Page 11). Applying the *Costonly_star_align* algorithm to the strings A_{i^*}, B and C, we get a matrix *matf* satisfying:

$matf(j, k)$ = the cost of an optimal star-alignment(A_{i^*}, B_j, C_k).

Then applying the *Costonly_star_align* algorithm to the strings $rev(A_{i*}^T)$, $rev(B)$ and $rev(C)$, we get a matrix *matb* satisfying:

$matb(j, k)$ = the cost of an optimal star-alignment (A_{i*}^T, B_j^T, C_k^T).

For any star-alignment (A, B, C), there exist $j \in [0, N]$ and $k \in [0, P]$ such that the star-alignment is the concatenation of a star-alignment(A_{i*}, B_j, C_k) and a star-alignment (A_{i*}^T, B_j^T, C_k^T). Thus the cost of an optimal star-alignment (A, B, C) is

$$\min\{matf(j, k) + matb(N-j, P-k) \mid j \in [0, N] \text{ and } k \in [0, P]\}.$$

If the minimum is attained at j^* and k^*, then (i^*, j^*, k^*) is an optimal midpoint for the problem. Now the crucial point for using the divide-and-conquer method is that the concatenation of an optimal star-alignment (A_{i*}, B_{j*}, C_{k*}) and an optimal star-alignment $(A_{i*}^T, B_{j*}^T, C_{k*}^T)$ is an optimal star-alignment(A, B, C). Therefore we can employ the midpoint (i^*, j^*, k^*) to split the star-alignment problem into two sub problems of star-aligning shorter strings. Then sub problems can be solved by calling the above processing recursively.

The recursion's boundary cases, i.e. when the size of one of the three strings is 1 or 0, are handled directly by using the *backtracking* technique since only quadratic space is required now.

The full algorithm for finding a center-string is outlined in Figure 2.3. It uses $O(NP + \log M)$ space: $O(NP)$ which is the dynamical allocated space for *matf* and *matb* or for the boundary cases, and $O(\log M)$ which is the implicit activation stack needed for no more than $\lfloor \log M \rfloor + 1$ levels of recursion. Now consider the time complexity requirement. Obviously, the procedure *Costonly_star_align* for strings of sizes M, N and

Algorithm *find_center_string_star_align*(A, B, C)
Input: A, B and C (strings of size M, N and P).
{We assume the *lookup* relation has been computed}
Output: *final_seq* (a center-string of star alignment) and *cost* (the optimal cost of star alignment)

begin
cost = *findcenter*(A, B, C)
print(*final_seq*) {a center string for the star-alignment}
end

recursive function ***findcenter***(A, B, C)
var array *matf*[0...N][0...P], *matb*[0...N][0...P];
{dynamically allocated in the implementation}
begin
if ($M \leq 1$) or ($N \leq 1$) or ($P \leq 1$) then
{Take $I1$ as the minimum of {M, N, P}, $I2$ and $I3$ as others, allocate at most two matrices of size of $I2$ x $I3$ to store information for backtracking}
apply backtracking technique directly to find a partial center-string, which will be appended to final_seq.
else
begin
$i^* = \lfloor M/2 \rfloor$;
Allocate space for matf and matb;
(α) *Costonly_star_align*(A_{i^*}, B, C, *matf*);
(β) *Costonly_star_align*($rev(A_{i^*}^T)$, $rev(B)$, $rev(C)$, *matb*);
(γ) Find j^* and k^* minimizing(*matf*(j, k) + *matb*($N-j$, P - k));
Free the space of matf and matb;
(δ) *cost*1 = *findcenter*(A_{I^*}, B_{j^*}, C_{k^*});
(ε) *cost*2 = *findcenter*($A_{i^*}^T$, $B_{j^*}^T$, $C_{k^*}^T$);
output *cost*1 + *cost*2.
end
end

Figure 2.3: Algorithm find_center_string_star_align.

P takes $O(MNP)$ time complexity; we assume it is c_1MNP. Then in the algorithm *find_center_string_star_align*, boundary cases take $O(M + NP)$ time complexity; line (α) takes $c_1(M/2)NP$ time, line (β) also takes $c_1(M/2)NP$ and line (γ) takes c_2NP time. So the main body (lines α, β and γ) of the top-level call takes $c_1MNP + c_2NP$ time. The time complexity for the main bodies of the two recursive calls at lines (δ) and (ε) is

$$c_1(M/2)[jk + (N - j)(P - k)] + c_2[jk + (N - j)(P - k)],$$

which is no more than $c_1(M/2)NP + c_2NP$. It follows by induction that the total time taken in the worst case, including recursive calls and boundary cases, is no more than

$$c_1MNP(1 + \tfrac{1}{2} + \tfrac{1}{4} + \cdots) + c_2(\log M)NP + O(M + NP),$$

which equals $2c_1MNP + c_2(\log M)NP + O(M + NP)$. Therefore the time complexity required for algorithm *find_center_string_star_align* is approximately twice that for the cost-only version *costonly_star_align.*

In the above discussion we did not consider gap costs. When gap costs are involved, the situation is much complicated. If the recurrence relation $Cost(i, j, k)$ is to be redefined to include the consideration of gap costs, we will have to not only consider the values of $Cost(i - \delta_1, j - \delta_2, k - \delta_3)$, but also analyze the ending pattern of the partial alignment of $(A_{i-\delta_1}, B_{j-\delta_2}, C_{k-\delta_3})$. Define the *history* of a partial alignment to be the amount of information necessary to determine the cost of any possible extensions. To find an optimal alignment it is necessary in general to know, at each node, the minimum cost of the partial alignment in each historical situation. Alstchul [1] presents a general

analysis of gap costs for tree and star-alignments and infers that the number of relevant histories for star-alignment of n input strings using gap costs is

$$2^n - 1 + \sum_{i=0}^{n-1} \binom{n}{i} 3^i = 4^n - 3^n + 2^n - 1.$$

In our case n=3, so there are 44 histories to be considered for each (i, j, k). Due to time constraint, we are not able to implement gap costs in 3-star alignment here.

2.3 Tree Alignment

A *loaded tree* is a tree whose nodes are associated with sequences. The cost of an edge in the tree is defined as the edit distance (optimal alignment cost) between the two sequences associated with the ends of the edge. The cost of a tree is the sum of the costs of all edges. Here we are interested in the so-called *tree alignment* problem which is, given a set X of sequences and a phylogeny T which is defined as a tree structure such that each leaf is assigned a unique sequence of X, we need to construct a sequence for each internal node such that the total cost of the tree is minimized.

The problem of tree alignment with a given phylogeny is NP-hard even if the phylogeny is a binary tree [29]. Some heuristic algorithms also have been proposed [11, 23]. In the following, we outline an efficient approximation algorithm based on the results of Jiang, Lawler and Wang [14].

First, we need some notations. For a (rooted) tree T, $r(T)$ denotes the root of T, $c(T)$ denotes the cost of T, $Leaf(T)$ denotes the set of the leaves of T. For each node v of T, T_v denotes the subtree of T rooted at v. A leaf that is a descendant of node v is called a *descendant leaf* of v. When all leaves have been assigned sequences, we define $S(v)$ to be the set of sequences assigned to the descendant leaves of v. A loaded tree is called a *lifted* tree if for every subtree T_v, there exists a path from the root $r(T_v)$ to a leaf such that every node on the path is associated with the same leaf sequence. Calculating this *lifted* tree is an important step in solving Tree Alignment problem. We will show this calculation later.

Let $X = \{s_1, \ldots, s_k\}$ be a set of sequences and T a non-rooted phylogeny for X such that the degree of each internal node of T is 3. To construct a loaded tree, we need the following steps.

Step 1: **Initialize the internal nodes.**

Take an arbitrary edge uv of the tree T. Adding a new node r and replacing the edge uv with two new edges ru and rv, we get a rooted tree $\overline{T}$ with root r.

For each $v \in \overline{T}$ and each $s_i \in S(v)$, let $D[v, s_i]$ denote the cost of an optimal lifted tree for T_v with v being assigned the sequence s_i. $D[v, s_i]$ can be computed as follows. For each leaf v, we define $D[v, s_i] = 0$ if s_i is assigned to v. Let v be an internal node, and v_1 and v_2 its children. For each $s_i \in S(v)$, s_i must belong to one of $S(v_1)$ and $S(v_2)$. We have the recurrence relation:

$$D[v, s_i] = \begin{cases} min_{Sj \in S(v2)} \{D[v_1, s_i] + D[v_2, s_j] + dist(s_i, s_j)\} & \textbf{if } s_i \in S(v_1) \\ min_{Sj \in S(v1)} \{D[v_1, s_j] + D[v_2, s_i] + dist(s_i, s_j)\} & \textbf{if } s_i \in S(v_2) \end{cases}$$

The full algorithm is described in Figure 2.4. It outputs a lifted tree with cost at most $2c(T^{min})$ and requires $O(k^3 + k^2n^2)$ time in the worst case, where T^{min} denotes an optimal loaded tree and n denotes the maximum length of the given sequences.

Algorithm Initialization(X, T)
Input: $X = \{s_1, \ldots, s_k\}$(a sequence set), T (a phylogeny for X)
Output: Lifted tree T.

1. **begin**
2. **for** each pair (i, j), $1 \le i < j \le k$, **do**
3. compute $dist(s_i, s_j)$.
4. Construct $\overline{T}$.
5. **for** each level of $\overline{T}$, with the bottom level first, **do**
6. **for** each node v at the level **do**
7. **for** $i = 1$ to k
8. **if** $s_i \in S(v)$ **then** compute $D[v, s_i]$.
9. Select an $s \in X$ such that $D[r(T), s]$ is minimized.
10. Compute the lifted tree T by backtracking.
11. **end.**

Figure 2.4: Algorithm **Initialization**.

When selecting the virtual root for a non-rooted tree, I use the Depth First Search algorithm to locate an optimal position for this virtual root node. By this way, we can avoid any long branch, hence, avoid unbalanced tree. The algorithm for virtual root selection is shown in Figure 2.5.

Algorithm Virtual Root Selection (T)

Input: A non-rooted tree T
Output: A rooted tree T'

1. **begin**
2. For each leaf node, do Depth First Search and memorize its longest path.
3. Compare and select the longest path among those paths found in step 2.
4. Select the node u that resides in the middle of the longest path selected in step 3.
5. Select the edge $\{u, v\}$ that is connected to the node u and is along the longest path.
6. Adding a new node r and replacing the edge $\{u, v\}$ with two new edges $\{r, u\}$ and $\{r, v\}$, Then T is changed to the rooted tree T' and the node r is called the virtual root of T.
7. Output this rooted tree T'.
8. **end.**

Figure 2.5: Algorithm **Virtual Root Selection.**

Step2: **Local Optimization.**

Starting with a lifted tree, we can use an iterative improvement method to update and improve the sequences assigned to the internal nodes to get a better approximation. Recall that the degree of each internal node of the phylogeny under our consideration is three. Based on each internal node, we can construct a 3-component which is a subtree consisting of the internal node and three edges connecting to it. Then on each 3-component local optimization is performed by the star-alignment technique introduced in previous section.

To illustrate these two procedures more clearly, let's consider the phylogeny in Figure 2.6, which contains nine given species on its nine leaves. To construct an evolutionary tree, we assign one of the nine given sequences to each internal node by applying algorithm introduced in Step1. Then we divide the phylogeny into seven 3-components as shown in Figure 2.6. Local optimization is done for every 3-component as follows. For the 3-component in Figure 2.7 (a), from the labels (sequences) s_1, s_2, and s_3 of the three terminals, we can compute the label c_1 (sequence) of the center using dynamic programming (introduced in next section) to minimize the cost of the component. The revised c_1 can then be used to update the center label c_2 of an overlapping 3-component (see Figure 2.7 (b)). The algorithm converges the tree cost since each local optimization reduces the cost of the tree by at least one. Thus, if the process is repeated long enough, every 3-component will become optimal. However, this does not necessarily result in an optimal loaded tree. Nonetheless, by experiment, it seems the algorithm can produce a reasonably good loaded tree after 5 iterations.

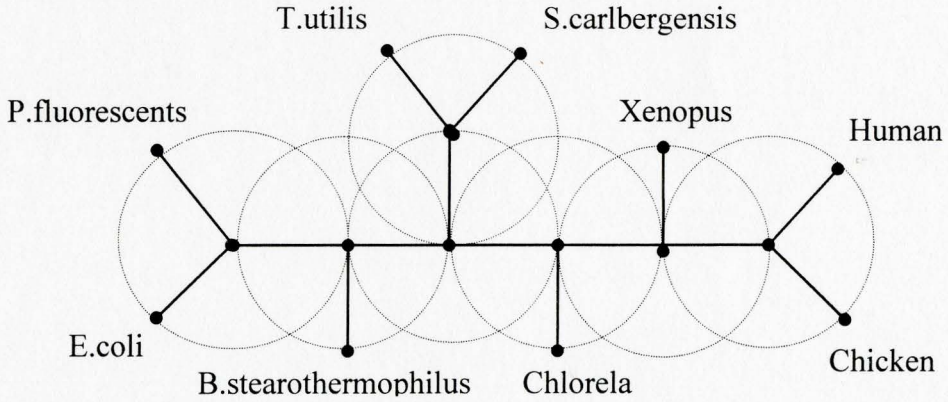
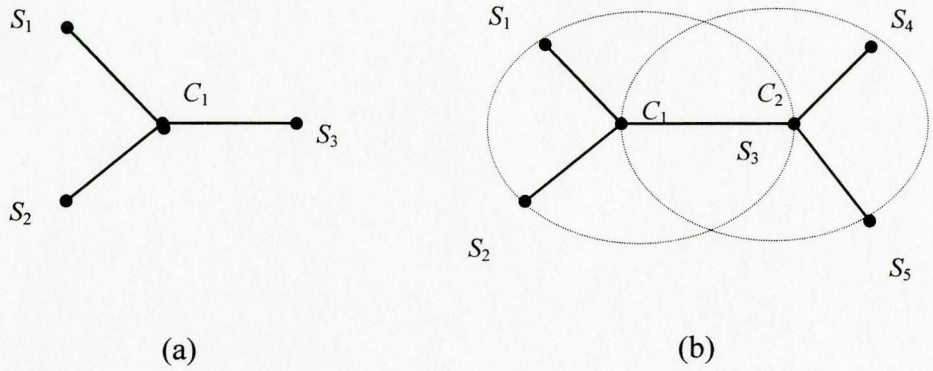


Figure 2.6: A phylogeny with nine species that is divided into seven 3-components.



(a) (b)

Figure 2.7: (a) A 3-component. (b) Two overlapping 3-components.

2.4 Tree Alignment as Multiple Sequence Alignment

Tree Alignment is essentially a multiple sequence alignment problem because of the following connection. After we find an (approximately) optimal loaded tree, we can make a multiple sequence alignment for all leaf sequences based on the pairwise alignment relationship of all adjacent node sequences in the tree. Suppose that we have computed optimal pairwise alignments for all edges, then we can compute a multiple sequence alignment for all leaf sequences by doing sequence length adjustment among leaf and internal node sequences as follows. For example, a part of the tree T is shown in Figure 2.8.

Figure 2.8: A part of the computed phylogenetic tree.

In this tree, the node sequences between any edge are aligned. The alignments are shown above.

We will align A and B via the alignments of $\{A, Q\}$ and $\{B, Q\}$ sequence pairs. Let's give different name $Q1$ and $Q2$ to Q respectively to distinguish these two alignments. Since $Q1$ and A are aligned, they get the same length after padding. The same situation happens between $Q2$ and B. Now we adjust the length of $Q1$ and $Q2$ by inserting space(s) in either of them accordingly to let them be exactly equal (i.e. the same character appears at same position.). When we insert a space in $Q1$, we must insert a space in A at the same position. When we insert a space in $Q2$, we also must insert a space in B at the same position. By this method, we can make sure that A and $Q1$ always have equal sequence length, and so do B and $Q2$. (See Figure 2.9). Finally we get that both A and B are aligned. This is the result that we want.

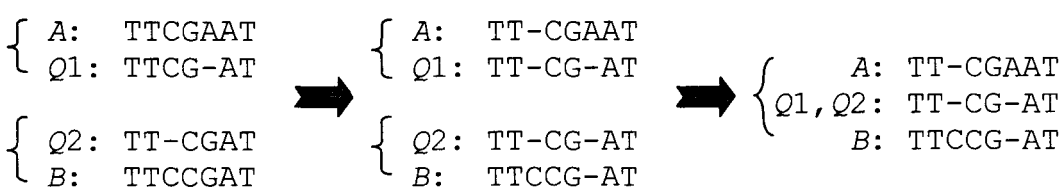


Figure 2.9: Adjusting sequence length.

To start multiple sequence alignment for all tree leaf sequences, first we should use any connected and aligned pair of node sequences to form a sequence set X. This sequence set has the properties that each member sequence has the same sequence length and all sequences in the set are aligned. Then we select any one of neighbor node sequences, say sequence A, and adjust its length against the sequence, say B, which is aligned and connected with A and belong to the set X. To do length adjustment, we need to add or insert space(s) in A or B so both of their lengths may expand to fit each other. When inserting a space into B, we have to insert a space into all other sequences of set X at the same position in order to keep them aligned and having equal length. After the length of sequence A is adjusted against B, A can be put into set X. Then we are going to find another neighbor node sequence and repeat above steps. Finally, all tree node sequences are adjusted to have the same length. By now all leaf sequences are in set X and are multi-aligned. Then we pick those leaf sequences out and output them. That is the result we are interested in. The multiple sequence alignment algorithm is shown in Figure 2.10.

Let's check the time complexity for this algorithm. Steps 1 - 6 take $O(1)$ time. Step 9 takes $O(1)$ time. Step 10 takes $O((P+1)N)$ time when we consider adding the P_{th} node sequence since we have to go through the whole sequence length N to make the adjustment and insert space for every member sequence in set X. Here N is the average length of the sequences. The number $P+1$ accounts for the total number of all existing members of X plus two more sequences s_i and a_{j2}. Step 7 has to consider all tree nodes. Therefore steps 7 - 13 take

Algorithm Multiple Sequence Alignment (T)

Input: Phylogenetic tree T with leaf sequences $\{s_1, \ldots, s_k\}$ and internal node sequences $\{s_{k+1}, \ldots, s_m\}$
Output: Multiple aligned leaf sequences $\{a_1, \ldots, a_k\}$

1. **begin**
2. **if** $k = 1$ **then** output one sequence and **end.**
3. **else if** $k = 2$ **then** output two pairwise aligned sequences and **end.**
4. **else if** $k > 2$ **then**
5. **begin**
6. Select two connected and aligned end node sequences a_1 and a_2 from any one tree edge to form a sequence set X.
7. **while** exist a node sequence s_i that is not in set X **do**
8. **begin**
9. Select a non X node sequence s_i that has a connected neighbor node sequence a_j in set X. (Assume a_j has two forms a_{j1} and a_{j2}, where a_{j1} is aligned with other members of X and a_{j2} is aligned with s_i.)
10. Adjust the length between a_{j1} and a_{j2} by means of inserting space(s) in either sequence to make their length equal. Any space(s) inserted in a_{j1} will be inserted in all member sequences of X at same position(s). Any space(s) inserted in a_{j2} will be inserted in s_i at same position(s).
12. Let $a_i = s_i$ and add a_i into X.
13. **end**
14. **end**
15. Output the optimal and multiple aligned leaf sequences $\{a_1, \ldots, a_k\}$.
16. **end.**

Figure 2.10: Multiple Sequence Alignment Algorithm

$$O(4N) + O(5N) + O(6N) + O(7N) + \ldots\ldots + O((m+1)N)$$

$$= O((4 + 5 + 6 + 7 + \ldots\ldots + (m+1))N)$$

$$\approx O(\tfrac{1}{2} m^2 N)$$

$$\approx O(2k^2 N) \quad \text{(here the tree is a similar to a complete binary tree, thus } m \approx 2k\text{)}$$

$$\approx O(k^2 N)$$

time complexity in the worst case, where m is the total number of nodes, k is the total number of input leaf sequences and N is the longest sequence length. Finally this algorithm takes $O(k^2N)$ time in the worst case.

2.5 Phylogeny Reconstruction

One of the tasks of TAAR is to infer and construct a phylogenetic tree that best represents the evolution relationship among the given set of sequences. Inferring a phylogeny is really an estimation procedure. Therefore we are making a "best estimate" of an evolutionary history based on incomplete information. In the context of molecular systematics, we generally do not have direct information about the past --- we have access only to contemporary species and molecules. Because we can postulate evolutionary scenarios by which any chosen phylogeny could have produced the observed data, we must have some basis for selecting one or more preferred trees from among the set of possible phylogenies. Phylogenetic inference methods seek to accomplish this goal in one of two ways: by defining a specific sequence of steps (algorithm) for constructing the best tree, or by defining a criterion for comparing alternative phylogenies to one another and deciding which is better (or that they are equally good) [8].

Here we consider the second way. This method has two logical steps. The first is to define the optimality criterion (objective function) for evaluating a given tree, i.e., the

value that is assigned and subsequently used for comparing one tree to another. The second is to use specific algorithms to compute the value of the objective function and to find the trees that have the best values according to this criterion (a maximum or minimum value, as appropriate). The price we pay for this method is that the methods tend to be much slower than those of the first method, a consequence of having to search for the tree(s) with the best value.

Most of the analytical techniques result in the inference of a non-rooted tree or non-rooted phylogeny, that is, a phylogeny in which the earliest point in time (the location of the common ancestor) is not identified. We generally use "tree" and "phylogeny" interchangeably. The parts of a phylogenetic tree go by a variety of names. The contemporary taxa correspond to *terminal nodes*, also called leaves, or external nodes. The branching points within a tree are called *internal nodes*. Nodes are called vertices or points. The branches connecting (incident to) pairs of nodes are also called edges. We will also use the term *peripheral branches* to refer to the branches that end at a leaf and the term *interior branches* to refer to branches that are not incident to a leaf.

A non-rooted, strictly bifurcating tree (one in which every internal node has degree 3) has K terminal nodes (corresponding to the taxa) and K-2 internal nodes. The tree has $2K$-3 branches, of which K-3 are interior and K are peripheral. The total number of distinct non-rooted, strictly bifurcating trees for K taxa [28] is

$$B(K) = \prod_{i=3}^{K} (2i - 5)$$

Adding a root adds one more internal node and one more interior branch. Since the root can be placed along any of the $2K$-3 branches, the number of possible rooted trees is increased by a factor of $2K$-3. Thus the exhaustive search for optimal trees is greatly time consuming and not practical.

Here we use heuristic stepwise addition of taxa to a growing tree to reduce computing time. First, any three taxa are chosen for the initial star tree. Next, one of the unplaced taxa is selected for next addition. Each of the three trees that would result from joining the unplaced taxon to the tree along one of its (three) branches is evaluated, and the one whose tree cost is optimal is saved for the next around (see Figure 2.11). In this next round, yet another unplaced taxon is connected to the tree, this time to one of the five possible branches on the tree saved from the previous round. The process terminates

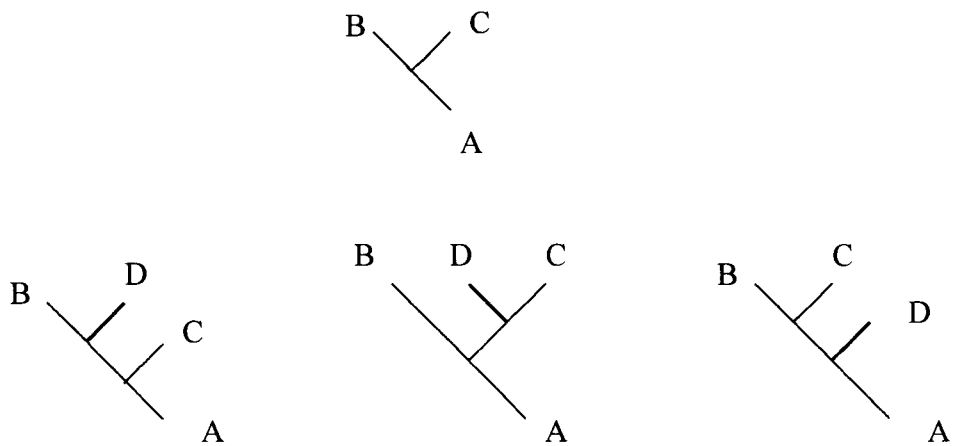


Figure 2.11: Enumeration of 3 possible unrooted trees for four taxa.

when all taxa have been joined to the tree. The phylogeny reconstruction algorithm is described in Figure 2.12.

```
Algorithm Phylogeny Reconstruction (X)

Input: X = {s1,..., sk} (sequences set)
Output: an unrooted phylogeny T with optimal tree score

var cost
1.  begin
2.    if k = 1 then construct and output a one-node tree and end
3.    else if k = 2 then construct and output a two-node tree and end
4.    else if k = 3 then construct and output a star tree and end
5.    else if k > 3 then
6.      use s1, s2 and s3 to construct a star tree T.
7.      for i = 4 to k do
8.      begin
9.        for each branch of the T do
10.       begin
11.         Join si to the branch.
12.         Do tree initialization.
13.         Do local optimization for the T.
14.         Calculate the tree cost Tc.
15.         if first time join si to the branch then
16.             cost = Tc and save tree T
17.         else if cost > Tc then
18.             cost = Tc and save tree T.
19.       end
20.     end
21.  Output the optimal tree T and score cost.
22. end.
```

Figure 2.12: Algorithm Phylogeny Reconstruction.

In this algorithm, we just randomly choose first three taxa for the initial star tree (here we select first three sequences) and later sequentially add the rest of the taxa in the same order in which they are presented in the data matrix. This strategy is also used in PHYLIP package [Joseph Felsenstein, University of Washington, Seattle, WA 98195, U.S.A.]. We only save one optimal tree after a taxa addition. By this early cut-off way, we can speed up computation time greatly.

Let's see the time complexity of this algorithm. Each of step 2 to step 4, and step 6 takes $O(1)$ time respectively. Step 11 takes $O(1)$ time. Step 12 takes $O(i^3 + i^2 N^2)$ time in the worst case (see algorithm in Figure 2.4). Here N means the average sequence length and i means total number of leaves on the current tree. The local optimization in step 13 takes $O((i\text{-}2) N^3) \approx O(i N^3)$ time in the worst case. Step 14 makes the summation of costs of all edges. So it takes $O(1)$ time. Step 15 to step 18 takes $O(1)$ time since it only makes comparison and saves the tree with the optimal cost. Thus steps 9-18 take

$$(O(i^3 + i^2 N^2) + O(i N^3) + O(1)) i$$

$$\approx O(i^4 + i^3 N^2 + i^2 N^3)$$

time. Since the running time of this algorithm is mainly contributed by steps 7 – 20, its time complexity is shown as follows:

$$O(4^4 + 4^3 N^2 + 4^2 N^3) + O(5^4 + 5^3 N^2 + 5^2 N^3) + \ldots\ldots + O(k^4 + k^3 N^2 + k^2 N^3)$$

$$= O(4^4 + 5^4 + \ldots\ldots + k^4 + (4^3 + 5^3 + \ldots\ldots + k^3) N^2 + (4^2 + 5^2 + \ldots\ldots + k^2) N^3)$$

$$\approx O(k (k+1)(2k+1)(3k^2+3k-1)/30 + (k (k+1) / 2)^2 N^2 + (k (k+1) (2k+1) / 6) N^3)$$

$$\approx O(k^5 + k^4 N^2 + k^3 N^3)$$

This time is a little high. But it is still feasible for a few sequences.

2.6 Generalized Tree Alignment

Given sequences X, the *generalized tree alignment* problem is to find a set of sequences Y and a loaded tree T (sequences from X are assigned to leaves and those from Y are

assigned to internal nodes of T) which minimizes the cost of T over possible sets Y and trees T. This problem is proved to be MAX SNP-hard [29]. One of the goals of this project is to make an approximately optimal multiple sequence alignment for a given set of sequences via tree alignment. But for tree alignment, we must have a phylogeny. Therefore we approach this problem by first computing an approximately optimal phylogenetic tree with all leaf nodes being assigned with the given sequences. Then we use the algorithm in Figure 2.10 to make a multiple alignment and output the result. The algorithm for this purpose is called Generalized Tree Alignment. It is shown in Figure 2.13.

Algorithm Generalized Tree Alignment (X)

Input: a set X of sequences $\{s_1, \ldots, s_k\}$
Output: multiply aligned sequences $\{a_1, \ldots, a_k\}$

1. **begin**
2. Call Phylogeny Reconstruction algorithm (see Figure 2.10).
3. Call Multiple Sequence Alignment algorithm (see Figure 2.8).
4. Output multiply aligned all member sequences of set X.
5. **end.**

Figure 2.13: Generalized Tree Alignment algorithm

The algorithm's time complexity is calculated as follows. The Phylogeny Reconstruction algorithm takes $O(k^5 + k^4N^2 + k^3N^3)$. The Multiple Sequence Alignment Algorithm takes $O(k^2N)$. So the total time complexity of Generalized Tree Alignment algorithm is

$$O(k^5 + k^4N^2 + k^3N^3) + O(k^2N) = O(k^5 + k^4N^2 + k^3N^3).$$

In this project, there are two sets of operations with similar functions. One has prefix ***fast***. The other one has prefix ***optimal***. The difference here is mainly in the Tree Alignment algorithm. In the *optimal* version of the algorithm as given in section 2.3, we would do local optimization for the whole tree after tree lifting. In the *fast* version of the algorithm, we would not do local optimization after tree lifting. Thus the time complexity of this algorithm is about $O(k^3 + k^2N^2)$, where k is the total number of input sequences and N is the average length of the sequences. Hence the fast versions of Phylogeny Reconstruction and Generalized Tree Alignment have a total time complexity $O(k^5 + k^4N^2)$.

Chapter 3

Graphical User Interface

In a software system, the user interface is considered as the mechanism through which a dialogue between the computer and the user is established. It plays a vital part in the software system's efficiency. The specialty called Human-Computer Interaction (HCI) has emerged as the study of people, computer technology and the ways they influence each other. Both the developers of software systems and users accept that just being able to do a task on a computer is not the only important factor. Interface plays a very prominent role in a computer software system. It is in many ways the "packaging" for a computer system. If it is easy to learn, simple to use, straightforward, and forgiving, the user will be inclined to make good use of what is inside and be highly impressed. Now a days, the graphical user interface is also considered as contributing to communication visually.

In this chapter, we will briefly review some interface style and development issues in general and then review the architecture of the X Window System, MOTIF Programming Language and their fundamentals that are related to this project.

3.1 Interface Development

The term "user interface" can be defined as the software component of an application that translates user's actions into requests for functions, and that provides to the user feedback about the consequences of his/her action.

A good user interface should provide an end user with a facile, natural environment for conducting various tasks fast, efficiently, accurately, and inexpensively. The nature of the software component of the user interface has been driven and limited by the hardware component. As hardware has become more sophisticated, options for interaction style have grown.

The following are some of common interface styles:

- **Command and query interface**: Communication is purely textual and is driven via commands and responses to system-generated queries.
- **Menu interface**: The set of options available to the user is presented on the screen. An option is selected by either using the mouse or typing some key. Since the options are visible, they are less demanding on the user, relying on recognition rather than recall.
- **Form-fills and spreadsheets**: The user is presented with a display comprising a grid of cells, each of which can contain a value. This type of interface is used primarily for data entry and data analysis application.
- **WIMP interface**: This type of user interface is characterized by windows, icons, menus and pointing devices. It is the default interface style for the majority of interactive systems in use today. The important features are (1) displaying different types of information simultaneously; (2) enabling the user to switch context without losing visual connection with other work; (3) enabling the user to perform various tasks in a facile manner; (4) increasing the interaction efficiency.

Each of them is encountered across every application area. The trend is toward multitasking, window-oriented, and point-and-click interfaces. Ideally, users can customize the interface to suit their working style, rather than adapting their own working style to accommodate the interface's way of doing things.

The most important and natural method for a user interface development is the iterative development methodology, which includes building one or more prototypes to get requirement specification and comments from clients. To make the user interface easier to program, many different kinds of tools have been created. These include X Window systems, toolkits, Motif programming language, and other interface builders.

The survey conducted by Myers and Rosson [17] seems very interesting. It has shown that in today's applications, an average of 48% of the code is devoted to the user interface portion. The average time spent on the user interface portion is 45% during the design phase, 50% during the implementation phase, and 37% during the maintenance phase.

3.2 The X Window System

The X Window System, called X for short, is a network-based graphics window system that was developed at MIT in 1984. Since then, several versions of X have been developed. It has been adopted as an industry standard windowing system that allows programmers to develop fundamental portable graphical user interfaces. Till now, A consortium of industry leaders has been continually directing and developing X Window System.

The X Window System's architecture is based on the client-server model. A single process, known as the *server*, is responsible for all input and output devices. An application that uses the facilities provided by the X server is known as a *client*. The syntax and semantics of the conversation between servers and clients are defined by *X Protocol*. Clients use the protocol to send requests to the server to create and manipulate windows, to generate text and graphics, to receive input from the user, and to communicate with other clients. The server uses the protocol to send information back to the client in response to various requests and to deliver keyboard and other user input on to the appropriate clients. The X Window System allows clients to be run on any machine in a computer network, and be displayed on any other machine(s) in that network.

The X protocol has been implemented with a library so those application programmers do not have to think in low-level terms. This library provides a procedural interface that conceals many of the details of the protocol. Various utility functions are also provided that are not protocol-related but important in building applications. The exact interface for the library may differ for each programming language. The C libraries are the most widely used. They include a low-level procedural interface to the X protocol called Xlib, which defines an extensive set of functions that provide complete access and control over the display, windows, and input devices.

Although programmers can use Xlib to build applications, this relative low-level library can be tedious and difficult to use correctly. Many programmers prefer to use the higher-level X Toolkit to mask some of the complexity of the X protocol. A toolkit is a

collection of prewritten functions that implement all the features and specifications of a particular GUI. The X Toolkit consists of two parts: a layer known as the *X Toolkit Intrinsics* (Xt), and a set of user-interface objects called *widgets* and *gadgets*. The widget and gadget set provide a convenient interface components for creating and manipulating X Windows, colormaps, events, and other cosmetic attributes of display, while the Xt provides a framework that allows the programmer to combine these components to produce a complete user interface. In short, widgets can be thought of as building blocks

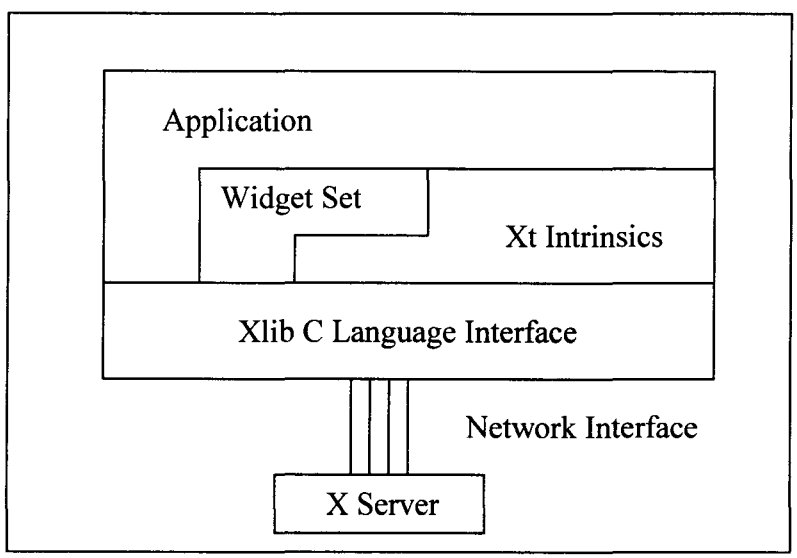


Figure 3.1 Programmer's view of the X Window System

that the programmer uses to construct a complete application. Figure 3.1 shows the architecture of an application based on a widget set and the Xt.

An important X concept, which needs to be introduced here, is the *event*. An event is a notification, sent by the X server to a client, that some condition has changed. The server generates events as a result of some user input, or as a side effect of a request

to the X server. The server sends each event to all interested clients, who determine what kind of event has occurred by looking at the type of the event. To receive events, applications must specifically request the X server to send the types of events in which they are interested. Most X applications are completely *event-driven* and are designed to wait until an event occurs, respond to the event, and then wait for the next event. The event-driven approach provides a natural model for interactive applications. The user does not need to navigate a deep menu structure and can perform any action at any time. The user, not the application, is in control. The application simply performs some setup and goes into a loop from which application functions may be invoked in any order as events arrive.

3.3 Motif Programming Model

The widgets that Xt provides are generic in nature and impose no user-interface policy whatsoever. That is the job of a user-interface toolkit such as Motif. Motif is a set of guidelines that specifies how a user interface for graphical computers should *look*. The job of Motif is solely to provide a consistent appearance and behavior for user-interface controls. So people do not have to use the X Window System to implement a Motif-style graphical user interface. However, to enhance portability and robustness, the Open Software Foundation (OSF) chose to implement the Motif GUI using X as the window system and the X Toolkit Intrinsics as the platform for the Application Programmer's Interface (API). We already know that Xt provides an object-oriented framework for creating reusable, configurable user-interface components called widgets. Here Motif

provides widgets for such common user-interface elements as labels, buttons, menus, dialog boxes, scrollbars, and text-entry or display areas. In addition, there are widgets called managers, whose only job is to control the layout of other widgets, so the application doesn't have to worry about details of widget placement when the application is moved or resized.

A widget operates independently of the application, except through prearranged interactions. For example, a button widget knows how to draw itself, how to highlight itself when it is clicked on with the mouse, and how to respond to that mouse click. The general behavior of a widget is defined as part of the Motif library. Xt defines certain base classes of widgets, whose behavior can be inherited and augmented or modified by other widget classes (subclasses). The base widget classes provide a common foundation for all Xt-based widget sets. A *widget set,* such as Motif's Xm library, defines a complete set of widget classes, sufficient for most user-interface needs. Xt also supports lighter-weight objects called *gadgets*, which for the most part look and act just like widget, but their behavior is actually provided by the manager widget that contains them. Most widgets and gadgets inherit characteristics from objects above them in the class hierarchy. For example, the Motif PushButton class inherits the ability to display a label from the Label widget class, which in turn inherits even more basic widget behavior from its own superclasses.

The complete set of widgets that provided by Motif toolkit is designed to implement the particular application user-interface style that is specified in two documents: the *Motif Style Guide*, which defines the external look and feel of

applications, and the *Application Environment Specification*, which defines the application programmer's interface (API). Motif toolkit may be implemented on a wide range of computer platforms and enables programmers to produce completely Motif-compliant applications in a relatively short amount of time. The Motif interface was intentionally modeled after IBM's Common User Access (CUA) specification, which defines the interface for OS/2 and Microsoft Windows.

In this project, since the application we are developing is mainly used on UNIX and X Window platforms, and Motif is mainly for X Window environment, so we chose Motif Xm library toolkit to implement the user-interface of our application software. The Xlib and Xt are also used as a complete system for constructing our user interfaces. Most Motif applications are worked out same as our way.

Managing resources is an important part of programming with X. Widgets are designed so that many of their resources can be modified by the user at run-time. Resources are named data units that specify widget attribute values such as colors, fonts, images, text, positions and sizes of windows, or any customizable parameter that affects the behavior of the application. Resources can be set in four ways:

- In the application code when/after the widget is created.
- Through the resource database.
- In a command line option.
- Dynamically while the application is running.

If a resource is not set in any of these ways, Motif will set the resource to a default value. Setting resources in application code is considered as *hard* coding. Users can not

customize hard coded resources unless the application code is modified and recompiled. Therefore, the application programmer should only set the resource values in program for the resources that are not allowed to be changed by a user. To make programs customizable, a good approach is to provide an application default resource file for every program so those users can customize an application by simply changing the appropriate entries in the resource file. By convention, this file is stored in the directory */usr/lib/X11/app-defaults* and it has the same name as the application with the first letter capitalized.

When a Motif program is initialized, the connection to the X server is set and the resource database is created and embedded with the program. The resource specifications in the user's resource files are loaded into the resource database. There are four resource files and they are the *application defaults file* whose path can be identified by the environment variable XFILESEARCHPATH, the *per-user application defaults file* whose path is identified by the environment variable XUSERFILESEARCHPATH, the *user's defaults* which is the file **.Xdefaults**, and the *user's per-host defaults* whose path can be identified by the environment variable XENVIRONMENT.

Most widgets are prolific in their use of resources. For each widget class, there are many resources that neither the application nor the user should ever need to change. Among those resources, the *callback resources* for a widget are a particularly important class of resources that must be set in the application code. A widget that expects to interact with an application provides a callback resource for each type of interaction it supports. An application associates a function with the callback resources in which it is

interested; the function is invoked when the user performs certain actions in the widget. For instance, a PushButton provides a callback for when the user activates the button. Note, however, that not every event that occurs in a widget results in a callback to an application function. Widget is designed to handle many events themselves, with no interaction from the application. For example, a Text widget typically provides a complete set of editing commands via internal widget functions called *actions*. Actions are mapped to events in a *translation table*. This table can be augmented, selectively overridden, or completely replaced by settings contained in the implementation of a widget class, in application code, or in a user's resource files.

As we mentioned before, a Motif user interface is created using both the Motif Xm library and the Intrinsics' Xt library. Xt provides functions for creating and setting

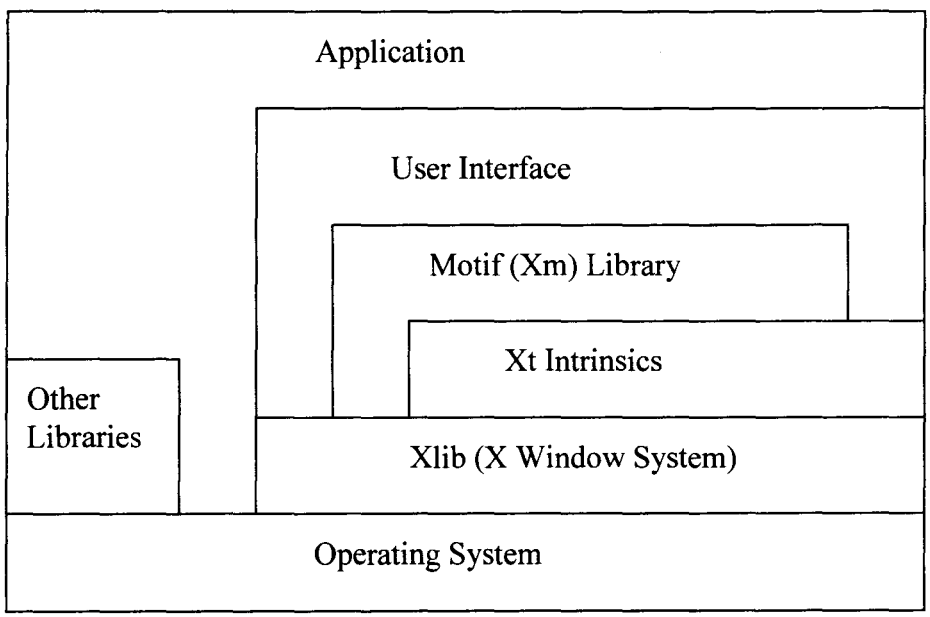


Figure 3.2 User-interface library model

resources on widgets. Xm provides the widgets themselves, plus an array of utility routines and convenience functions for creating groups of widgets that are used collectively as single user-interface components. An application may also need to make calls to the Xlib layer to render graphics or get events from the window system. In the application itself, rather than in the user interface, you may also be expected to make lower-level system calls into the operating system, file system, or hardware-specific drivers. Thus, the whole application may have calls to various libraries within the system. Figure 3.2 represents the model for interfacing to these libraries.

Chapter 4

Project Development

This project is called Tree Alignment and Reconstruction (TAAR). It is a software application mainly based on Motif Intrinsics Toolkit, which follows an object-oriented and event-driven model. In general, such an application consists of three parts:

1. Creating and manipulating Motif widgets to build the desired user interfaces;
2. Using C, C++, Xm, Xt and Xlib to develop the code of application's working flow and callback actions;
3. Attaching the application code to the user interface via callback procedures and event handlers that are executed when the user performs some action on a widget.

4.1 Interface Structure

Let's see the widget hierarchy used in TAAR. In Figure 4.1, we divide the initial screen of TAAR into four main areas: **Main Menu Area**, **Drawing Area**, **Message Panel**, and **Operation Area**. The hierarchical structure is depicted in Figure 4.1 and explained below. Motif toolkit contains some widget class sets called Shell, Manager, Primitive and Gadget widget class sets. The Manager widget class set mainly includes MainWindow, Form, DrawingArea, PanedWindow, RowColumn, Frame, SelectionBox and Scale classes. The Primitive widget class set mainly includes PushButton,

CascadeButton, ArrowButton, ToggleButton, DrawnButton, Label, List, Scrollbar, Text and TextField classes.

In Figure 4.1, the *toplevel* is the application's top-level window created by the call to initialize the toolkit. It is a Shell class widget that handles the application's interaction with the window manager and acts as the parent of all of the other widgets in the application. Thus a shell widget serves as a wrapper around its child widgets. The *Main Window* is a manager widget and belongs to the MainWindow widget class that belongs to Manager super widget class. It is created from and is the child of the *toplevel* widget.

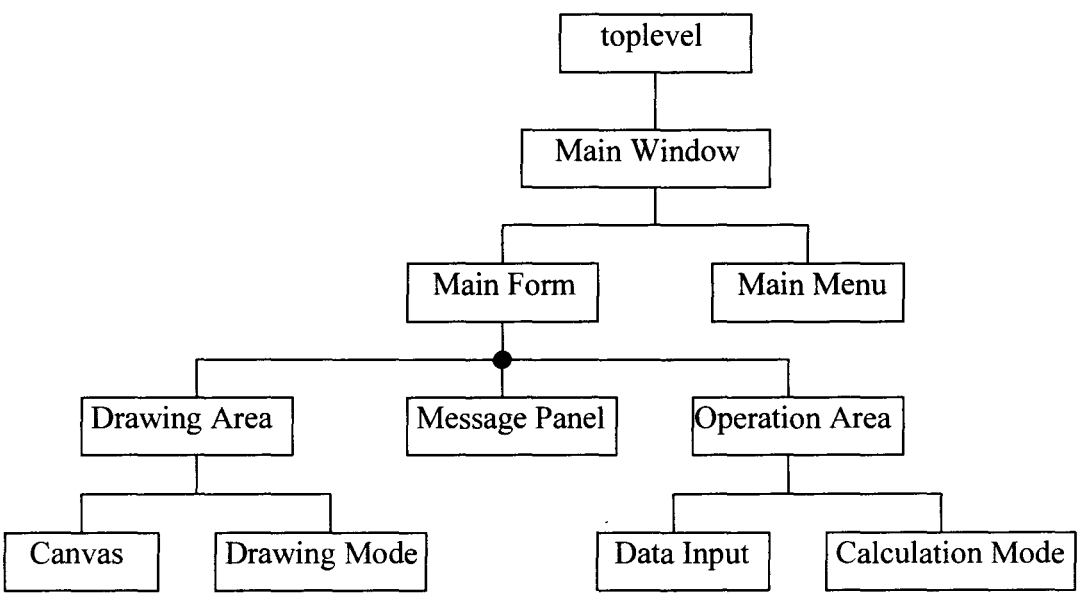


Figure 4.1 Primary structure of TAAR

The MainWindow widget acts as the standard layout manager for the main window of an application. It is specially tuned to pay attention to the existence of a MenuBar, a

command area, a message area, a work region, and ScrollBars, although all of these areas are optional. In application TAAR, we only keep a menu bar and a work region.

The *Main Window* manages two children of *Main Menu* and *Main Form* area in a row. Here the *Main Form* is a widget instance of another manager widget class called Form class. The Form widget provides a great deal of control over the placement and sizing of the widgets it manages. A Form can lay out its children in a grid-like manner or it can allow its children to link themselves to one another in a chain-like fashion. Form uses constraint resources to specify how children are resized and positioned relative to each other and the Form as a whole. In TAAR, *Main Form* manages three children areas. They are *Drawing Area*, *Message Panel* and *Operation Area*. The *Message Panel* and *Operation Area* have a fixed size. The *Drawing Area* has a relative size. The size of the *Drawing Area* can change when *Main Form* window size changes. This has a certain advantage since the size of drawing canvas can be adjusted automatically when main window size is changed.

The *Main Menu* contains a menu bar that is created for some application operations such as Open a file, Exit and Help. This menu bar layout is managed by MainWindow widget. In the Motif toolkit, a MenuBar is not implemented as a separate widget, but as a set of CascadeButtons arranged horizontally in a RowColumn widget. Each CascadeButton is associated with a PulldownMenu that can contain PushButtons, ToggleButtons, Labels, and Separators. The managing RowColumn widget has a resource setting indicating that it is being used as a MenuBar. One does not need to know any specific details about any of these widgets in order to create a functional

MenuBar, since Motif provides convenient routines for creating self-sufficient menu systems.

In the *Drawing Area*, a set of drawing related operation push buttons and a DrawingArea widget are contained. The DrawingArea widget, which also be called *Canvas*, provides an area in which an application can display graphics. Callback routines can be used to notify the application when expose and resize events take place and when there is input from the keyboard or mouse. The DrawingArea can also be used to manage the geometry layout for child widgets, but its functionality in this area is quite limited. Here we use this drawing canvas to draw, manipulate and display any tree on it. In this area, the events *ButtonPress*, *ButtonMotion*, *ButtonRelease* are all the events in which this application is interested. *Event Handlers* are invoked by the Xt Intrinsics when a specific type of event occurs. The event type and associated application function are registered using the *XtAddEventHandler* function. The related drawing operation buttons take care of tree loading, saving and drawing modes.

The *Message Panel* contains a Text widget to display application related messages. So user can see current application running instructions and messages.

The *Operation Area* contains two main child areas. One is *Data Input*; another one is *Operation Mode*. In *Data Input* area, there is a PanedWindow widget that contains two separate List windows. One is for sequence list input; another one is for calculation score matrix list input. These lists are scrollable. The PanedWindow widget manages its children in a vertically tiled format. Its width always matches the widest widget in its list of managed children; the widget forces all of its children to stretch to the same width as

that widget. Each pane in a PanedWindow contains a child widget; every pane has an associated sash (or grip) that allows the user to change the height of the pane interactively. Resizing a pane with the grip can cause the widgets in other panes to change size. In TAAR, we use this PanedWindow widget to save window area since sequence list and score matrix list may be long and either one of these two child window height can be adjusted to get large viewing area.

In *Operation Mode* main area, it contains an OptionMenu widget for selecting operation mode, and a push button for starting calculation. An OptionMenu is similar to a PulldownMenu in that they are both associated with CascadeButtons. However, there are also several major differences between the two types of menus. In an OptionMenu, the CascadeButton is not part of a MenuBar. Instead, it is created as the child of a RowColumn widget that also contains a Label. Another difference is that the menu pops up on top of the CascadeButton, instead of dropping down from it. The label on the CascadeButton is one of the elements in the menu; the CascadeButton displays the current menu selection. The Motif toolkit handles the management of the PulldownMenu for the OptionMenu, so its handle is not available to us, nor does it need to be. Because of the design of the OptionMenu, it cannot have cascading menu. We use this kind of menu to save space on the main window.

Moreover, there are other interface areas brought up by popping up dialog boxes. You can think of dialog boxes as an application's *secondary windows*. Since dialogs are not meant to remain on the screen for very long, they do not need all of the decorations that are typically provided by the window manager. However, dialogs are not completely

independent like menus, so they do need to be controlled by the window manager. For example, if an application is iconified, its dialog boxes are typically iconified as well. Dialog boxes are usually implemented in Xt using TransientShells. Motif provides two main types of dialog boxes: message dialogs and selection dialog. Message dialogs are designed to allow an application to communicate with the user, while selection dialogs prompt the user to enter different types of information. It is also possible to create custom dialogs for specialized application functionality. Dialog are not visible until a certain user command is given, or a situation arises in which the program requires user input. In TAAR, dialog boxes are used for a variety of purposes including (1) prompting user for input/output file names; (2) viewing/inputting/modifying sequences and score matrices; (3) displaying sequence alignments; (4) displaying any message; and (5) getting confirmation from users if some concerned action is going to happen such as pressing the Clear button.

4.2 Displaying Graphs

In TAAR, there is a graph editor. The user is able to draw, edit and display a (undirected) graph in the drawing area in which vertices are represented by circles and edges are represented by lines.

In X Window programming, drawing things like points and lines can be easily done by first creating a *graphics context* (GC) and then calling Xlib graphics functions. The GC is an X Window System resource which contains 23 distinct attributes to specify

things like color and line width. When one object could be over displayed by another, the GC's GCFunction attribute should be considered since it specifies how each pixel of a new image is combined with the current contents of a destination. This attribute is commonly set as the *XOR* mode so that drawing an object twice would restore the screen to its original state. Here we use this property to erase an image object and perform rubber-banding operation in our TAAR system.

To draw an object, the coordinates have to be specified in pixel units. Coordinates are always relative to the upper left corner of a drawing canvas window. The x coordinate increases toward the right and the y coordinate increases downwards.

Now we are going to explain how vertices and edges are displayed in TAAR. A vertex (also called a node) is represented on the screen as a circle with a radius of r (r = 5 pixels in TAAR). When clicking the mouse to create a vertex, the circle is displayed such that its center is at the position of the hotspot of the cursor. The center's coordinates are also called the coordinates of the vertex and stored as a part of the internal representation of the vertex.

An edge is represented on the screen by a line connecting two nodes. Internally, it is represented by a relation between two vertices. Please notice that we cannot simply draw a line connecting the centers of two nodes since the overlapping parts between the line and nodes will not be displayed properly. The natural choice is to choose a boundary point from each node. However, to ease the calculation involved, it is much easier to simply pick up a boundary point from the surrounding rectangle of each node. Therefore

we choose the two ending positions of the line as follows. Assume two vertices are of coordinates (x_1, y_1) and (x_2, y_2) and displayed as nodes of radius r. Define the slope as

$$k = \frac{y_2 - y_1}{x_2 - x_1}.$$

Define $\sigma_x = 1$ if $x_1 \le x_2$ and -1 otherwise, and $\sigma_y = 1$ if $y_1 \le y_2$ and -1 otherwise.

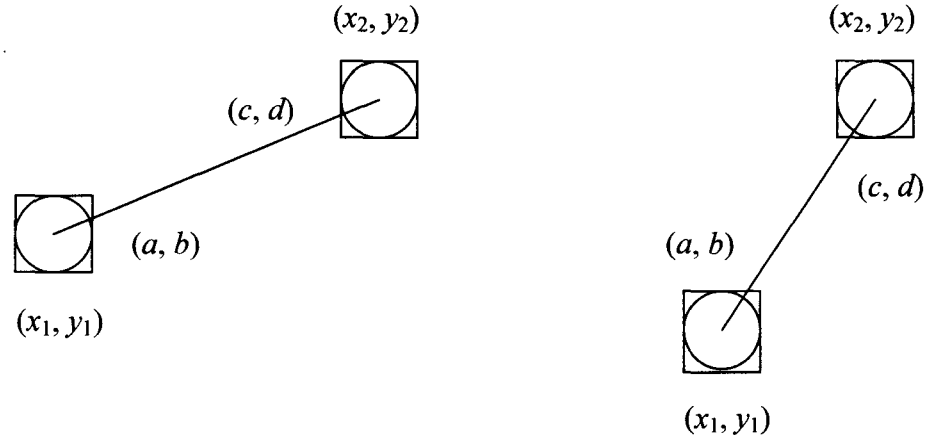
If $-1 \le k \le 1$, as shown in Figure 4.2(a), we calculate

$$a = x_1 + \sigma_x r\,; \quad b = y_1 + \sigma_x r\, k\,; \quad c = x_2 - \sigma_x r\,; \quad d = y_2 - \sigma_x r\, k\,;$$

If $k > 1$ or $k < -1$, as shown in Figure 4.2(b), we calculate

$$a = x_1 + \sigma_y \left\lfloor \frac{r}{k} \right\rfloor; \; b = y_1 - \sigma_y r\,; \quad c = x_2 - \sigma_y \left\lfloor \frac{r}{k} \right\rfloor; \; d = y_2 + \sigma_y r\,;$$

Then the line connecting points (a, b) and (c, d) is displayed as the edge connecting vertices (x_1, y_1) and (x_2, y_2).



(a) $-1 \le k \le 1$ (b) $k > 1$ or $k < -1$

Figure 4.2: Deriving the line for an edge.

Once nodes and lines are drawn, information about them, such as node coordinates and connection relationship, has to be saved by the application program in some internal data structures since the workstation has no memory of the fact that something is drawn. The workstation has to use this information to redraw all those drawn objects when the window needs refresh its display.

Manipulating vertices and edges is conducted by first selecting corresponding nodes and lines. Then procedures are invoked by event handlers to perform some actions. From the user's point of view, a selection can be done by using the mouse to point to the node or line that is displayed on the screen and then click the mouse button. However, from the programmer's point of view, the program has to do a lot of background computation to compare the user selected cursor position with the coordinates of the vertex or edge and see if the difference is small enough (as predefined). Here the concern is whether the hotspot of the cursor is within the neighborhood of the intently selected node or line object. As the mouse is clicked, every object will receive event messages. Any object with cursor hotspot standing in its neighborhood would be selected to accept some actions. How is the neighborhood defined for a vertex and an edge? Assume a vertex has (a, b) as its coordinates, then its neighborhood is defined as $\{(x, y)| |x-a| < \text{r and } |y-b| < r\}$ (r = 5 pixels in TAAR). The neighborhood of an edge will require much more computation. It is defined as a $2*r$ width band surrounding the line segment. Assume the two ends of the line are (a, b) and (c, d) as shown in Figure 4.2, then the line's algebraic equation is

$$\frac{y-b}{x-a} = \frac{b-d}{a-c},$$

which can be rewritten as

$$(b-d)x + (a-c)y + cb - ad = 0.$$

The distance between a point (x_o, y_o) and the line is calculated by

$$\text{dist}(x_o, y_o, a, b, c, d) = \frac{|(b-d)x_0 + (a-c)y_0 + cb - ad|}{\sqrt{(b-d)^2 + (a-c)^2}}.$$

Here we can define the neighborhood of the edge (see Figure 4.3) to be

$\{(x, y) \mid \text{dist}(x, y, a, b, c, d) < r,\ \min(a, c) < x < \max(a, c),\ \min(b, d) < y < \max(b, d)\}$.

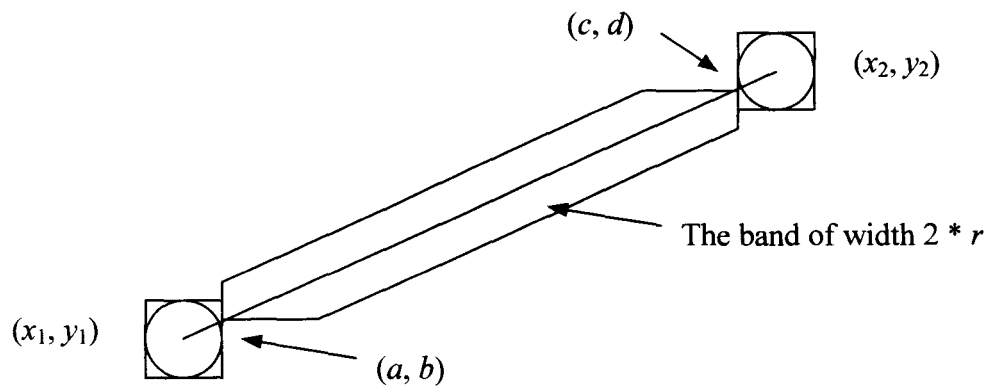


Figure 4.3: The neighborhood of an edge.

In above figure, we can see that any mouse click event happens with the hotspot of the cursor located in the band of $2*r$ width of a line segment will let the line segment be selected and accept consequent actions.

4.3 Data Structure

The primary data structures used in TAAR will be discussed in this section. Several global data structures are defined and implemented in TAAR for the whole application. They are used for dynamic data storage of individual sequence and the list of all sequences, individual score matrix arrays and the list of them, nodes, edges and tree structures.

4.3.1 Sequence

The common set of information of sequences includes the original name, the title name when belonging to a tree node, the sequence itself, the sequence length, the sequence format, and the type of the sequence. The data structure for sequences is defined as type **SEQtype.** It forms an array as *seqList*. This list array (*seqList*) is used to store all input sequences. The SEQtype data structure is showed in Figure 4.4.

```
typedef truct {
        char        name[MAXLENGTH];     /* sequence name */
        char        title[MAXLENGTH];    /* auxiliary name */
        char        *content;            /* character sequence */
        int         length;              /* sequence length */
        short       format;              /* sequence format */
        boolean     isdna;
}SEQtype;
```

Figure 4.4: The data structure for sequences.

4.3.2 Score Matrix

The information needed for storing calculation-needed score matrix is the score matrix name, the characters considered for calculating scores, and the score values represented in a two dimensional array. The data structure for storing this information is declared as structure *score*. The arrays form an array as *scoreList*. This list is for storing all input score matrices. The data structure is showed in Figure 4.5.

```
struct score {
        char    name;                           /* score matrix name */
        char    order[MaxLetters + 1];          /* character set */
        float   matrix[MaxLetters][MaxLetters]; /* score value matrix */
};
```

Figure 4.5: The data structure for score matrix.

4.3.3 Tree

The tree in TAAR is undirected and weighted. A vertex contains various information such as node coordinates at the drawing area, the sequence associated with it, and the edges connected to it. In the information associated with an edge, it contains the alignment of two sequences assigned to the both ends of the edge. Here we use the *adjacency lists* data structure to represent a tree. That is, each vertex is associated with a linked list consisting of all the edges incident on this vertex, and all vertices are stored in an array. The data types that are used to describe the adjacency lists structure is shown in Figure 4.6.

```
typrdef struct _LIST{
        int             index;          /* index of the vertex */
        float           weight;         /* cost of the alignment */
        char            *seq1;          /* padded sequence 1 after alignment */
        char            *seq2;          /* padded sequence 2 after alignment */
        PARAtype        *para;          /* parameters for doing alignment */
        struct _LIST    *next;          /* next edge */
}LISTtype;

typrdef struct _NODE{
        int             _x;             /* x-coordinate */
        int             _y;             /* y-coordinate */
        SEQtype         *con;           /* sequence assigned to the vertex */
        LISTtype        *head;          /* edge list associated to the vertex */
}NODEtype;
```

Figure 4.6: The adjacency lists structure.

The tree is represented by **NODEtype** *nodearray*[MAXNODE]. It is initialized in the array with the size of MAXNODE and type of NODEtype. The offset of each element in the array coincides with the vertex index label. This permits direct access to vertex data and thus reduces the searching operations that would otherwise be used so frequently in many functions. The global variable *nodes_count* is used to keep the number of vertices of the graph. Existing vertices are always kept in the first *nodes_count* cells of *nodearray*[MAXNODE].

TAAR has some predefined global constants that need the user's attentions. They are:

- **MAXNODE:** The total node number of all trees in the drawing area. Its value must be greater than the actual number of the drawing nodes and be greater than two and half times the MaxSeqNum constant.

- **MaxSeqNum:** The maximum number of sequences to be input in SEQUENCE LIST. Its value must be greater than the actual number of the input sequences.
- **MaxScoreNum:** The maximum number of score matrices to be input in SCORE MATRIX LIST. Its value must be greater than the actual number of input score matrices.
- **MaxLetters:** The maximum number of alphabet letters in any input score matrix. Its value must be greater than the actual number of alphabet letters in any input score matrix.
- **WIDTH:** Drawing area width in pixel unit. Its value must not exceed the positive value range of a machine dependent *int* data type.
- **HEIGHT:** Drawing area height in pixel unit. Its value must not exceed the positive value of a machine dependent *int* data type.
- **OPT_TIMES:** The iteration times of doing star alignment optimization for a tree. Its value ranges from 1 to any reasonably large positive integer number.

The default values are provided for these constants. These constant values are modifiable and may affect the operations of TAAR program. The user may modify them according to the user's requirement before compiling the code of TAAR. For instance, OPT_TIMES will significantly affect the execution time of TAAR. We choose value 4 as its default value in TAAR since our testing has shown that the tree cost usually stabilizes after four rounds of optimization across the tree.

4.4 Operations

Let's give a general idea about how TAAR's drawing operations are done. First, the insertion, deletion, moving, label, unlabel, root operations on a tree are illustrated as follows:

- Add a vertex: Coordinates and/or a sequence of the vertex are inserted into the *nodes_counts*-th cell of *nodearray*[] and *nodes_count* is incremented by 1.
- Add an edge: Assume that it connects the *i*th and the *j*th vertices. The program will allocate memory space for one LISTtype structure, fill it with the label of the *j*th vertex and other edge related information, and insert it into the beginning of the adjacency list of the *i*th vertex. The process is repeated with the *i* and *j* exchanged.
- Delete a vertex: All edges connecting to the vertex are deleted from the adjacency list of the vertices connected to the deleted vertex. The information related to the deleted vertex in *nodearray*[] is deleted. The last vertex in *nodearray*[] is moved to the cell of the deleted vertex. The adjacency lists of the vertices adjacend to the former last vertex are updated. *nodes_count* is decremented by 1.
- Delete an edge: The adjacency lists associated with each of the two ends are searched. The involved list items, one from each list, are removed and their memory spaces are released.
- Moving a vertex: The coordinates information of this vertex will be updated in the according cell of *nodearray*[].

- Label a node: Here the labeling means assigning the sequence and its information to a node. When doing labeling, the selected sequence and its information (i.e. name and length etc.) would be copied into the related node data structure of *nodearray*[].
- Unlabel a node: This means deleting the sequence and its information on a node. When doing unlabeling, the sequence and its information that are assigned onto this node sometime before would be deleted from the related node data structure of *nodearray*[].
- Root: This operation redraws the graph so that it looks like a rooted tree. The algorithm for this operation is that it starts from the selected root node and implements Depth First Search for the whole graph to get the node relationship in the graph. Then from the selected root node position coordinates, it assigns position coordinates to each node and increases the distance of the same level nodes while node level increases.

All above operations can be executed after one makes the selection of the drawing mode on the draw mode menu buttons. This mode is saved in a global variable so all functions know what the drawing mode the user is working on. The user must make a clear selection before doing any tree operation. A tree drawn on the screen can be stored into a file with a certain customized format. Later this file can be retrieved back and the tree as well as its attached sequences will be displayed again in the drawing area. This is a convenient feature for the user to store the tree and its attached sequences.

Second, the input sequence list window and score matrix list window have related sizes. Due to the limited area of user interface screen, we have implemented a Pane window to contain these two list windows. So user can adjust their size relatively by drag a small button of the Pane window. This way, the user can get relatively large view size for either one of these two list windows.

In these two list windows, each of them contains an operation pull-down menu. The user can open, save, edit and delete the input sequences and matrices in the list. The user can also assign any one of the list sequences to a node. When assigning a sequence to a node, the sequence in the input sequence list *seqList*[] is actually copied into the SEQtype node data structure of that node index cell in *nodearray*[]. So later we can use this sequence directly to calculate alignment.

Third, the operation mode or calculation mode of TAAR is specified by a selection button. This way, it can save space, and the user can easily click the button to make a selection. After the operation is selected, the mode is stored in a global variable so all functions know the current operation.

4.5 The Undoing Facility

In TAAR, we provide a convenient common feature, Undo and Redo. These facilities allow users to cancel a command, to recover from operating mistakes that may damage and also allow users to do input testing, knowing that they can back up easily if the result is not what they had expected.

To perform an Undo operation, we have to store or back up the relative data and information for each operation done by users. This way, we can let the user recover old data. For example, we have to save a deleted data so the user can undo a delete operation. In TAAR, we have developed a one-level undo facility for the graph editor. Any graph drawing related operation can be undone up to one level. Executing undo twice successively will let system restore the state as if the undo operations were not executed.

There are five graph drawing modes for which we are interested in providing undo operation. They are drawing or moving a node (Node), drawing an edge (Edge), assign a sequence to a node (Label), delete the sequence from a node (Unlabel), and delete a node or edge (Cut). According to the actual amount of information to be saved, we define the data type as shown in Figure 4.7. The *union* construct is used to allow storage sharing and a data member called *flag* is used for the interpretation of the stored information.

A memory space *undo_buffer* is used to save drawing information. When a drawing or editing operation is performed, we first check which part of the graph will be changed and save the old data in *undo_buffer*, and then update the graph as the drawing operation required. For the undo operation, the most expensive editing is the deletion of a node, which involves deleting the node and the edges connecting to it from nodearray[], saving sequence and other information of these deleted node and edges into *undo_buffer*, and rearranging the vertex array of the graph.

```
struct undo_struct {
        UNDOTYPE        flag;           /* one of five drawing modes */

        union{
         struct {                       /* node is moved */
           int          index;
           int          x, y;
         }move_node;

         struct {                       /* node's content is changed */
           int          index;
           SEQTYPE      *content;
         }change_con;

         struct {                       /* changing edge  */
           int          index;
           int          index2;
           float        weigh;
           char         *seq1,  *seq2;
           PARAtype     *para;
         }re_edge;

         struct {                       /* changing node */
           int          index;
           NODEtype     node;
         }re_node;

        }set;

}   undo_buffer;
```

Figure 4.7: Structure of *undo_buffer*

Chapter 5

A Guide to Tree Alignment and Reconstruction

In this chapter, we will make a brief introduction to Tree Alignment and Reconstruction (TAAR) application. This software system is intended to be installed on a UNIX operating system with X Window (X11R5) and MOTIF 1.2 (or higher version) supporting environment. It was tested on Sun Ultra-2 machine with SUN OS 5.5.1, X11R5 and MOTIF 1.2 environment, and runs very well.

This application implements FAST and (approximately) OPTIMAL versions of Tree Alignment, Generalized Parsimony Phylogeny Reconstruction and Generalized Tree Alignment algorithms. General Affine gap penalty functions are used in all FAST algorithms, although only gap-extension penalty is used in OPTIMAL algorithms for the sake of speed.

The sequences can be DNA, RNA or PROTEIN or others. In general, as long as the sequence characters are in the range of a--z and A--Z (not sensitive to character case), and the score matrix alphabet contains the sequence alphabet, the sequences would be accepted by this application.

Using a pointing device (usually a mouse) with pointing-and-clicking operation style can do all operations. Some operations can also be alternatively accomplished by using keyboard input.

5.1 Getting Started

If TAAR is not installed, you should first read the README file contained in TAAR application package and installs it onto a UNIX system with X Window and Motif supporting environment. Then make compilation of TAAR to get an executable file "taar". TAAR can be downloaded through Internet and can be find in web page of Computer and Software Department of McMaster University, Canada.

To get started, type: **taar** on a system prompt (or type other name that you compiled or changed to the TAAR's executing file). Figure 5.1 shows the initial screen of TAAR. If the screen is not displayed properly, you should check whether the resources file is installed appropriately.

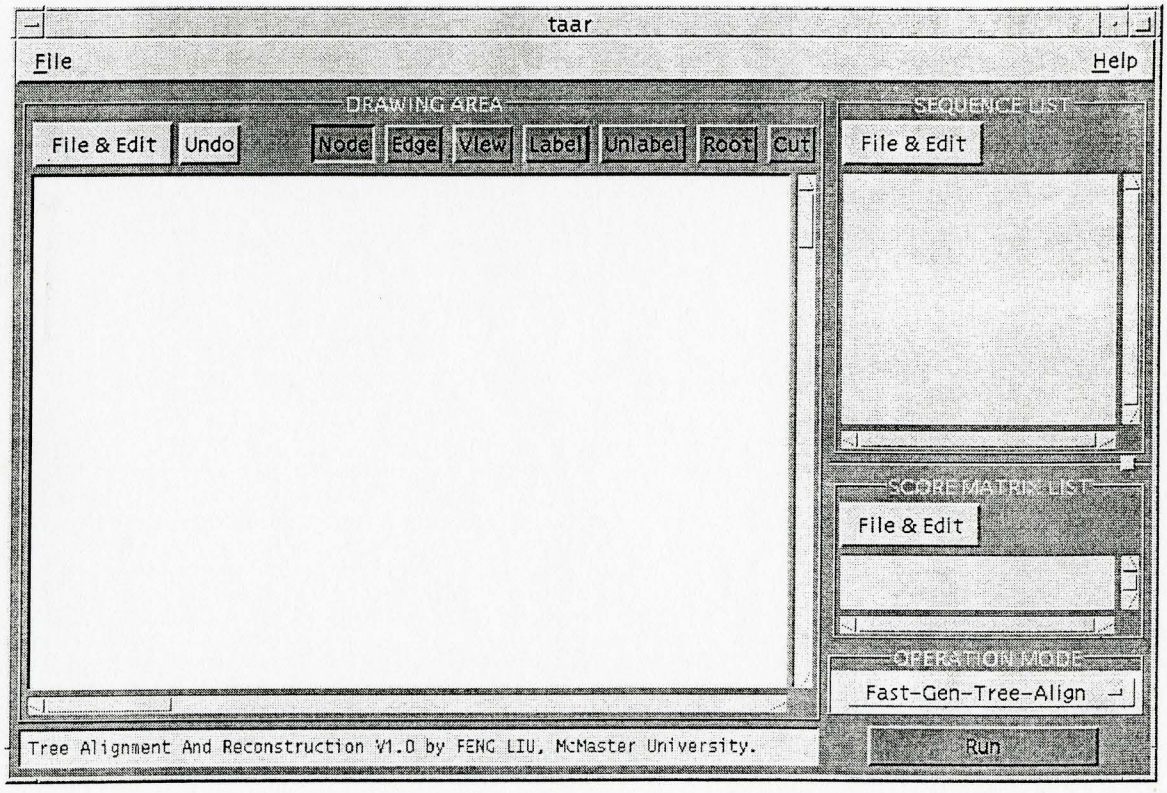


Figure 5.1: Initial screen of TAAR

5.2 Screen Display and Operation

As we discussed earlier, TAAR divides the screen into six main areas as shown in Figure 5.2. Let's discuss each area one by one in following sections and see what functions they provide respectively.

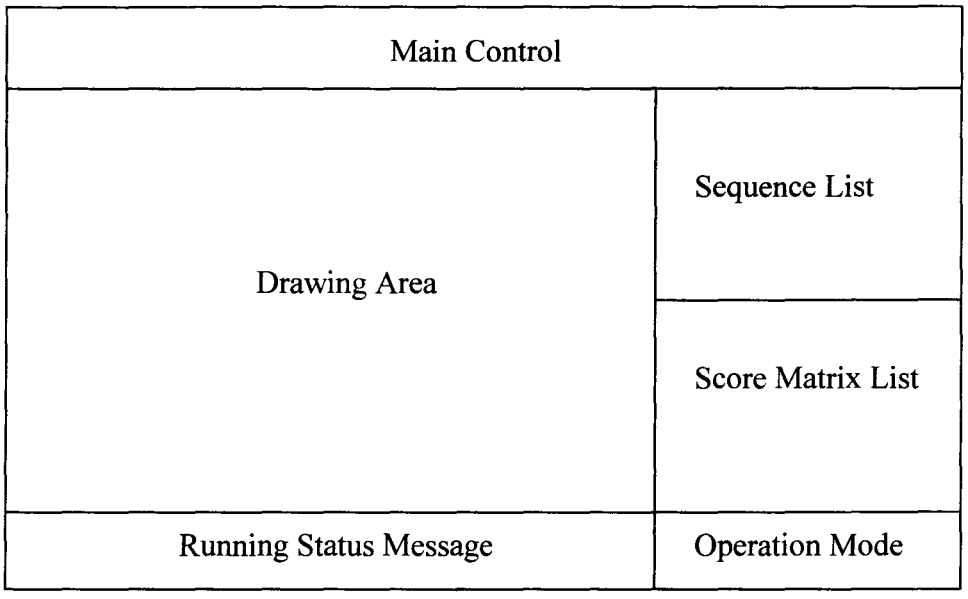


Figure 5.2: The main screen areas of TAAR

5.2.1 Main Control Panel

The Main Control Panel is a menu bar that provides general operations for whole application. There are couples of pull down menus associated with it. For example, some of these operations are *exit* and *help*. These menus can be activated by mouse click or ALT-<Key> keyboard input. The <Key> represents the underline character of a menu command word. User can see on line help or exit application from these menus.

5.2.2 Drawing Area

The Drawing Area contains a Canvas, a pull-down menu and a series drawing mode selection buttons. The pull-down menu contains all graph-related file operation commands. User can save the graph into a disk file and later open it and display it on the canvas by invoking menu items here. Those menu commands are Load, Save As, Print and Clear. Their functions are shown as following:

- **Load**: Load tree structures and their leaf sequences from a file into the drawing Canvas.

  How to operate: Click on this menu option and a File Selection Dialog window will be popped up, make a file name selection and click LOAD button to load the tree structures and their leave sequences from the file. The maximum number of tree nodes is limited by the constant MAXNODE which is defined in the file *main.h* of this application.

- **Save**: Save all tree structures and leaf sequences of the drawing Canvas into a file.

  How to operate: Click on this menu option and a File Selection Dialog window will be popped up, input a file name or make a file name selection and click SAVE button of this dialog window to save all tree structures and their leaf sequences into the file.

- **Print**: Save user selected tree image or window image into the file *taar_Tree_Image_Temp.ps* with postscript format and then print it out to the default printer of your system.

How to operate: When clicking this menu item, the cursor will change to a cross shape. Then you can click on any window to dump that whole window image to the file and print it out. You may also push down the left mouse button and move the mouse while holding down the button to select a rectangular part image and dump this image part to the file and print it out.

When first click or push the left mouse button, one beep sound will be heard. When image is stored into the file, two-beep sound will be heard. User should make sure the name *taar_Tree_Image_Temp.ps* is not used before printing. Otherwise the content of the file with this name will be written over by the new image file (i.e. old file will be lost!). User also should make sure the default printer is a postscript printer. For how to set up a default printer to a specific printer, please ask your system administrator or read UNIX system help by typing command: *man setenv*. The printing command for this menu is implemented with *lpr taar_Tree_Image_Temp.ps* command in *taar_main.c* file. User may change it by adding a ***-Pprintername*** option after *lpr* to let it send the image file to the ***printername*** printer. Here ***printername*** is the name of the postscript printer.

- **Clear**: Clear the Canvas area.

  How to operate: Click on this button and a confirmation dialog will pop up, then make your selection.

The Canvas takes up most part of this area. It is the area where graphs are composed and manipulated and relevant data are displayed. The user only uses mouse to compose a graph by pointing, clicking and dragging. Those drawing mode buttons provide various graph editing related modes. They are Node, Edge, View, Label, Unlabel, Root, and Cut. Each of them provides different functions as following:

- **Node mode**: let the user generate a new node or drag the node to move it to another position.

  How to operate: To draw a node: point and click once at any empty Canvas area to drop a node. The node will appear as a small empty circle. To move a node: point at a node and press the left mouse button, then move the pointer while holding down the button to move the node to the new position, release the button to settle the node to this new position.

- **Edge mode**: let the user draw an edge between two nodes.

  How to operate: Point at a node and press the left mouse button, move point to another node while holding down the button, then release button at the node to settle the edge between these two nodes.

- **View & Edit mode**: let the user view and edit the sequence assigned to the node or view a pair of aligned sequences between two end nodes of an edge.

  How to operate: Point and click a node once, an edit window with the node sequence will be popped up. User can view and edit the sequence on this window.

When point and click an edge once, a window with a pair of aligned sequences will be popped up.

- **Label mode**: let the user assign a sequence from the SEQUENCE LIST to a node.

  How to operate: Point and click a name once in the SEQUENCE LIST to highlight it, then point and click a node in Canvas once to assign this named sequence to the node. The node will appear as a solid node.

- **Unlabel mode**: let the user delete the sequence on a node if there is a sequence assigned to this node.

  How to operate: Point and click the node with a sequence once. The node will appear as a small empty circle.

- **Root mode**: let current graph be redrawn as a rooted tree as the clicked node being the root node and keeping existing graph structure.

  How to operate: Point and click a node once, the original graph will be redrawn as a rooted tree. This pointed node will be the root node of the rooted tree.

- **Cut mode**: let the user delete a node and its connected edges or delete an edge.

  How to operate: Point and click a node once to delete the node and its connected edges. Point and click an edge once to delete the edge.

5.2.3 Running Status Message

The Running Status Message displays current operation related messages. It may show some operation tips, or warning messages or error messages. The user can get some useful information from this message board.

5.2.4 Sequence List

The Sequence List area takes care of inputting, displaying and editing all sequences for alignment. There is another separate list window below this sequence list window. This lower window is for score matrix input. There is a small button between these two list window borders. The user can point at it and push it up or down to adjust the upper and lower window size relatively while push and hold the left mouse button.

Sequence input, edit and display is operated in SEQUENCE LIST window. The inputted sequences can only contain characters in the range of a--z, A--Z and the Space character. There is no limit on the length of a sequence. The maximum number of input sequences for an operation is limited by the constant **MaxSeqNum** which is defined in file "main.h" of this application. Here are some sample sequences with valid formats:

```
        aaatttc cCCGGGAccc AATTTgg
or      aaatttc cccgggaccc aatttgg
or      AAATTTC CCCGGGACCC AATTTGG
or      AAATTTCCCCGGGACCCAATTTGG
or      aaatttccccgggacccaattgg
```

Sequences are loaded into this application by first creating a sequence file with a valid format. This is a FASTA format [20], which is described later in this chapter, and it can include any comments in the file. This sequence file is loaded in by using File & Edit pull down menu in SEQUENCE LIST window. Here is a valid sample input sequence file:

```
#any comment goes here in one line

>name of sequence one
AAATTTCCGG CCCGGGACCC AATTTGG

#another comment here in one line

>name of another sequence
TTTAGCGGTT AAGCGGTTGC GGTCGGTATA GGCTT

>name of one more sequence
TTCGGGCCCAGGGCCAAATCGG
```

Please note that both comment and sequence name must be contained in one line (less than 255 characters before hitting the 'Return' key). A comment must start with a special character '#'. A sequence name must start with a character '>'. Each sequence name should have a sequence followed immediately. A sequence can be continued in any number of lines, and be of any length as long as its format is valid.

In this sequence list window, there is a pull-down menu that contains all sequence related operation commands. They are Load, Edit & View, Cut, Save As and Clear. You must first click on this menu button to see the following items:

- **Load:** Load sequences from a file into the SEQUENCE LIST window.

How to operate: Select 'Load' menu command and a File Selection Dialog will be popped up; make a file name selection and click LOAD button in this dialog to load the sequences from the file. The maximum number of sequences to be loaded into this window is limited by the constant **MaxSeqNum** that is defined in file "main.h" of this application.

- **Edit & View:** View and Edit a sequence.

  How to operate: Highlight a sequence name in the list by point and click this name; then select 'Edit & View' menu command to pop up a window to display the named sequence, or double click on this name to bring up a display window. This named sequence can be viewed, edited, saved or printed out in that dialog window.

- **Cut:** Cut a selected sequence out from TAAR.

  How to operate: Highlight the sequence name, which you would like to cut, in the list by pointing and clicking at it in the list; then select 'Cut' menu command to cut this named sequence out from the list. When this sequence is cut out, it still exists in the disk file but not in TAAR.

- **Save As:** Save all sequences that are listed in the SEQUENCE LIST window to a user selected file.

How to operate: Select 'Save As' menu command to pop up a File Selection Dialog; then select a file name and click SAVE button in this dialog window to save all sequences in the list into the file.

- **Clear:** Cut all sequences out from TAAR in the SEQUENCE LIST window.

  How to operate: Select 'Clear' menu command will pop up a confirmation dialog window; then make your selection. Please be noted that those cut out sequences still exist in the disk file but not in TAAR.

5.2.5 Score Matrix List

This list window takes care of inputting, editing and displaying all score matrices needed for calculations. The inputted score matrix must contain a '-' character and some other characters. These characters are case insensitive. The alphabet of the score matrix must include all letters existing in the input sequences. The maximum size of the alphabet letters of a score matrix is limited by the constant **MaxLetters** which is defined in the file "main.h" of this application. The maximum number of score matrices to be loaded into this window is limited by the constant **MaxScoreNum** which is also defined in file "main.h". The scores must be non-negative real or integer values in order that the algorithms can work. Here TAAR only takes symmetric score matrix. So A→C score is same as C→A score. The following score matrix is a sample score matrix with a valid format:

```
-       A       C       T       G
0    2.25    2.25    2.25    2.25
        0    1.75    1.75       1
             0       1    1.75
                     0    1.75
                          0
```

The representative characters for each row in this matrix have the same order as those for score matrix column. It is shown as following:

```
   - A C T G
-
A
C
T
G
```

Here '-' means gap extension penalty. In TAAR, we consider gap extension penalty for all OPTIMAL calculations, and consider both open gap penalty and gap extension penalty for all FAST calculations. Score matrix is loaded into this application by first creating a score matrix disk file with a valid format (several score matrices can be stored in one file). Then it is loaded in by using LOAD command of File & Edit menu in SCORE MATRIX LIST window. Following is a sample input score matrices file with a valid format:

```
#Any comment goes here in one line

##One score matrix name

-       A       C       T       G
0    2.25    2.25    2.25    2.25
        0    1.75    1.75       1
             0       1    1.75
                     0    1.75
                          0
```

```
#Another comment here in one line

##Another score matrix name

 -       C       T       G       A
 0       5       5       5       5
         0       5       5       5
                 0       5       5
                         0       5
                                 0
```

Please note that a comment or a score matrix name must be in one line (less than 255 characters before hitting the 'Return' key). A comment must start with ONE symbol '#'. A score matrix name must start with TWO symbols '##'. Scores should fill the upper-right half triangle, and can be continued in any number of lines.

SCORE MATRIX LIST window also contains File & Edit menu button. This menu provides similar functions as the menu provided in SEQUENCE LIST window. But here it is only for score matrices.

5.2.6 Open Gap Penalty Input

For all FAST alignments, when the user click the RUN button to do alignment calculation in Operation Mode area, a dialog window for inputting OPEN GAP penalty value will pop up. User can input an integer or decimal value for the calculation needed open gap penalty in this dialog window. If the input value is 0, then only extension gap penalty is considered for those FAST alignments.

5.2.7 Operation Mode

In this area, there are a Calculation option menu button and a RUN command button. The option menu button is for making a selection of calculation mode and the RUN command button is for starting the calculation of selected mode. After the user finish all preparation work, the user must press this button to start the selected calculation and alignment work that is done by computer. If any condition is unsatisfied for doing a specified operation, an error message will pop up during this run time and the calculation work will stop. Otherwise, the work result will be shown after a while.

After clicking the option menu button and making a selection, the selected operation name will be shown on the button face so user can easily see it. TAAR provides eight calculation modes. They are:

- **Fast-RootedTree-Align:** Fast Rooted Tree Alignment

  Input: Drawing a rooted tree in Drawing Area and assigning all sequences to be aligned from Sequence List window to all tree leaves respectively.

  Output: A multiple alignment of all tree leaf sequences with calculation time optimization.

- **Fast-Tree-Align:** Fast Tree Alignment

  Input: Drawing a non-rooted tree in Drawing Area and assigning all sequences to be aligned from Sequence List window to all tree leaves respectively.

  Output: A multiple alignment of all tree leaf sequences with calculation time optimization.

- **Fast-Phylogeny-Recons:** Fast Phylogeny Reconstruction

  Input: In Sequence List window, only loading in those considered sequences.

  Output: A Phylogeny tree with a minimum tree cost and all loading sequences assigned to the tree leaves. Calculation time optimization is in concern.

- **Fast-Gen-Tree-Align:** Fast Generalized Tree Alignment

  Input: In Sequence List window, only loading in sequences to be aligned.

  Output: A multiple alignment of all loading sequences and a Phylogeny tree with a minimum tree cost and all loading sequences assigned to all tree leaves respectively. Calculation time optimization is in concern.

- **Opt-RootedTree-Align:** Optimal Rooted Tree Alignment

  Input: Drawing a rooted tree in Drawing Area and assigning all sequences to be aligned from Sequence List window to all tree leaves respectively.

  Output: An optimal multiple alignment of all tree leaf sequences.

- **Opt-Tree-Align:** Optimal Tree Alignment

  Input: Drawing a non-rooted tree in Drawing Area and assigning all sequences to be aligned from Sequence List window to all tree leaves respectively.

  Output: An optimal multiple alignment of all tree leaf sequences.

- **Opt-Phylogeny_Recons:** Optimal Phylogeny Reconstruction

Input: In Sequence List window, only loading in considered sequences.

Output: A Phylogeny tree with an optimal tree cost and all loading sequences assigned to the tree leaves.

- **Opt-Gen-Tree-Align:** Optimal Generalized Tree Alignment

  Input: In Sequence List window, only loading in sequences to be aligned.

  Output: An optimal multiple alignment of all loading sequences and a Phylogeny tree with an optimal tree cost and all loading sequences assigned to all tree leaves respectively.

The above operations are formed by the combinations of some basic algorithms. These basic algorithms are:

- **FAST algorithm:** This algorithm is based on the tree-lifting technique for tree alignment, and involves no iterative improvement or local cost optimization. The algorithms should run very fast, and can give good estimates.

- **OPTIMAL algorithm:** For each tree, it does tree-lifting and then repeats star alignment OPT_TIMES times on all in-nodes of the tree to progressively improve the tree cost. (The constant OPT_TIMES is defined in file "main.h". User can modify it.)

- **Non-rooted Tree Alignment:** When given a non-rooted tree of degree 3 (i.e. each in-node has three incident edges) and all tree leaves are labeled with sequences, it select a virtual root at an optimal position in the tree and uses tree-lifting technique to compute a sequence for each in-node to minimize the cost of the tree. Then it induces a multiple alignment of the leaf sequences from the optimal pairwise alignments implied by the tree edges. (The virtual root needed for the tree lifting is automatically and optimally selected by this application to balance the height of the two subtrees under the virtual root.)

- **Rooted Tree Alignment:** When given a rooted binary tree (i.e. each in-node except the root has three incident edges and the root has two incident edges), whose leaves are labeled with sequences, it uses tree-lifting technique to compute a sequence for each in-node and the root to minimize the cost of the tree. Then it induces a multiple alignment of the leaf sequences from the optimal pairwise alignments implied by the tree edges. (The virtual root needed for the tree lifting is the same root node of this rooted tree. So the user can decide where to place the virtual root.)

- **Phylogeny Reconstruction:** The method is based on the generalized parsimony. When given a list of sequences, if the list has at most three sequences, it constructs a unique two or three node star tree; if the list has more than three sequences, then it constructs a non-rooted degree-3 phylogeny tree. To construct this tree, it first

constructs the star tree consisting of the first three sequences of the list. Then it adds a new leaf node labeled with the next sequence in the list to the tree by trying inserting the leaf node at each tree edge and settling down with the one that has a minimum tree alignment cost. Repeating this step until all the sequences in the list are inserted into the tree.

- **Generalized Tree Alignment:** When given a list of sequences, it does Phylogeny Reconstruction first and then Tree Alignment on this constructed phylogenetic tree. This will yield a multiple alignment as well as an underlying phylogenetic tree.

When an operation needs a rooted tree, user must select a root node among all tree nodes. Otherwise, an error message will pop up. Generally speaking, fast algorithm runs quickly and the result is a good estimate, and the optimal algorithm can give a better result but the time spending on calculation is a little longer. The time for optimal algorithm may be considerably longer especially for longer sequences to be aligned.

5.3 Displaying Sequence and Alignments

In TAAR, any sequence, which is in Canvas or Sequence List, can be individually displayed and be edited in a separate small window. Any pairwise sequence alignment associated with tree edge can be displayed in this window. The multiple sequence

alignment can also be displayed in this small window. The user is only allowed to view the alignment and are not allowed to edit the alignment. These features are provided to users for the user's convenience. The sequence window for displaying individual sequence is shown as Figure 5.3.

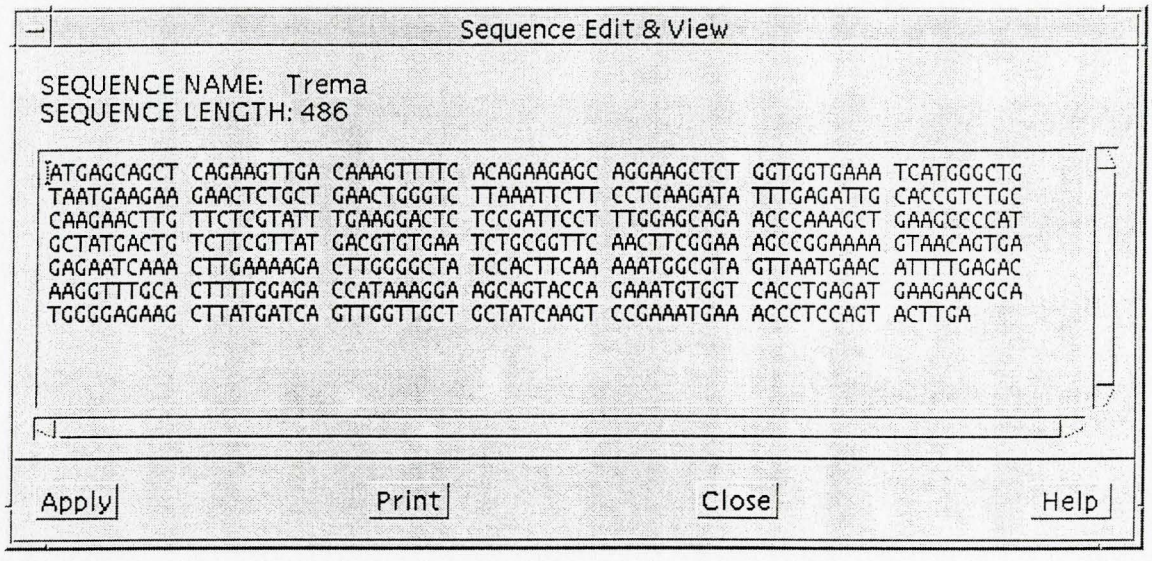


Figure 5.3: The View & Edit dialog window

In this dialog window, user can edit the sequence with any character in a--z, A--Z, symbol "-" and space. The sequence character alphabet set must be a subset of the score matrix alphabet set (This score matrix is to be used for sequence alignment calculation). If this condition is not satisfied, an error message will popup when the user click Save button to save the edition. Otherwise the sequence after edition will be saved into memory space of related data structure. This edition is not yet saved into disk file. The user may explicitly save it into disk file by click Save menu option in related File & Edit pull down menu after closing this editing window.

The window for viewing pairwise alignment is shown in Figure 5.4. In this window, user can get all alignment-related information. It includes alignment score, total gaps, sequence names and score matrix name.

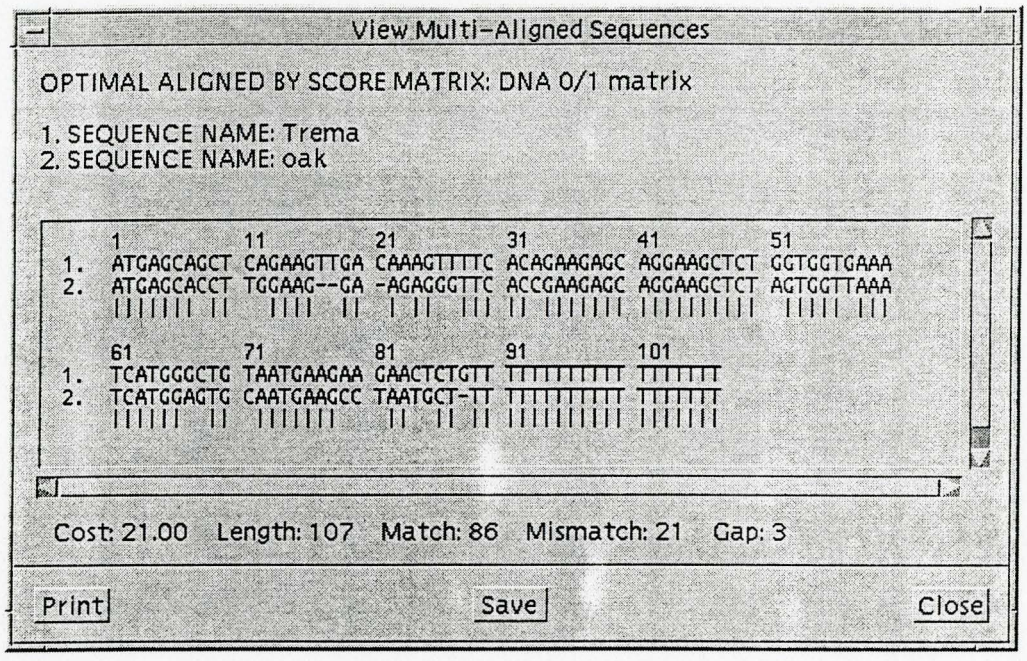


Figure 5.4: The dialog window for viewing pairwise alignment

The window for viewing multiple sequence alignment is shown in Figure 5.5. This window also shows all multiple sequence alignment related information.

5.4 File Format

In TAAR, the sequence, score matrix and tree can be written into a file and be loaded into TAAR by clicking menu commands. These features are convenient for the user to store

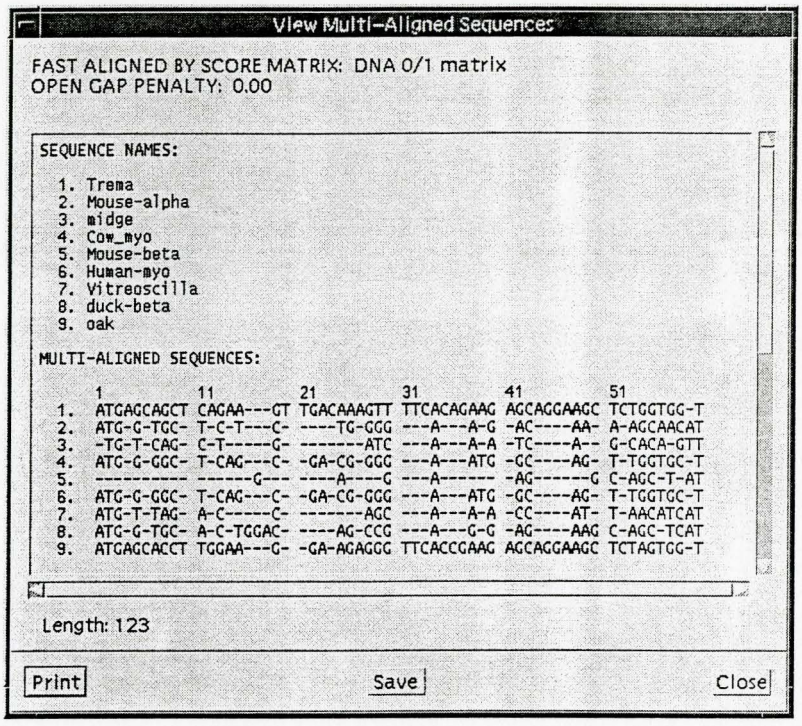


Figure 5.5: The dialog window for viewing multiple sequence alignment

various sequence, score matrix and tree files and later retrieve, use and save them easily. Thus we need to specify some file formats to help this file transferring work.

For sequence file, we adapted currently popular sequence file format called **FASTA** format [20] for TAAR usage. This format specifies as following: The sequences are delimited by an angle bracket ">" in column 1. The text immediately after the ">" is used as the sequence name and the title. Everything on the following lines until the next ">" or the end of the file is one sequence. An example of FASTA format single sequence file is

> RABSTOUT rabbit Guinness receptor

LKMHLMGHLKMGLKMGLKGMHLMHLKHMHLMTYTYTTYRRWPLWMWLPDFGHAS

ADSCVCAHGFAVCACFAHFDVCFGAVCFHAVCFAHVCFAAAVCFAVCAC

We modified the FASTA format slightly to allow any comments appearing in one line mode. The user may specify comment by starting with a symbol "#" and followed by one line comment. These comments may not exist in any line between the sequence name and the sequence itself. One of valid sample sequence file is shown and explained in **5.2.4 Sequence List** of this chapter.

The score matrix file format is similar with sequence file format. User should specify score matrix name and score values. User can also specify any comments in the file. One of valid sample score matrix file is shown and explained in **5.2.5 Score Matrix List** of this chapter.

The tree file format is customized for TAAR. User may compose this file according to its format and load it into the Canvas area of TAAR. The tree file format is also similar with above two file formats and can specify node information in it.

Chapter 6

An Example Run of TAAR

In this chapter, we present an example execution of TAAR. The testing data (sequences and score matrix) are chosen from the sample data included in the TAAR package. The input sequence file, score matrix file and output multiple sequence alignment file are presented in Appendices A, B and C respectively. Here we only demonstrate Optimal calculation modes. The Fast calculation modes are similar. The test procedures are described below.

6.1 Start the TAAR Program

To start TAAR program, type *taar* on a system prompt. Please refer to Chapter 5.1 for further instructions.

6.2 Load the Sequences

The required sequences for testing with the FASTA format can be found in TAAR directory as the file: \sample-data\Globin.seq. To load the sequences, click on File & Edit menu button in **Sequence list** frame window and then click on Load menu as follows:

File & Edit → Load

An open file dialog window will be opened. Select the correct directory and the file in this dialog window. Then click on **Open** button in this window. Now all sequences in **Globin.seq** file will be loaded into the Sequence List frame window. Please refer to Appendix A for the actual sequences.

We can view or edit individual sequence here by highlighting the sequence name and then clicking on Edit menu as follows:

File & Edit → Edit

or double click on the sequence name. The View & Edit window will pop up with the selected sequence displayed in it.

6.3 Load the Score Matrices

The required score matrix for testing is "DNA 0/1 matrix" in the file: \sample-data\SAMPLE-SCORES. To load this score matrix, we have to load this file and get all score matrices in this file. Here we should click on File & Edit menu button in **Score Matrix** list frame window and then click on Load menu as follows:

File & Edit → Load

An open file dialog window will be created. Select the correct directory and the file in this dialog window. Then click on **Open** button in this window. Now all score matrices in **SAMPLE-SCORES** file will be loaded into the Score Matrix List frame window. Please refer to Appendix B for the actual score matrices. The screen shot after loading the files is shown in Figure 6.1.

We can view or edit individual score matrix here by highlighting its name and clicking on Edit menu as follows:

File & Edit → Edit

or double click on the score matrix name. The View & Edit window will pop up with the selected score matrix shown in it.

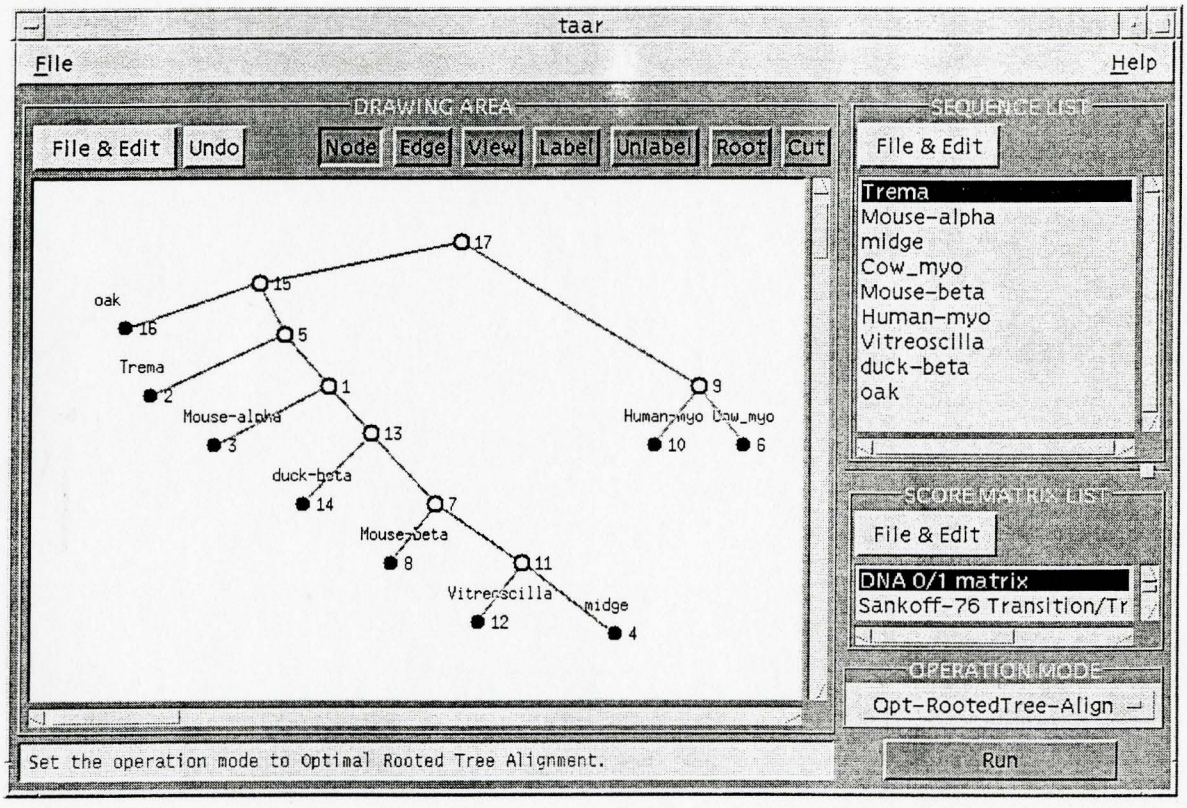


Figure 6.1: A screen shot after loading sequence and score matrix files and drawing the rooted-tree.

6.4 Calculations

6.4.1 Optimal Rooted Tree Alignment Operation Mode Test

1. Draw a rooted tree

Please refer to Chapter 5.2.2 for the usage of drawing commands. In the drawing Canvas, draw a rooted binary tree with total 9 leaves.

2. Assign sequences to the leaf nodes

Use **Label** drawing mode to assign sequences in Sequence List window to all leaf nodes respectively. You may delete the assigned sequence from the node by using **Unlabel** drawing mode. The result screen shot is shown in Figure 6.1.

3. Select a score matrix

Select the **DNA 0/1 matrix** for the calculation by click on its name in the list of Score Matrix List window.

4. Select an operation mode

Select the **Opt-RootedTree-Align** operation mode by clicking on the option button in Operation mode frame window.

5. Start calculation

Click Run button to start the calculation. Any operation progress message or error will be shown on the Running Status Message text box. The multiple aligned sequences are shown in Appendix C under Opt-RootedTree-Align section. The screen shot of the calculation result is shown in Figure 6.2 as follows.

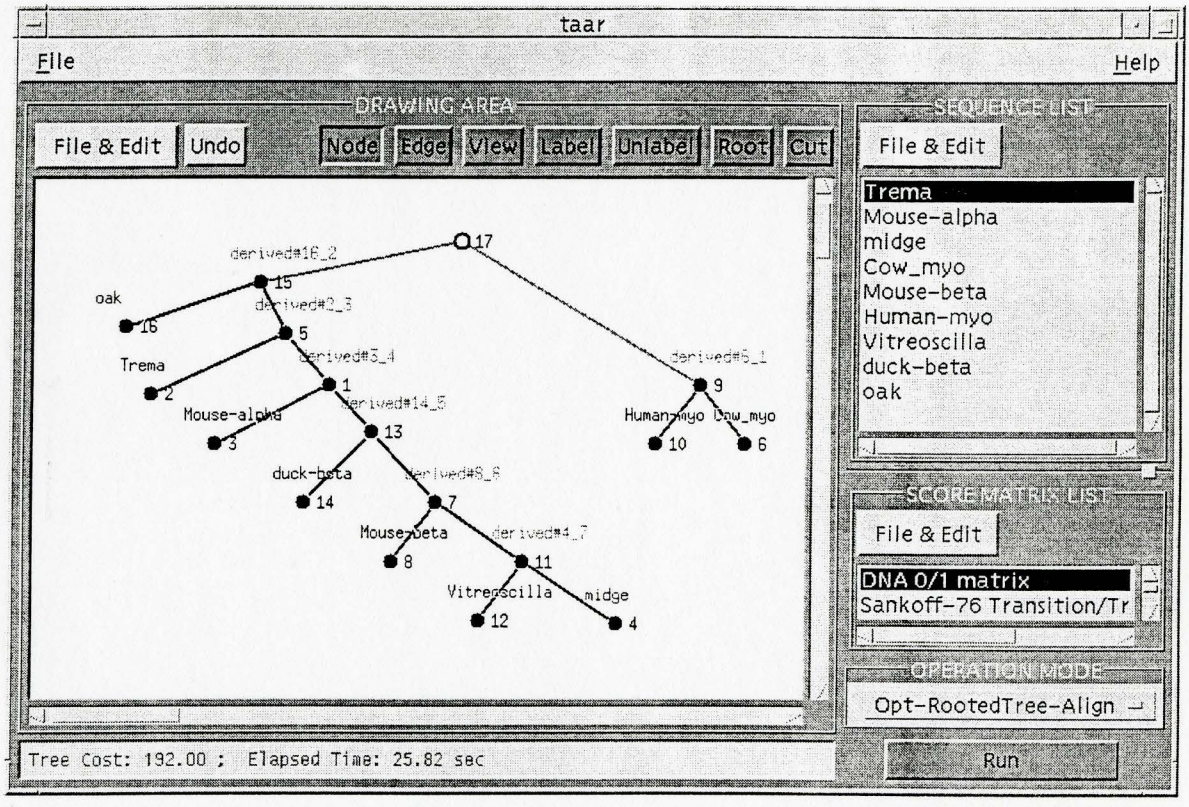


Figure 6.2: The screen shot of **Opt-RootedTree-Align** calculation result.

6.4.2 Optimal Tree Alignment Operation Mode Test

1. Draw a degree-3 tree

Please refer to Chapter 5.2.2 and 5.2.7 for the usage of drawing commands and the requirement of degree-3 tree. In drawing Canvas, draw a degree-3 tree with total 9 leaves.

2. Assign sequences to the leaf nodes

Use **Label** drawing mode to assign sequences in Sequence List window to all leaf nodes respectively. You may delete the assigned sequence from the node by using **Unlabel** drawing mode. The result screen shot is shown in Figure 6.3.

3. Select a score matrix

Select the **DNA 0/1 matrix** for the calculation by click on its name in the list of Score Matrix List window.

4. Select an operation mode

Select the **Opt-Tree-Align** operation mode by clicking on the option button in Operation mode frame window.

5. Start calculation

Click Run button to start the calculation. Any operation progress message or error will be shown on the Running Status Message text box. The multiple aligned

sequences are shown in Appendix C under Opt-Tree-Align section. The screen shot of the calculation result is shown in Figure 6.4 as follows.

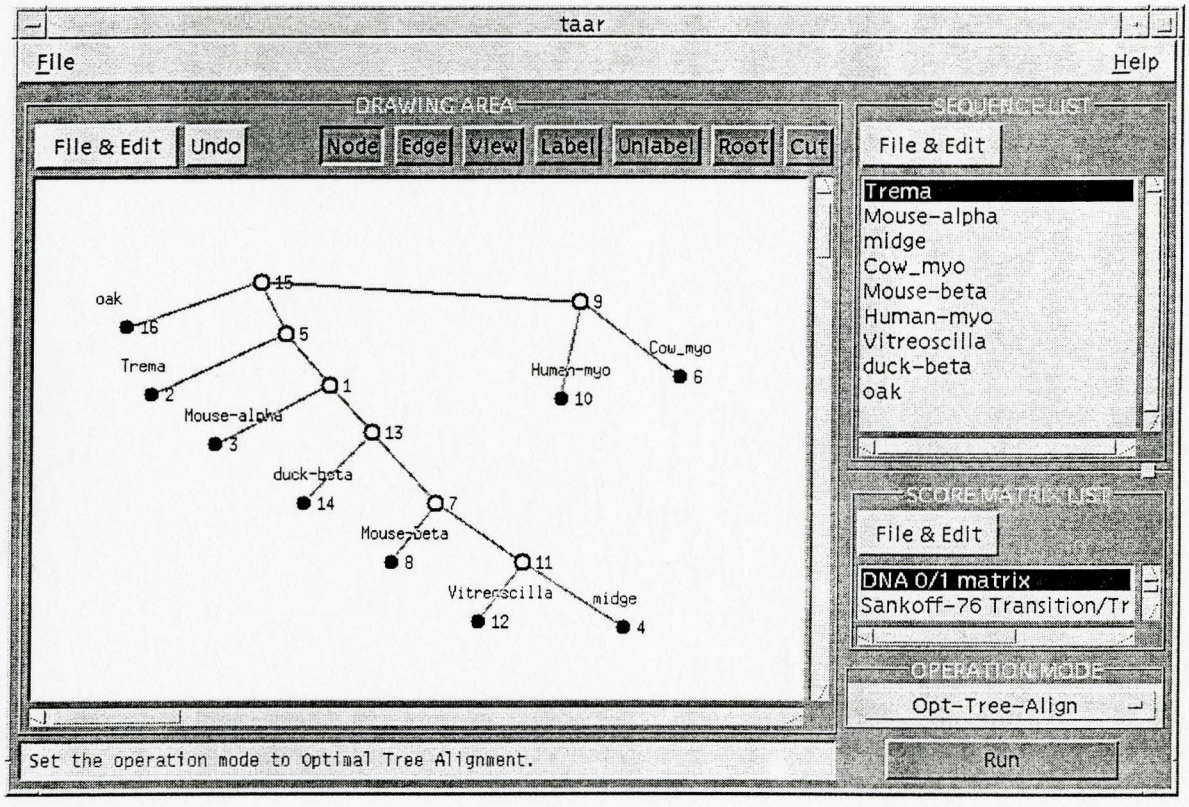


Figure 6.3: A screen shot after loading sequence and score matrix files and drawing the degree-3 tree.

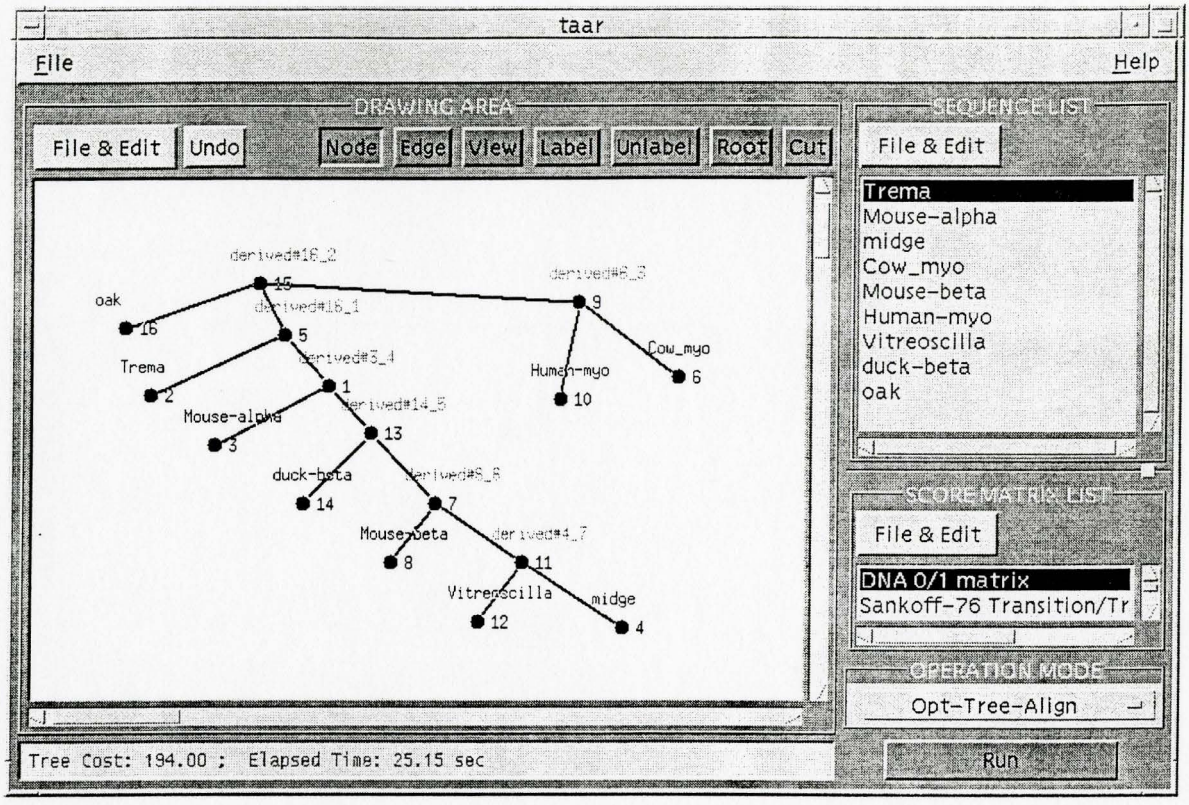


Figure 6.4: The screen shot of **Opt-Tree-Align** calculation result.

6.4.3 Optimal Generalized Tree Alignment Operation Mode Test

We would not specifically demonstrate the testing of Optimal Phylogeny Reconstruction (Opt-Phylogeny-Recons) operation mode here since it is a part of Optimal Generalized Tree Alignment calculation which is shown in the following test. For this operation, we do not need to draw a tree. The phylogeny tree will be calculated by TAAR.

1. Select a score matrix

Select the **DNA 0/1 matrix** for the calculation by click on its name in the list of Score Matrix List window.

2. Select an operation mode

Select the **Opt-Gen-Tree-Align** operation mode by clicking on the option button in Operation mode frame window.

3. Start calculation

Click Run button to start the calculation. Any operation progress message or error will be shown on the Running Status Message text box. The multiple aligned sequences are shown in Appendix C under Opt-Gen-Tree-Align section. The screen shot of the multiple sequence alignment calculation result is shown first after the operation is done. It is shown here in Figure 6.5 as follows. The constructed phylogenetic tree shown in the Canvas area of TAAR is also shown here in Figure 6.6.

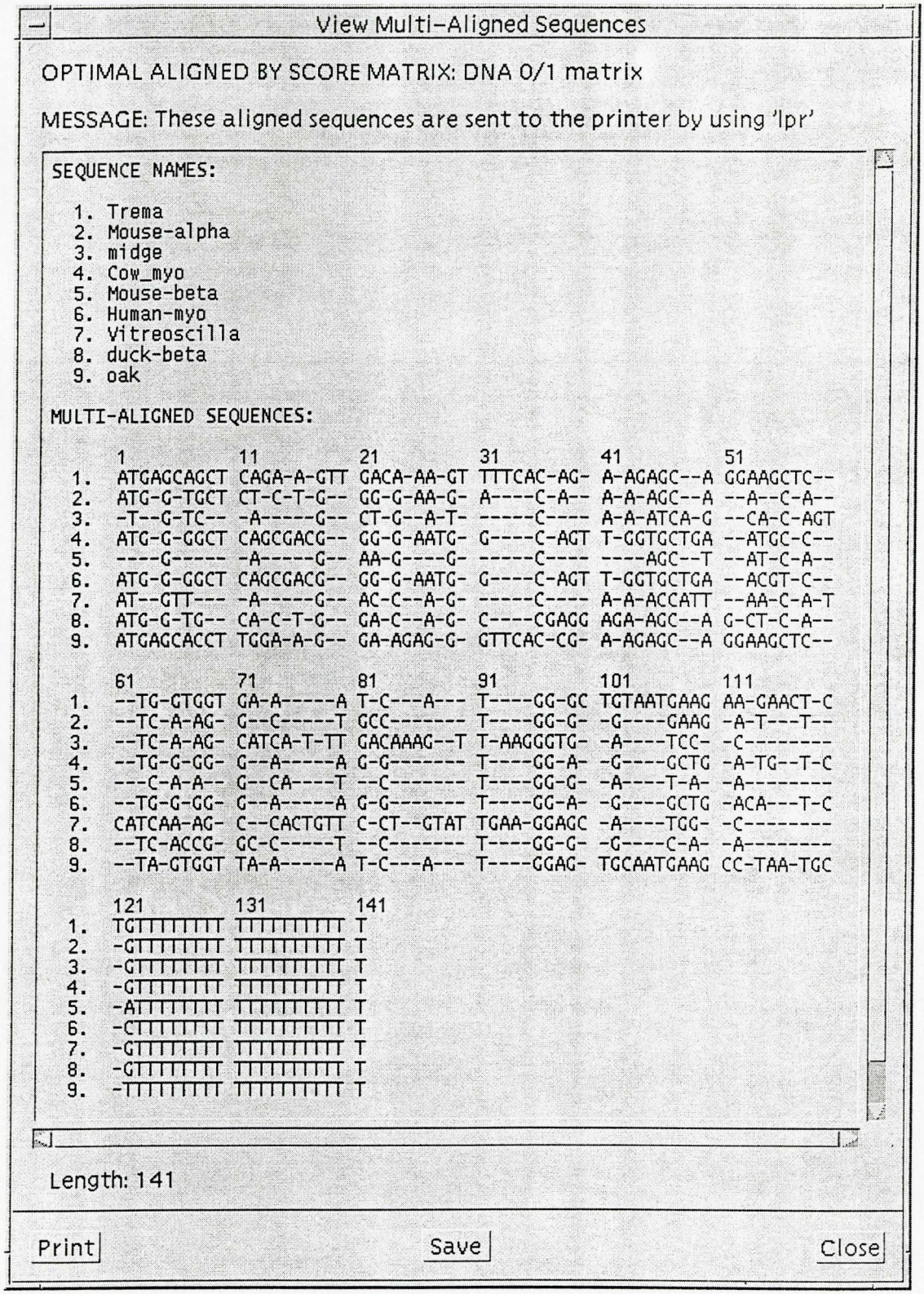


Figure 6.5: The screen shot of **Opt-Gen-Tree-Align** calculation result.

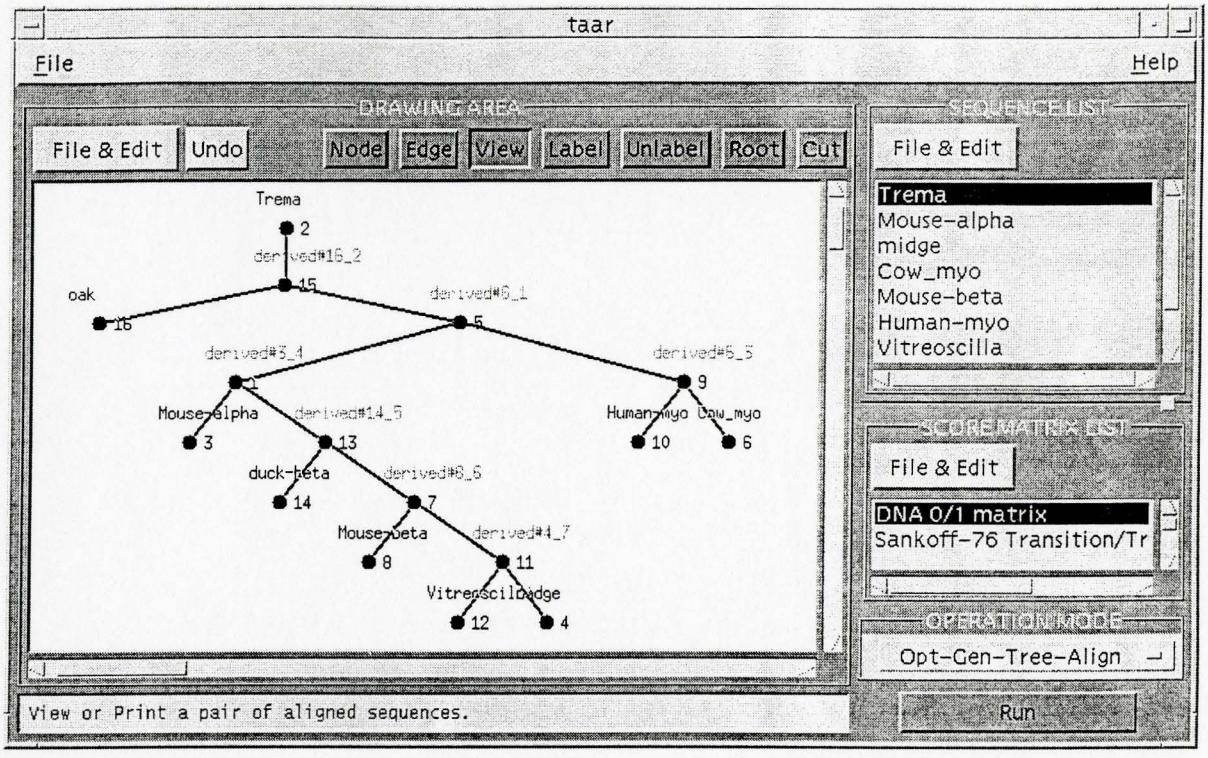


Figure 6.6: The screen shot of **Opt-Gen-Tree-Align** calculation result of Phylogeny Reconstruction.

6.5 Real Data Test

We also did some tests with real biology molecular data. The test results of phylogenetic tree and multiple sequence alignment are shown in Appendix D. TAAR was able to find the correct tree for 5S Ribosomal RAN sequences [23] and produce an almost correct tree for the GLOBIN sequences which differs from the correct one by one nearest neighbor interchange operation. The multiple sequence alignment output by TAAR also greatly resembles the original multiple sequence alignment.

Chapter 7

Concluding Remarks

The design and development of Tree Alignment and Reconstruction (TAAR) application is based on the idea of Sequence Analysis Tool (SAT) [3] and is to add more sequence alignment algorithms and to satisfy the user's advanced requirements. TAAR provides an interactive method for the sequence analysis performance of various alignment-related algorithms such as pairwise and 3-star alignment, phylogeny reconstruction and generalized tree alignment with or without a given tree structure. In fact, TAAR provides much more reliable calculation processing, more calculation functions, less data limiting requirements, and easier understanding and operation of user interface.

For instance, in SAT, the requirement of a sequence length is a predetermined constant. Users have to decide it before compile the application program. But TAAR does not need to predetermine the sequence length. The data structures for storing sequence are dynamically created to accommodate any length of sequences. Here the sequence length limit only depends on the size of the user's computer system memory space. As another example, in SAT, the three buttons of a mouse pointer device have different operation functions. It is difficult for user to remember those functions and operating methods. The operation of SAT also depends on the specific pointer device very much (i.e., we have to use a three button pointing device). Now in TAAR, the user only needs to point and press the left most button of a pointing device to do any operation. Thus it is easier for the user to operate since the operation is not dependent on

any specific pointing device. In some operation mode, the user may use the keyboard to do the operation as well as a pointing device. In yet another instance, SAT does not provide reliable multiple sequence alignment and it is easy to crash. TAAR provides reliable multiple sequence alignment and can shows the alignment result in an easily understandable format. The user can choose to print the result out by a printer or save it into a file.

In future development, I would like to say that we should transport this application to different computer platforms so more users can use it. Right now it can only be used on UNIX and X Window System platforms. Since the Internet is becoming more popular and JAVA is becoming more robust on various platforms, TAAR should be transformed into JAVA and implemented on the Internet. Another direction is that presently PC development is very fast and the CPU speed of a PC grows quickly. If the user's demand for this sequence analysis application on Microsoft Windows platform is high, then a quick solution is to use MS Visual C++ or Visual Basic programming language to transform TAAR into a MS Windows platform based application. Since most parts of TAAR programs are written in C, so this transformation should be relatively easy.

Appendix A

Sample Input Sequence File

#The short version Globin sequences for testing purpose are shown here.

```
#Length: 207
>Trema

ATGAGCAGCT CAGAAGTTGA CAAAGTTTTC ACAGAAGAGC AGGAAGCTCT
GGTGGTGAAA TCATGGGCTG TAATGAAGAA GAACTCTGTT TTTTTTTTTT
TTTTTTT


#Length: 74
>Mouse-alpha

ATGGTGCTCT CTGGGGAAGA CAAAAGCAAC ATCAAGGCTG CCTGGGGGAA
GATTGTTTTT TTTTTTTTTT TTTT


#Length: 79
>midge

TGTCAGCTGA TCAAATCAGC ACAGTTCAAG CATCATTTGA CAAAGTTAAG
GGTGATCCCG TTTTTTTTTT TTTTTTTTT


#Length: 86
>Cow_myo

ATGGGGCTCA GCGACGGGGA ATGGCAGTTG GTGCTGAATG CCTGGGGGAA
GGTGGAGGCT GATGTCGTTT TTTTTTTTTT TTTTTT


#Length: 52
>Mouse-beta

GAGAAGGCAG CTATCACAAG CATCTGGGAT AAATTTTTTT TTTTTTTTTT TT
```

#Length: 86
>Human-myo

ATGGGGCTCA GCGACGGGGA ATGGCAGTTG GTGCTGAACG TCTGGGGGAA
GGTGGAGGCT GACATCCTTT TTTTTTTTTT TTTTTT

#Length: 83
>Vitreoscilla

ATGTTAGACC AGCAAACCAT TAACATCATC AAAGCCACTG TTCCTGTATT
GAAGGAGCAT GGCGTTTTTT TTTTTTTTTT TTT

#Length: 73
>duck-beta

ATGGTGCACT GGACAGCCGA GGAGAAGCAG CTCATCACCG GCCTCTGGGG
CAAGTTTTTT TTTTTTTTTT TTT

#Length: 103
>oak

ATGAGCACCT TGGAAGGAAG AGGGTTCACC GAAGAGCAGG AAGCTCTAGT
GGTTAAATCA TGGAGTGCAA TGAAGCCTAA TGCTTTTTTT TTTTTTTTTT TTT

Appendix B

Sample Input Score Matrix File

Here are some sample cost matrices. User can use them to do
some testing job. Note that the open gap penalty will be input online.

```
##DNA 0/1 matrix

  G      A      C      T      -

  0      1      1      1      1
         0      1      1      1
                0      1      1
                       0      1
                              0


##Sankoff-76 Transition/Transversion matrix

  -       A       C       T       G

  0    2.25    2.25    2.25    2.25
          0    1.75    1.75       1
                  0       1    1.75
                          0    1.75
                               0


##DNA PAM 20 Transition/Transversion matrix

  A      C      G      T      -

  0     34     23     34     34
         0     34     23     34
                0     34     34
                       0     34
                              0
```

##DNA PAM 85 Transition/Transversion matrix

A	C	G	T	-
0	2	1	2	2
	0	2	1	2
		0	2	2
			0	2
				0

##Dayhoff PAM 120 matrix

A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	-
10	16	13	13	16	14	13	12	16	14	16	15	15	17	12	12	12	20	17	13	14
	7	14	16	17	12	16	17	12	15	17	11	14	17	14	14	15	12	19	16	14
		9	11	18	13	12	13	11	15	17	12	16	17	15	12	13	18	15	16	14
			8	20	12	10	13	13	16	18	14	17	20	15	13	14	21	18	16	14
				4	20	20	18	17	16	20	20	19	19	16	14	16	21	14	15	14
					7	11	16	10	16	15	13	14	19	13	15	15	19	18	16	14
						8	14	14	16	17	14	17	19	14	14	15	21	17	16	14
							8	17	17	18	16	17	18	15	12	14	21	19	15	14
								6	17	16	15	17	15	14	15	16	18	14	16	14
									7	12	15	12	13	16	15	13	20	15	10	14
										8	17	10	13	16	17	16	18	16	12	14
											8	13	19	15	14	14	18	19	17	14
												5	14	16	15	14	20	17	12	14
													5	18	16	17	14	9	16	14
														7	12	14	20	19	15	14
															10	11	15	16	15	14
																9	19	16	13	14
																	1	14	21	14
																		5	16	14
																			8	14
																				14

Appendix C

Test Output of Multiple Sequence Alignment

1. Multiple sequence alignment output of Opt-RootedTree-Align operation mode

```
OPTIMAL ALIGNED BY SCORE MATRIX: DNA 0/1 matrix

SEQUENCE LENGTH: 146

SEQUENCE NAMES:

  1. Trema
  2. Mouse-alpha
  3. midge
  4. Cow_myo
  5. Mouse-beta
  6. Human-myo
  7. Vitreoscilla
  8. duck-beta
  9. oak

MULTI-ALIGNED SEQUENCES:

     1          11         21         31         41         51
  1. ATGAGCAGCT CAGAAGTTG- ACAAAGTTTT CA-C-AG-A- AGAGC---AG G--AA-GC-T
  2. ATG-G-TGCT CT-CTG--G- G-GAA--G-- -A-C-A--A- A-AGC---A- A--CA----T
  3. -T--G-TC-- -A---G--C- T-G-A--T-- ---C----A- A-ATCA-GC- A--CAGT--T
  4. ATG-G-GGCT CAG--C--G- A--CGG-G-- ------G--- A-A-----TG G--CA-G--T
  5. ----G----- -A---G--A- A-G----G-- ---C------ --AGC--TA- T--CA-----
  6. ATG-G-GGCT CAG--C--G- A--CGG-G-- ------G--- A-A-----TG G--CA-G--T
  7. AT--GTT--- -A---G--A- C-C-A--G-- ---C----A- A-ACCATTA- A--CA-TCAT
  8. ATG-G-TG-- CA-CTG--G- A-C-A--G-- C--CGAGGAG A-AGC---A- GCTCA----T
  9. ATGAGCACCT TGGAAG--GA A-GAGG-GTT CACC--G-A- AGAGC---AG G--AA-GC-T

     61         71         81         91         101        111
  1. C-T-GGTG-- G-----TGAA -ATCA----- --T----GG- GCTGTAA--T GAA-GAAGAA
  2. C-A-AG-G-- C-----TG-- ---CC----- --T----GG- G--G------ GAA-G-AT--
  3. C-A-AG-CAT CA-T-TTGA- ----CAAAG- -TT-AAGGGT G--A------ TCC---C---
  4. --T-GGTG-- C-----TGAA TGCC------ --T----GG- G-GG-AAGGT GGAGG-CTGA
  5. C-A-A--G-- CA----T--- ----C----- --T----GG- G--A------ T-A---A---
  6. --T-GGTG-- C-----TGAA CGTC------ --T----GG- G-GG-AAGGT GGAGG-CTGA
  7. CAA-AG-C-- CACTGTTC-- ----CT--GT ATTGAA-GGA GC-A------ TGG---C---
  8. C-ACCG-GC- C-----T--- ----C----- --T----GG- G--G------ C-A---A---
  9. C-T-AGTG-- G-----TTAA -ATCA----- --T----GGA G-TGCAA--T GAAGC-CTAA
```

```
     121         131         141
  1. CT--CTGTTT TTTTTTTTTT TTTTTT
  2. -T----GTTT TTTTTTTTTT TTTTTT
  3. ------GTTT TTTTTTTTTT TTTTTT
  4. -TGTC-GTTT TTTTTTTTTT TTTTTT
  5. ------ATTT TTTTTTTTTT TTTTTT
  6. -CATC-CTTT TTTTTTTTTT TTTTTT
  7. ------GTTT TTTTTTTTTT TTTTTT
  8. ------GTTT TTTTTTTTTT TTTTTT
  9. -TG-C-TTTT TTTTTTTTTT TTTTTT
```

2. Multiple sequence alignment output of Opt-Tree-Align operation mode

OPTIMAL ALIGNED BY SCORE MATRIX: DNA 0/1 matrix

SEQUENCE LENGTH: 151

SEQUENCE NAMES:

1. Trema
2. Mouse-alpha
3. midge
4. Cow_myo
5. Mouse-beta
6. Human-myo
7. Vitreoscilla
8. duck-beta
9. oak

MULTI-ALIGNED SEQUENCES:

```
     1           11          21          31          41          51
  1. AT-G-AGCAG CT-CAGAAGT TGACAAA--G TTTTCA-C-A G-A-AGAGC- --AGG--AA-
  2. ATGG-TGCT- CTCTGG--G- -G---AA--G -----A-C-A --A-A-AGC- --A-A--CA-
  3. -T-G-TC-A- ----GC-T-- -G----A--T -------C-- --A-A-ATCA -GC-A--CAG
  4. AT-G--G--- ----GG--C- -TC-AGC--G -----A-C-G --G-GGA-A- --TGG--CA-
  5. ---G----A- ----GA-A-- -G-------G -------C-- ------AGC- -TA-T--CA-
  6. AT-G--G--- ----GG--C- -TC-AGC--G -----A-C-G --G-GGA-A- --TGG--CA-
  7. AT-GTT--A- ----GA-C-- -C----A--G -------C-- --A-A-ACCA TTA-A--CA-
  8. ATGG-TGCA- C--TGG-A-- -C----A--G ----C--CGA GGAGA-AGC- --A-GCTCA-
  9. AT-G-AGCAC CT-TGGAAG- -GA-AGAGGG --TTCACC-G --A-AGAGC- --AGG--AA-
```

```
    61          71          81          91          101         111
1.  GC-TC-T-GG TG--G----- TGAA-ATCA- ------T--- -GG-GCTGTA A--TGAAGAA
2.  ---TC-A-AG -G--C----- TG-----CC- ------T--- -GG-G--G-- ----GAA---
3.  T--TC-A-AG -CATCA-T-T TGA-----CA AAG--TT-AA GGGTG--A-- ----TCC---
4.  G--T--T-GG TG--C----- TGAATGCC-- ------T--- -GG-G-GG-A AGGTGGAG--
5.  ----C-A-A- -G--CA---- T-------C- ------T--- -GG-G--A-- ----T-A---
6.  G--T--T-GG TG--C----- TGAACGTC-- ------T--- -GG-G-GG-A AGGTGGAG--
7.  TCATCAA-AG -C--CACTGT TC------CT --GTATTGAA -GGAGC-A-- ----TGG---
8.  ---TC-ACCG -GC-C----- T-------C- ------T--- -GG-G--G-- ----C-A---
9.  GC-TC-T-AG TG--G----- TTAA-ATCA- ------T--- -GGAG-TGCA A--TGAAG--

    121         131         141         151
1.  G--AACT-CT -GTTTTTTTT TTTTTTTTTT T
2.  G---A-T--T -GTTTTTTTT TTTTTTTTTT T
3.  ----C----- -GTTTTTTTT TTTTTTTTTT T
4.  GCTGA-T-GT CGTTTTTTTT TTTTTTTTTT T
5.  ----A----- -ATTTTTTTT TTTTTTTTTT T
6.  GCTGA-C-AT CCTTTTTTTT TTTTTTTTTT T
7.  ----C----- -GTTTTTTTT TTTTTTTTTT T
8.  ----A----- -GTTTTTTTT TTTTTTTTTT T
9.  CCTAA-TGCT --TTTTTTTT TTTTTTTTTT T
```

3. Multiple sequence alignment output of Opt-Gen-Tree-Align operation mode

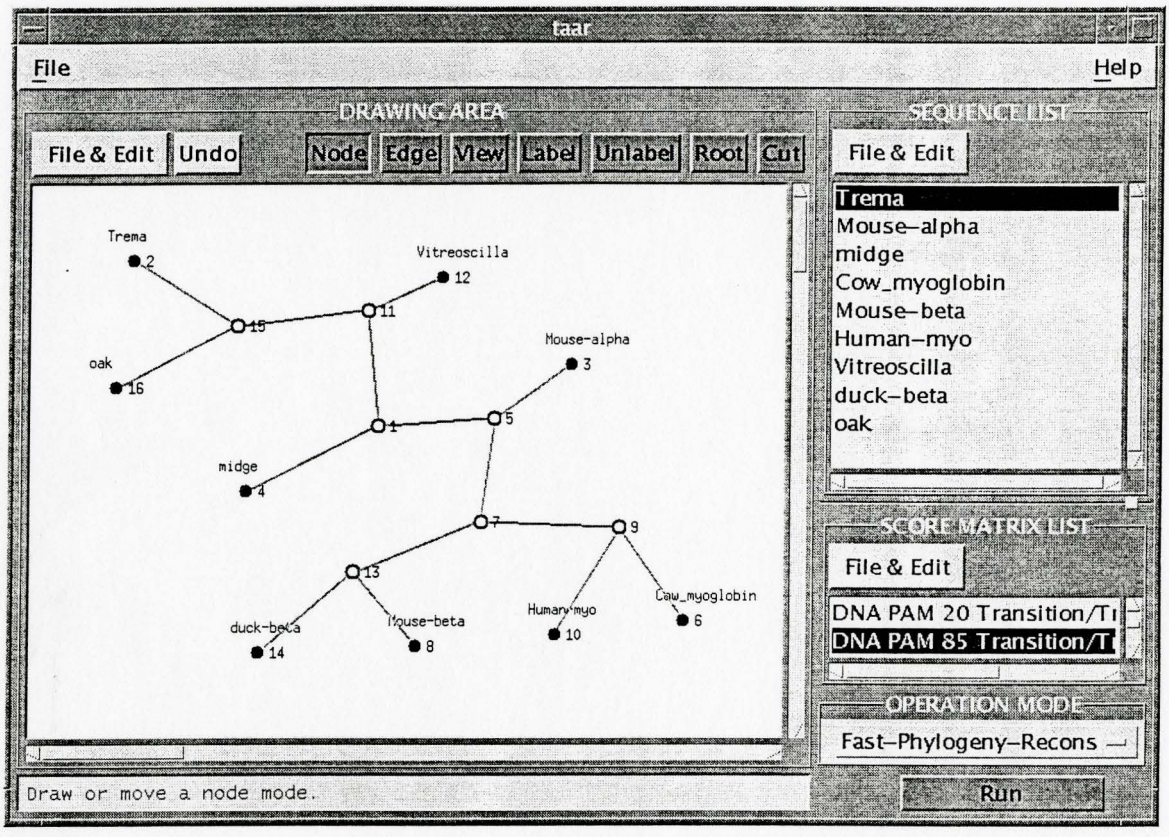
```
OPTIMAL ALIGNED BY SCORE MATRIX: DNA 0/1 matrix

SEQUENCE LENGTH: 141

SEQUENCE NAMES:

  1. Trema
  2. Mouse-alpha
  3. midge
  4. Cow_myo
  5. Mouse-beta
  6. Human-myo
  7. Vitreoscilla
  8. duck-beta
  9. oak

MULTI-ALIGNED SEQUENCES:
```

```
    1           11          21          31          41          51
1.  ATGAGCAGCT CAGA-A-GTT GACA-AA-GT TTTCAC-AG- A-AGAGC--A GGAAGCTC--
2.  ATG-G-TGCT CT-C-T-G-- GG-G-AA-G- A----C-A-- A-A-AGC--A --A--C-A--
3.  -T--G-TC-- -A-----G-- CT-G--A-T- -----C---- A-A-ATCA-G --CA-C-AGT
4.  ATG-G-GGCT CAGCGACG-- GG-G-AATG- G----C-AGT T-GGTGCTGA --ATGC-C--
5.  ----G----- -A-----G-- AA-G----G- -----C---- ----AGC--T --AT-C-A--
6.  ATG-G-GGCT CAGCGACG-- GG-G-AATG- G----C-AGT T-GGTGCTGA --ACGT-C--
7.  AT--GTT--- -A-----G-- AC-C--A-G- -----C---- A-A-ACCATT --AA-C-A-T
8.  ATG-G-TG-- CA-C-T-G-- GA-C--A-G- C----CGAGG AGA-AGC--A G-CT-C-A--
9.  ATGAGCACCT TGGA-A-G-- GA-AGAG-G- GTTCAC-CG- A-AGAGC--A GGAAGCTC--

    61          71          81          91          101         111
1.  --TG-GTGGT GA-A-----A T-C---A--- T----GG-GC TGTAATGAAG AA-GAACT-C
2.  --TC-A-AG- G--C-----T GCC------- T----GG-G- -G----GAAG -A-T---T--
3.  --TC-A-AG- CATCA-T-TT GACAAAG--T T-AAGGGTG- -A----TCC- -C--------
4.  --TG-G-GG- G--A-----A G-G------- T----GG-A- -G----GCTG -A-TG--T-C
5.  ---C-A-A-- G--CA----T --C------- T----GG-G- -A----T-A- -A--------
6.  --TG-G-GG- G--A-----A G-G------- T----GG-A- -G----GCTG -ACA---T-C
7.  CATCAA-AG- C--CACTGTT C-CT--GTAT TGAA-GGAGC -A----TGG- -C--------
8.  --TC-ACCG- GC-C-----T --C------- T----GG-G- -G----C-A- -A--------
9.  --TA-GTGGT TA-A-----A T-C---A--- T----GGAG- TGCAATGAAG CC-TAA-TGC

    121         131         141
1.  TGTTTTTTTT TTTTTTTTTT T
2.  -GTTTTTTTT TTTTTTTTTT T
3.  -GTTTTTTTT TTTTTTTTTT T
4.  -GTTTTTTTT TTTTTTTTTT T
5.  -ATTTTTTTT TTTTTTTTTT T
6.  -CTTTTTTTT TTTTTTTTTT T
7.  -GTTTTTTTT TTTTTTTTTT T
8.  -GTTTTTTTT TTTTTTTTTT T
9.  -TTTTTTTTT TTTTTTTTTT T
```

Appendix D

Real Data Test Result

1. GLOBIN sequences and the reconstructed phylogeny from them by Fast-Phylogeny-Recons operation with DNA PAM 85 Transition score matrix




```
>Trema

ATGAGCAGCT CAGAAGTTGA CAAAGTTTTC ACAGAAGAGC AGGAAGCTCT GGTGGTGAAA
TCATGGGCTG TAATGAAGAA GAACTCTGCT GAACTGGGTC TTAAATTCTT CCTCAAGATA
TTTGAGATTG CACCGTCTGC CAAGAACTTG TTCTCGTATT TGAAGGACTC TCCGATTCCT
TTGGAGCAGA ACCCAAAGCT GAAGCCCCAT GCTATGACTG TCTTCGTTAT GACGTGTGAA
TCTGCGGTTC AACTTCGGAA AGCCGGAAAA GTAACAGTGA GAGAATCAAA CTTGAAAAGA
CTTGGGGCTA TCCACTTCAA AAATGGCGTA GTTAATGAAC ATTTTGAGAC AAGGTTTGCA
CTTTTGGAGA CCATAAAGGA AGCAGTACCA GAAATGTGGT CACCTGAGAT GAAGAACGCA
TGGGGAGAAG CTTATGATCA GTTGGTTGCT GCTATCAAGT CCGAAATGAA ACCCTCCAGT
ACTTGA
```

>Mouse-alpha

ATGGTGCTCT CTGGGGAAGA CAAAAGCAAC ATCAAGGCTG CCTGGGGGAA GATTGGTGGC
CATGGTGCTG AATATGGAGC TGAAGCCCTG GAAAGGATGT TTGCTAGCTT CCCCACCACC
AAGACCTACT TCCCTCACTT TGATGTAAGC CACGGCTCTG CCCAGGTCAA GGGTCACGGC
AAGAAGGTCG CCGATGCTCT GGCCAATGCT GCAGGCCACC TCGATGACCT GCCCGGTGCC
CTGTCTGCTC TGAGCGACCT GCATGCCCAC AAGCTGCGTG TGGATCCCGT CAACTTCAAG
CTCCTGAGCC ACTGCCTGCT GGTGACCTTG GCTAGCCACC ACCCTGCCGA TTTCACCCCC
GCGGTAC

>midge

TGTCAGCTGA TCAAATCAGC ACAGTTCAAG CATCATTTGA CAAAGTTAAG GGTGATCCCG
TTGGTATCCT ATATGCTGTC TTCAAGGCTG ATCCATCAAT CATGGCTAAA TTCACACAAT
TCGCTGGAAA GGACCTCGAA TCAATCAAGG GAACAGCTCC ATTTGAAACT CATGCCAATA
GAATTGTCGG ATTCTTCTCA AAGATCATTG GTGAACTTCC AAACATTGAT GGAGATGTCA
ATACATTCGT TGCCTCACAT AAGCCCCGTG GAGTTACACA TGATCAATTA AACAACTTCC
GTGCTGGATT CGTCAGCTAC ATGAAGGCTC ACACTGACTT CGCTGGAGCT GAAGCAGCCT
GGGGTGCAAC TCTTGACACT TTCTTCGGAA TGATCTTCTC AAAGATGTAA

>Cow_myoglobin

ATGGGGCTCA GCGACGGGGA ATGGCAGTTG GTGCTGAATG CCTGGGGGAA GGTGGAGGCT
GATGTCGCAG GCCATGGGCA GGAGGTCCTC ATCAGGCTCT TCACAGGTCA TCCCGAGACC
CTGGAGAAAT TTGACAAGTT CAAGCACCTG AAGACAGAGG CTGAGATGAA GGCCTCCGAG
GACCTGAAGA AGCATGGCAA CACGGTGCTC ACGGCCCTGG GGGGTATCCT GAAGAAAAAG
GGTCACCATG AGGCAGAGGT GAAGCACCTG GCCGAGTCAC ATGCCAACAA GCACAAGATC
CCTGTCAAGT ACCTGGAGTT CATCTCGGAC GCCATCATCC ATGTTCTACA TGCCAAGCAT
CCTTCAGACT TCGGTGCTGA TGCCCAGGCT GCCATGAGCA AGGCCCTGGA ACTGTTCCGG
AATGACATGG CTGCCCAGTA CAAGGTGCTG GGCTT

>Mouse-beta

GAGAAGGCAG CTATCACAAG CATCTGGGAT AAAGTGGACT TGGAAAAAGT TGGAGGAGAA
ACTCTGGGAA GGCTCCTGAT TGTTTACCCA TGGACTCAGA GATTCTTTGA CAAGTTTGGA
AACCTCTCTT CTGCCCTGGC CATCATGGGA AACCCCCGGA TTAGAGCCCA TGGCAAGAAA
GTGCTGACAT CCTTGGGCTT GGGGGTTAAG AACATGGACA ACCTCAAGGA GACCATTGCT
CATCTCAGTG AGCTGCACTG TGACAAGCTT CATGTGGATC CTGAGAACTT CAAGCTCCTG
GGCAACATGT TGGTGATTGT CCTTTCTACT CATTTTGCCA AGGAATTCAC CCCAGAGGTG
CAGGCTGCCT GGCAGAAGCT GGTGATTGGA GTGGCCAATG CTCTGTCCCA CAAGTACCAT
TAA

>Human-myo

ATGGGGCTCA GCGACGGGGA ATGGCAGTTG GTGCTGAACG TCTGGGGGAA GGTGGAGGCT
GACATCCCAG GCCATGGGCA GGAAGTCCTC ATCAGGCTCT TTAAGGGTCA CCCAGAGACT
CTGGAGAAGT TTGACAAGTT CAAGCACCTG AAGTCAGAGG ACGAGATGAA GGCATCTGAG
GACTTAAAGA AGCATGGTGC CACTGTGCTC ACCGCCCTGG GTGGCATCCT TAAGAAGAAG

GGGCATCATG AGGCAGAGAT TAAGCCCCTG GCACAGTCGC ATGCCACCAA GCACAAGATC
CCCGTGAAGT ACCTGGAGTT CATCTCGGAA TGCATCATCC AGGTTCTGCA GAGCAAGCAT
CCCGGGGACT TTGGTGCTGA TGCCGAGGGG GCCATGAACA AGGCCCTGGA GCTGTTCCGG
AAGGACATGG CCTCCAACTA CAAGGAGCTG GGCTTCCAGG GCTAG

>Vitreoscilla

ATGTTAGACC AGCAAACCAT TAACATCATC AAAGCCACTG TTCCTGTATT GAAGGAGCAT
GGCGTTACCA TTACCACGAC TTTTTATAAA AACTTGTTTG CCAAACACCC TGAAGTACGT
CCTTTGTTTG ATATGGGTCG CCAAGAATCT TTGGAGCAGC CTAAGGCTTT GGCGATGACG
GTATTGGCGG CAGCGCAAAA CATTGAAAAT TTGCCAGCTA TTTTGCCTGC GGTCAAAAAA
ATTGCAGTCA AACATTGTCA AGCAGGCGTG GCAGCAGCGC ATTATCCGAT TGTCGGTCAA
GAATTGTTGG GTGCGATTAA AGAAGTATTG GGCGATGCCG CAACCGATGA CATTTTGGAC
GCGTGGGGCA AGGCTTATGG CGTGATTGCA GATGTGTTTA TTCAAGTGGA AGCAGATTTG
TACGCTCAAG CGGTTGAATA A

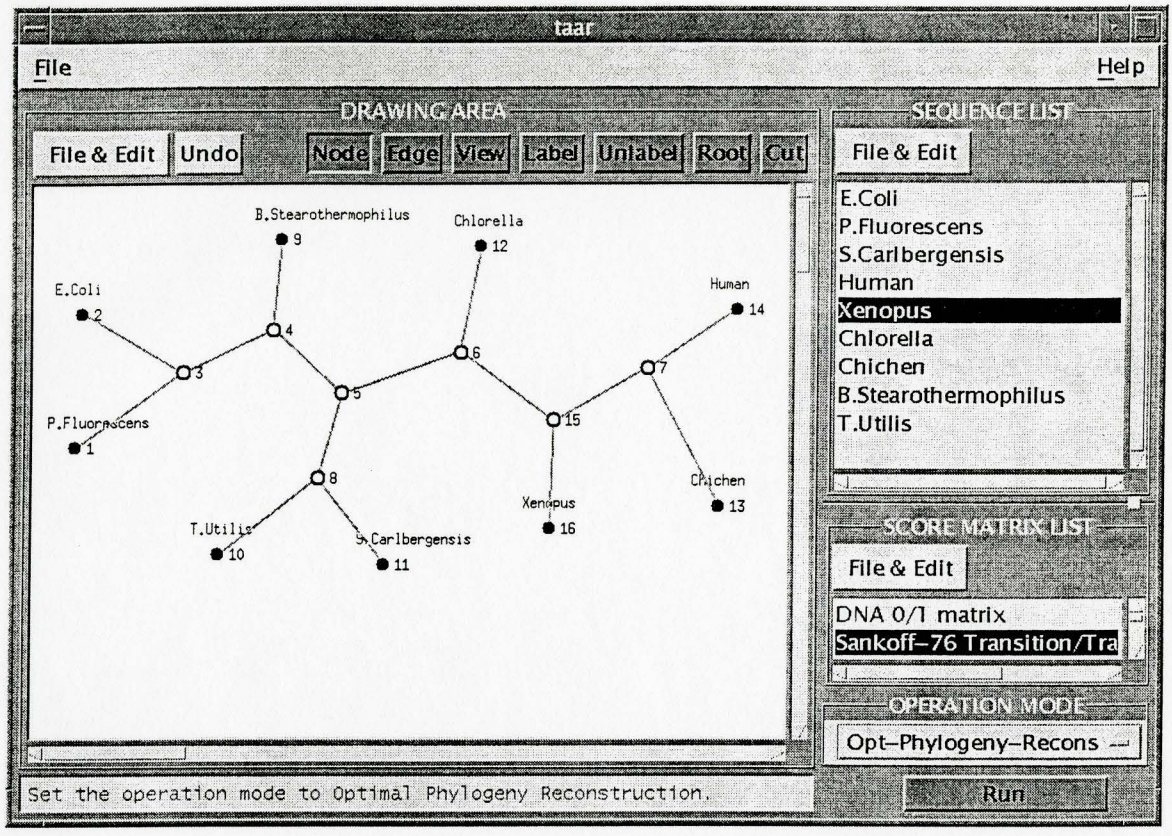
>duck-beta

ATGGTGCACT GGACAGCCGA GGAGAAGCAG CTCATCACCG GCCTCTGGGG CAAGGTCAAT
GTGGCCGACT GTGGAGCTGA GGCCCTGGCC AGGCTGCTGA TCGTCTACCC CTGGACCCAG
AGGTTCTTCG CCTCCTTCGG GAACCTGTCC AGCCCCACTG CCATCCTTGG CAACCCCATG
GTCCGTGCCC ATGGCAAGAA AGTGCTCACC TCCTTCGGAG ATGCTGTGAA GAACCTGGAC
AACATCAAGA ACACCTTCGC CCAGCTGTCC GAGCTGCACT GCGACAAGCT GCACGTGGAC
CCTGAGAACT TCAGGACCAA GGGTGCTCAT GTTCCCACAG CTCCTGGGTG ACATCCTCAT
CATCGTCCTG GCCGCCCACT TCACCAAGGA TTTCACTCCT GACTGCCAGG CCGCCTGGCA
GAAGCTGGTC CGCGTGGTGG CCCACGCTCT GGCCCGCAAG TACCACTAA

>oak

ATGAGCACCT TGGAAGGAAG AGGGTTCACC GAAGAGCAGG AAGCTCTAGT GGTTAAATCA
TGGAGTGCAA TGAAGCCTAA TGCTGGAGAA TTGGGTCTCA AGTTCTTCTT AAAGATATTT
GAGATCGCAC CGTCAGCACA GAAGCTCTTC TCTTTCTTGA AAGACTCTAA TGTTCCCCTG
GAACGGAACC CAAAGCTCAA GTCCCATGCC ATGTCTGTCT TTCTTATGAC CTGTGAATCC
GCAGTGCAGC TCCGGAAAGC TGGCAAAGTT ACTGTGAGGG AGTCAAGTTT GAAAAAGTTG
GGTGCTTCCC ACTTTAAACA TGGAGTGGCC GATGAACACT TTGAGGTAAC AAAATTTGCG
TTGCTGGAAA CAATCAAGGA AGCAGTCCCA GAGACGTGGT CGCCGGAGAT GAAGAATGCG
TGGGGAGAAG CTTACGATAA GTTGGTTGCT GCTATCAAAT TGGAAATGAA GCCTTCGTCT
TAG

2. Reconstructed phylogeny of 5S ribosomal RNA sequences by Opt-Phylogeny-Recons operation with Sankoff-76 Transition score matrix



3. Multiple sequence alignment by Fast-Gen-Tree-Align operation with Dayhoff PAM 250 score matrix and GAP Penalty 12

```
FAST ALIGNED BY SCORE MATRIX:  Dayhoff PAM 250 Matrix
OPEN GAP PENALTY:  12.00

SEQUENCE LENGTH: 189

SEQUENCE NAMES:

  1. HAHU Hemoglobin alpha chain - Human
  2. HAOR Hemoglobin alpha chain - Duckbill platypus
  3. HADK Hemoglobin alpha-A chain - Duck
  4. HBHU Hemoglobin beta chain - Human
  5. HBOR Hemoglobin beta chain - Duckbill platypus
  6. HBDK Hemoglobin beta chain - Duck
  7. MYHU Myoglobin - Human
  8. MYOR Myoglobin - Duckbill platypus
```

9. IGLOB Haemoglobin (Ctp HbVIIB-5)Midge (Chironomus thummi)
10. GPUGNI Nonlegume hemoglobin I - Swamp oak
11. GPYL Leghemoglobin I - Yellow lupine
12. GGZLB Bacterial hemoglobin - Vitreoscilla sp.

MULTI-ALIGNED SEQUENCES:

motif I

```
     1          11         21         31         41         51
 1.  -----V-LSP ADKTNVKAAW GKVGA----- ---HAG--EY GAEALERMFL SFPTTKTYF-
 2.  -----M-LTD AEKKEVTALW GKAAG----- ---HGE--EY GAEALERLFQ AFPTTKTYF-
 3.  -----V-LSA ADKTNVKGVF SKIGG----- ---HAE--EY GAETLERMFI AYPQTKTYF-
 4.  -----VHLTP EEKSAVTALW GKV------- ---NVD--EV GGEALGRLLV VYPWTQRFF-
 5.  -----VHLSG GEKSAVTNLW GKV------- ---NIN--EL GGEALGRLLV VYPWTQRFF-
 6.  -----VHWTA EEKQLITGLW GKV------- ---NVA--DC GAEALARLLI VYPWTQRFF-
 7.  -----G-LSD GEWQLVLNVW GKVEA----- ---DIP--GH GQEVLIRLFK GHPETLEKF-
 8.  -----G-LSD GEWQLVLKVW GKV------- ---EGDLPGH GQEVLIRLFK THPETLEKF-
 9.  MKFFAV-LAL CIVGAIASPL TADEASLVQS SWKAVS--HN EVEILAAVFA AYPDIQNKF-
10.  -----A-LTE KQEALLKQSW EVLKQ----- ---NIP--AH SLRLFALIIE AAPESKYVF-
11.  G----V-LTD VQVALVKSSF EEFNA----- ---NIP--KN THRFFTLVLE IAPGAKDLF-
12.  -----M-LDQ QTINIIKATV PVLKE----- ---HGV--TI TTTFYKNLFA KHPEVRPLFD
```

motif II

```
     61         71         81         91         101        111
 1.  ----PHF--- ---DLS--HG SAQVKGHGKK VADALTNAVA H--VDDMPN- -ALS---ALS
 2.  ----SHF--- ---DLS--HG SAQIKAHGKK VADALSTAAG H--FDDMDS- -ALS---ALS
 3.  ----PHF--- ---DLS--HG SAQIKAHGKK VAAALVEAVN H--VDDIAG- -ALS---KLS
 4.  ----ESFGDL STPDAV--MG NPKVKAHGKK VLGAFSDGLA H--LDNLKG- -TFA---TLS
 5.  ----EAFGDL SSAGAV--MG NPKVKAHGAK VLTSFGDALK N--LDDLKG- -TFA---KLS
 6.  ----ASFGNL SSPTAI--LG NPMVRAHGKK VLTSFGDAVK N--LDNIKN- -TFA---QLS
 7.  ----DKFKHL KSEDEM--KA SEDLKKHGAT VLTALGGILK K--KGHHEA- -EIK---PLA
 8.  ----DKFKGL KTEDEM--KA SADLKKHGGT VLTALGNILK K--KGQHEA- -ELK---PLA
 9.  ----SQFAGK ---DLASIKD TGAFATHATR IVSFLSEVIA L--SGNTSNA AAVN---SLV
10.  ----SFLKDS N--EIP--EN NPKLKAHAAV IFKTICESAT E--LRQKGH- -AVWDNNTLK
11.  ----SFLKGS S--EVP--QN NPDLQAHAGK VFKLTYEAAI QLEVNGAVA- -SDA---TLK
12.  MGRQESL--- ---EQP--KA LAMTVLAAAQ NIENLPAILP A--VKKIAV- -KHC---QAG
```

motif III motif IV motif V

```
     121        131        141        151        161        171
 1.  DLHAHKL--R VDPV--NFKL LSHCLLVTLA -AHLPAEFTP AV-HASLDKF LASVSTVLTS
 2.  DLHAHKL--R VDPV--NFKL LAHCILVVLA -RHCPGEFTP SA-HAAMDKF LSKVATVLTS
 3.  DLHAQKL--R VDPV--NFKF LGHCFLVVVA -IHHPAALTP EV-HASLDKF MCAVGAVLTA
 4.  ELHCDKL--H VDPE--NFRL LGNVLVCVLA -HHFGKEFTP PV-QAAYQKV VAGVANALAH
 5.  ELHCDKL--H VDPE--NFNR LGNVLIVVLA -RHFSKDFSP EV-QAAWQKL VSGVAHALGH
 6.  ELHCDKL--H VDPE--NFRL LGDILIIVLA -AHFTKDFTP EC-QAAWQKL VRVVAHALAR
 7.  QSHATKH--K IPVK--YLEF ISECIIQVLQ -SKHPGDFGA DA-QGAMNKA LELFRKDMAS
 8.  QSHATKH--K ISIK--FLEY ISEAIIHVLQ -SKHSADFGA DA-QAAMGKA LELFRNDMAA
 9.  SKLGDDH--K ARGV--SAAQ FGEFRTALVA YLQANVSWGD NV-AAAWNKA LDNTFAIVVP
10.  RLGSIHLKNK ITDP--HFEV MKGALLGTIK -EAIKENWSD EMGQAWTEAY NQLVATIKAE
11.  SLGSVHV--S KGVVDAHFPV VKEAILKTIK -EVVGDKWSE EL-NTAWTIA YDELAIIIKK
12.  VAAAHYP--I VGQE--LLGA IKEVLGDAAT -DDILDAWGK AY-GVIADVF IQVEADLYAQ
```

```
     181
 1.  KYR------
 2.  KYR------
 3.  KYR------
 4.  KYH------
 5.  KYH------
 6.  KYH------
 7.  NYKELGFQG
 8.  KYKEFGFQG
 9.  RL-------
10.  MKE------
11.  EMKDAA---
12.  AVE------
```

Bibliography

[1] Altschul, S. Gap Costs for Multiple Sequence Alignment, J. Theor. Biol. 138, pp. 279-309, 1989.

[2] Chan, S. C., Wong, A. K. C. and Chiu, D. K. T., A Survey of Multiple Sequence Comparison Methods, Bulletin of Mathematical Biology 54(4), pp. 563-598, 1992.

[3] Chen, X. D, An Interactive System for Sequence Analysis, M.Sc. thesis, McMaster University, 1994.

[4] Dayhoff, M.O., Schwartz, R. M. and Orcutt, B. C., A Model of Evolutionary Change in Proteins, Atlas of Protein Sequence and Structure, 5:345-352, 1978.

[5] Eppstein, D., Giancarlo, R. and Italiano, G. F., Sparse Dynamic Programming 1: Linear Cost Functions, J. Assoc. Comp. Machinery, 39(3), pp. 519-545, 1992.

[6] Flanagan, D., X Toolkit Intrinsics Reference Manual, Vol.5, O'Reilly & Associates, Inc., 1992.

[7] Gribskov, M. and Devereux, J., Sequence Analysis Primer, Stockton Press, 1991.

[8] Gusfield, D., Algorithms on Strings, Trees, and Sequences, Cambridge University Press, 1997.

[9] Gusfield, D., Efficient Methods for Multiple Sequence Alignment with Guaranteed Error Bounds, Bulletin of Mathematical Biology 55, pp. 141-154, 1993.

[10] Hein, J. J., A New Method that Simultaneously Aligns and Reconstructs Ancestral Sequences for Any Number of Homologous Sequences, When the Phylogeny is Given, Mol. Biol. Evol. 6, pp. 649-668, 1989.

[11] Hein, J. J., A Tree Reconstruction Method that is Economical in the Number of Pairwise Comparsions Used, Mol. Biol. Evol. 6, pp. 669-684, 1989.

[12] Heller, D. and Ferguson, P. M., Motif Programming Manual for OSF/Motif Release 1.2, Vol. 6A, O'Reilly & Associates, Inc., 1994.

[13] Hirschberg, D. S., A Linear Space Algorithm for Computing Longest Common Subsequences, Commun. Assoc. Comput. Mach., pp. 341-343, 1975.

[14] Jiang, T., Lawler, E. L. and Wang, L., Aligning Sequences Via an Evolutionary Tree: Complexity and Approximation, In Proc. of the Twenty-Sixth Annual Symposium on Theory of Computing, pp. 760-769, 1994.

[15] Jiang, T., Wang, L. and Lawler, E. L. Approximation Algorithms for Tree Alignment with a Given Phylogeny, Algorithmica, 16, pp. 302-315, 1996.

[16] Jiang, T. and Li, M., Optimization Problems in Molecular Biology, in Advances in Optimization and Approximation, Du, D. Z. and Sun, J. (eds.), Kluwer Academic Publishers, MA, 1994, 195-216.

[17] Myers, B. A. and Rosson, M. B., Survey on User Interface Programming, CHI'92 Conference Proceedings on Human Factors in Computer Systems, pp. 195-202, ACM Press, New York, 1992.

[18] Myers, E. W.and Miller, W., Optimal Alignments in Linear Space, CABIOS, 4(1), pp.11-17, 1988.

[19] Nye, A., Xlib Reference Manual, Vol.2, O'Reilly & Associates, Inc., 1992.

[20] Person, W.R. and Lipman, D.J., Improved Tools for Biological Sequence Comparison, Proc. Natl. Academy Science, 85:2444-48, 1988.

[21] Querie, V., and O'Reilly, T., X Window System User's Guide for X11 Release 5, Vol.3, O'Reilly & Associates, Inc., 1993.

[22] Saitou, N. and Nei, M., The Neighbor-Joining Method: A New Method for Reconstructing Phylogenetic Trees, Mol. Biol. Evol. 4-4, pp. 406-425, 1987.

[23] Sankoff, D., Cedergren, R. J. and Lapalme, G., Frequency of Insertion-Deletion, Transversion, and Transition in the Evolution of 5S Ribosomal RNA, Journal of Molecular Evolution 7, pp. 133-149, 1976.

[24] Sankoff, D. and Kruskal, J., Time Warps, String Edits, and Macromolecules: the Theory and Pactice of Sequence Comparison, Addison Wesley, Reading Mass., 1983.

[25] Sankoff, D., Minimal Mutation Trees of Sequences, ALAM J. APPL. Math. 28(1), pp. 35-42, 1975.

[26] Schwikowski, B. and Vingron, M., The Deferred Path Heurristic for the Generalized Tree Alignment Problem, Journal of Computational Biology, Vol.4, Num.3, Mary Ann Liebert, Inc., pp. 415-431, 1997.

[27] Smith, D. W., Biocomuting: Informatics and Genome Projects, Academic Press Inc., 1994.

[28] Swofford, D. L., Olsen, G. J., Waddell, P. J. and Hillis, D. M. Phylogenetic inference. In Hillis, D. M., Moritz, C. and Mable, B. K., editors, Molecular Systematics, 2nd Edition, Sinauer Associates, Sunderland Massachusetts, 1996.

[29] Wang, L. and Jiang, T., On the Complexity of Multiple Sequence Alignment, Journal of Computational Biology, 1-4, 337-348, 1994.

[30] Waterman, M. S., Introduction To Computational Biology: Maps, sequences and genomes, Chapman & Hall, London, 1995.

[31] Waterman, M. S., Mathematical Methods for DNA Sequences, CRC Press, 1989.