國立交通大學

生物資訊所

碩 士 論 文

MuSiC and MuSiC-ME：有效率的限制型多重序列比對工具

MuSiC and MuSiC-ME: Efficient Tools for Multiple Sequence Alignment with Constraints

研 究 生：黃彥斌

指導教授：盧錦隆　教授

中 華 民 國 九 十 三 年 六 月

MuSic and MuSiC-ME：有效率的限制型多重序列比對工具

MuSiC and MuSiC-ME: Efficient Tools for Multiple Sequence Alignment with Constraints

研 究 生：黃彥斌　　Student：Yen Pin Huang

指導教授：盧錦隆 教授　　Advisor：Prof. Chin Lung Lu

國 立 交 通 大 學

生 物 資 訊 所

碩 士 論 文

A Thesis Submitted to Institute of Bioinformatics

College of Biological Science and Technology

National Chiao Tung University in partial Fulfillment of the Requirements

for the Degree of Master in

Biological Science and Technology

June 2004

Hsinchu, Taiwan, Republic of China

中華民國九十三年六月

中文摘要

在生物資訊及計算生物的領域中, 多重序列比對 (Multiple Sequence Alignment) 在發掘基因體或蛋白質序列的生物意義上是很有用的工具。通常生物學家對序列的結構/功能/演化關係已有一些初步的認識, 如活化部位的殘基、分子間的雙硫鍵、受質結合的部位、酵素的活性及保守性的 Motifs 。因此, 在做多重序列比對的時候, 生物學家希望能有一個工具能夠讓一些結構性的/功能性的/保留性的核甘酸或殘基可以排比在一起。然而, 目前已有的多重序列比對工具大都無法滿足這種需求, 因爲他們都只根據序列的內容去排比而忽略了其他在結構/功能/演化上已知的訊息。因此, 在這個論文中, 我們主要的目的是去研究發展一種稱爲限制型的多重序列比對工具來解決這個問題。這個工具允許使用者輸入多條的序列及一些限制條件, 每個限制條件對應在序列上一些結構性的/功能性的/保守性的核甘酸或殘基, 然後產生一組多重序列比對使得一些滿足限制條件的核甘酸或殘基排比在一起。我們採用了所謂的 Progressive 的方法來解決限制型的多重序列比對問題。事實上, 這個方法的核心在於能否設計出有效率的演算法來解決限制型兩條序列比對問題 (Constrained Pairwise Sequence Alignment Problem)。我們採用 Dynamic Programming 及 Divide-and-Conquer 這兩種不同的方法分別設計出一個在時間上有效率及另一個在空間上有效率的演算法來求得最佳化的限制型的兩條序列比對。然後, 我們再根據這兩種演算法分別發展出兩個限制型多重序列比對工具: MuSiC (<u>Mu</u>ltiple <u>S</u>euqnece Al<u>i</u>gnment with <u>C</u>onstraints) 及 MuSiC-ME (<u>M</u>emory-<u>E</u>fficient MuSiC)。同時, 我們也利用一些冠狀病毒 (包含 SARS) 的 $3'$ UTR 序列來測試這兩種工具的可用性, 並從其所產生的序列比對中找到在 SARS 病毒中會折疊成 Pseudoknot 結構的片斷。

Abstract

Multiple sequence alignment (MSA) has received much attention in the fields of bioinformatics and computational biology because it is very useful for discovering the biological meanings of sequences. Usually, biologists may have advanced knowledge about the structures/functionalities/evolutionary relationships of sequences of interest, such as active site residues, intramolecular disulfide bonds, substrate binding sites, enzyme activities and conserved motifs (consensuses). They would expect an MSA program that is able to align these sequences such that the structural/functional/conserved bases (i.e., nucleotides or residues) are aligned together. However, most available MSA programs cannot satisfy such a requirement because they generate an alignment based only on the content of the sequences, ignoring the known functional/structural/conserved information. Hence, in this thesis, we study and develop a so-called constrained multiple sequence alignment (CMSA) tool, which takes as input the sequences and several user-specified constraints, each with corresponding to the known functional/structural/conserved bases, and generates an output of alignment in which the bases corresponding to a user-specified constraint are aligned together. We use the progressive approach to design efficient programs for heuristically solving the CMSA problem. The kernel of this approach is an efficient algorithms for optimally solving the constrained pairwise alignment (CPSA) problem. We use two different approaches, called as dynamic programming and divide-and-conquer, to design a time-efficient algorithm and a memory-efficient algorithm respectively for optimally solving the CPSA program. Based on those two algorithms, we then develop two programs, called MuSiC (Multiple Sequence Alignment with Constraints) and MuSiC-ME (Memory-Efficient MuSiC), respectively. To demonstrate the applicabilities of our programs, we test them on a data set of RNA sequences of 3′ UTRs of several coronaviruses, including SARS, for detecting a fragment in the 3′ UTR of SARS that is able to fold into a stable pseudoknotted structure, where such a pseudoknot is considered to be involved in the RNA replication of coronaviruses.

誌 謝

感謝我的指導教授盧錦隆老師耐心的指導, 教導了我許多做研究的方法, 並且在我研究上遭遇瓶頸時, 不厭其煩的提供協助; 感謝曾經幫助過我的師長, 以及時常鼓勵我的同學、學弟妹及朋友; 感謝家人的栽培與支持, 讓我得以無憂無慮的完成碩士學位。

Contents

List of Figures

Chapter 1

Introduction

Multiple sequence alignment (MSA) is one of the fundamental problems in bioinformatics and computational biology that have been studied extensively, because it is a useful tool in the phylogenetic analyses among various organisms, identification of conserved motifs and domains in a group of related proteins, secondary and tertiary structure prediction of a protein/RNA and so on [4, 5, 14, 23, 24]. The sum-of-pairs score is widely used for selecting an optimal MSA. This kind of MSA problem, called sum-of-pairs MSA (SPMSA) problem, can be solved by extending the dynamic programming algorithm of Needleman and Wunsch for aligning two sequence [22]. In the worst case, it needs to take $\mathcal{O}(2^k n^k)$ time to align k sequences of length n. This exponential time limits the dynamic programming technique to align only a small number of short sequences. Actually, the SPMSA problem has been shown to be NP-complete [3, 40], which means that it seems to be impossible to design an efficient algorithm to find the mathematically optimal alignment. Hence, some approximate and heuristic methods are introduced to overcome this problem.

For the approximate methods, Gusfield [13] first proposed a polynomial time approximation algorithm with performance ratio of $2 - \frac{2}{k}$. Then Pevzner [26] improved the performance ratio to $2 - \frac{3}{k}$. Recently, Bafna, Lawler and Pevzner [2] further improved the performance ratio to $2 - \frac{l}{k}$ for any fixed l. It is worth mentioning that Li, Ma and Wang [19] have given a polynomial time approximation scheme for finding a multiple sequence alignment within a constant band, which is often useful in many practical cases. For the heuristic methods, the most widely used heuristic methods are so-called progressive strategies [8, 11, 16, 35, 37].

Usually, biologists may have advanced knowledge about the structures/functionalities/evolutionary relationships of sequences of interest, such as active site residues, intramolecular disulfide bonds, substrate binding sites, enzyme activities and conserved motifs (consensuses). They would expect an MSA program that is able to align these sequences such that the structural/functional/conserved bases (i.e., nucleotides or residues) are aligned together. However, the currently existing MSA tools, such as CLUSTAL W, do not guarantee that the generated alignments meet such a requirement that some particular bases would be aligned together. Hence, Tang *et al.* [34] defined and studied the so-called constrained multiple sequence alignment (CMSA) problem that given the input sequences with several user-specified constraints, generates an MSA in which the bases corresponding to a user-specified constraint are aligned together, where each user-specified constraint corresponds to the know functional/structural/constrained bases. Tang *et al.* designed a dynamic programming algorithm for finding an optimal constrained alignment of two sequences and then used it as a kernel to develop a constrained multiple sequence alignment tool based on the progressive method, where each constraint used by Tang *et al.* is a single base. Their proposed algorithm for two sequences runs in $\mathcal{O}(\gamma n^4)$ time and consumes $\mathcal{O}(n^4)$ space, where γ is the number of the constrained bases and n is the maximum length of sequences. Later, this result was improved independently to $\mathcal{O}(\gamma n^2)$ time and $\mathcal{O}(\gamma n^2)$ space using the same approach of dynamic programming [7, 44].

In fact, each of the columns requested to be aligned together can represent a conserved site of a protein/DNA/RNA family and each conserved site may consist of a short segment of bases, instead of a single base. In other words, the user-specified constraint may be a fragment of bases. For some applications, biologists may further expect that some mismatches are allowed among the bases of the columns requested to be aligned. Hence, in this thesis, we consider a more generalized CMSA problem in which the user-specified constraints are a fragment of bases and some mismatched may occur in the columns requested to be aligned.

We use the progressive approach to design efficient programs for heuristically solving the CMSA problem. The kernel of this approach is an efficient algorithms for optimally solving the constrained pairwise alignment (CPSA) problem. First of all, we use the dynamic programming technique to design an algorithm of $\mathcal{O}(\gamma n^2)$ time and $\mathcal{O}(\gamma n^2)$ space optimally solving CPSA and then use this algorithm as the kernel to develope CMSA tool, called as MuSiC (Multiple Sequence Alignment with Constraints). The result greatly increases the performances and practical usage of the CMSA tools developed by the progressive approach. However, the requirement of $\mathcal{O}(\gamma n^2)$ memory still limits it to align a set of short sequences, at most several hundreds of bases. To align large genomic sequences, there is a need to design a memory-efficient algorithm for the CPSA problem, which is the key limiting factor relating to the applicable extent of the progressive CMSA tools. Hence, in the second part, we adopt the so-called divide-and-conquer approach to design a memory-efficient algorithm for optimally solving the CPSA problem, which runs in $\mathcal{O}(\gamma n^2)$ time, but consumes only $\mathcal{O}(\gamma\lambda n)$ space, where λ is the maximum of the lengths of constraints and usually $\lambda << n$ in practical applications. Based on this algorithm, we have finally developed a memory-efficient CMSA tool, called as MuSiC-ME (Memory-Efficient MuSiC), using the progressive approach.

To demonstrate the applicabilities of our programs, we test them on a data set of RNA sequences of $3'$ untranslated regions (UTRs) of several coronaviruses, including SARS, for detecting a fragment in the $3'$ UTR of SARS that is able to fold into a stable pseudoknotted structure, where such a pseudoknot is considered to be involve in the RNA replication of coronaviruses.

The rest of this thesis is organized as follows. In Chapter 2, we give the formal definition of the CMSA problem we study in this thesis and also introduce the main steps of the adopted progressive MSA. In Chapter 3, we first use the dynamic programming technique to design a time-efficient algorithm for optimally solving the CPSA problem, then use the divide-and-conquer technique to design a memory-efficient algorithm for the CPSA problem, and finally used the progressive approach to develop two CMSA programs. In Chapter 4, we demonstrate the applicability of our developed programs by testing them on a data set of RNA sequences. Finally, we make some conclusions in Chapter 5.

Chapter 2

Preliminaries

2.1 Problem Formulation

Let $\mathcal{S} = \{S_1, S_2, \cdots, S_k\}$ be the set of k sequences over the alphabet Σ. Then a *multiple sequence alignment* (MSA) of $\mathcal{S}$ is a rectangular matrix consisting of k rows of characters of $\Sigma \cup \{\text{-}\}$ such that no column consists entirely of dashes and removing dashes from row i leaves S_i for any $1 \leq i \leq k$. The *sum-of-pairs score* (SP score) of an MSA is defined to be the sum of the scores of all columns, where the score of each column is the sum of the scores of all distinct pairs of characters in the column. In practice, the score of the pair of two dashes is usually set to zero. Then the problem of finding an MSA of $\mathcal{S}$ with the optimal SP score is the so-called *sum-of-pairs MSA problem* [4, 5, 14, 23, 24].

Let $l(P)$ be the length of a fragment sequence P. Let $d_H(P', P'')$ denote the *Hamming distance* between two fragments P' and P'' of equal length (i.e., $l(P') = l(P'')$), which is equal to the number of mismatched pairs in the alignment of P' and P'' without any gap. Given an alignment $\mathcal{L}$ of $\mathcal{S}$, a *band* is defined as a block of consecutive columns in $\mathcal{L}$ (i.e., a submatrix of $\mathcal{L}$). For any band $\mathcal{B}$ of $\mathcal{L}$, $\mathcal{B}(S_i)$ denotes the fragment of S_i whose residues/nucleotides are all in the band $\mathcal{B}$, where $1 \leq i \leq k$. A sequence $S = s_1 s_2 \ldots s_\lambda$ is said to *appear* in $\mathcal{L}$ if $\mathcal{L}$ contains a band $\mathcal{B}$ of λ columns, say $x_1, x_2, \ldots, x_\lambda$, such that the characters of column x_j, where $1 \leq j \leq \lambda$, are all equal to s_j, or equivalently, $\mathcal{B}(S_i) = S$ for each $1 \leq i \leq k$. If $d_H(\mathcal{B}(S_i), S) \leq l(S) \times \epsilon$ for a given error ratio $0 \le \epsilon < 1$ (i.e., some mismatches are allowed between $\mathcal{B}(S_i)$ and S), then S is said to *approximately appear* in $\mathcal{L}$. From the biological viewpoint, S can be considered as the *consensus* among the fragment sequences in $\mathcal{B}$ and hence S is also called as an *induced consensus* by the band $\mathcal{B}$. For any two sequences S' and S'', $S' \prec S''$ is used to denote that S' (approximately) appears *strictly before* S'' in $\mathcal{L}$ (i.e., their corresponding bands do not overlap). Let $\mathcal{C} = (C_1, C_2, \ldots, C_\gamma)$ be a ordered set of γ constraints, each $C_i = c_{i,1}c_{i,2} \ldots c_{i,\lambda_i}$ with length of λ_i. Then the *constrained multiple sequence alignment* (CMSA) of $\mathcal{S}$ *with respect to* $\mathcal{C}$ is defined to be an MSA $\mathcal{L}$ of $\mathcal{S}$ in which all constraints of $\mathcal{C}$ approximately appear in the order $C_1 \prec C_2 \prec \ldots \prec C_\gamma$ such that $d_H(\mathcal{B}_j(S_i), C_j) \le \lambda_j \times \epsilon$ for each band $\mathcal{B}_j$ whose induced consensus is C_j, where $1 \le i \le k$ and $1 \le j \le \gamma$. Given a set $\mathcal{S}$ of k sequences along with an ordered set $\mathcal{C}$ of γ constraints and an error ratio ϵ, the so-called *constrained multiple sequence alignment problem* is to find a CMSA w.r.t. $\mathcal{C}$ with the optimal SP score.

2.2 Progressive multiple sequence alignment

The progressive approach is one of the widely used heuristics for efficiently finding a good MSA of several sequences. The ideas behind it are as follows [8, 11, 16, 35, 37]:

1. Compute the distance matrix by aligning all pairs of sequences: Usually, this distance matrix is obtained by applying FASTA [20, 27] or the dynamic programming algorithm of Needleman and Wunsch [22] to each pair of sequences.

2. Construct the guide tree from the distance matrix: For the existing progressive methods, they mainly differ in the method used to build the guide tree for directing the order of alignment of sequence. To build the guide tree, for example, PILEUP (a program of GCG packages) uses UPGMA (Unweighted Pair-Group Method using Arithmetic mean) method [33] and CLUSTAL W [37] uses NJ (Neighbor-Joining) method [31].

3. Progressively align the sequences according to the branching order in the guide tree: Initially, the closest two sequences in the tree are aligned using the normal

dynamic programming algorithm. After aligning, this pair of sequences is fixed and any introduced gaps cannot be shifted later (i,e., once a gap, always a gap). Then the next two closest pre-aligned groups of sequences are joined in the same way until all sequences have been aligned. (Here, we may consider a sequence as an aligned group of a sequence.) To align two groups of the pre-aligned sequences, the score between any two positions in these two groups is usually the arithmetic average of the scores for all possible character comparisons at those positions. We call this kind of scoring methods as a *set-to-set scoring*.

In fact, MST has been used as a significant tool for data classification in the fields of biological data analysis. In [34], Tang *et al.* proposed a variant of progressive method by using the Kruskal MST to construct the guide tree, called Kruskal merging order tree. The Kruskal merging order tree of k sequences is constructed as follows. First, we create a complete graph $G = (V, E)$ of k sequences in a way that each vertex of V represents a sequence and each edge e of E is associated with a weight $d(e)$ to represent the distance between the corresponding sequences of its end-vertices. Then we use the Kruskal's algorithm [18] to construct the Kruskal MST of G, denoted by $\mathcal{T}$. For completeness, we describe the Kruskal method for constructing $\mathcal{T}$ as follows.

1. Sort all edges of E in non-decreasing order according to their distances.

2. Initially, $\mathcal{T}$ is empty. Then we repeatedly add the edges of E in non-decreasing order to $\mathcal{T}$ in a way that if the currently adding edge e to $\mathcal{T}$ dose not create a cycle in $\mathcal{T}$, then we add e to $\mathcal{T}$; otherwise, we discard e.

Next, according to the Kruskal MST $\mathcal{T}$, we build the Kruskal merging order tree $\mathcal{T}_{\mathcal{K}}$ as follows.

1. Let $V = \{v_1, v_2, \ldots, v_k\}$ and $e_1, e_2, \ldots, e_{k-1}$ be the edges of $\mathcal{T}$ with $d(e_1) \leq d(e_2) \leq \ldots \leq d(e_{k-1})$.

2. For each vertex $v_i \in V$, we create a tree $\mathcal{T}_i$ such that $\mathcal{T}_i$ contains only a node v_i. For the purpose of merging trees, we consider $\mathcal{T}_i$ as a *rooted tree* by designating v_i as its root, and define the *merge* of two tree $\mathcal{T}_i$ and $\mathcal{T}_j$ respectively rooted at

v_i and v_j to be a new tree rooted at a new vertex u such that v_i and v_j become the children of u.

3. For each $e_K = (v_i, v_j)$, where K increases from 1 to $k-1$, we find the trees $\mathcal{T}_i$ and $\mathcal{T}_j$ containing v_i and v_j respectively and then merge them into a new tree. This process is continued until the remaining is only one tree. Then this final tree is the so-called *Kruskal merging order tree* $\mathcal{T}_{\mathcal{K}}$.

The construction of G for k sequences can be done in $\mathcal{O}(k^2)$ time and the computation of the Kruskal's MST $\mathcal{T}$ of G can be done in $\mathcal{O}(k^2 \log k)$ time. Then the construction of $\mathcal{T}_{\mathcal{K}}$ from $\mathcal{T}$ can be implemented by the disjoint set union and find algorithm proposed by Gabow and Tarjan [12] in $\mathcal{O}(m+k)$ time, where m denotes the number of union and find operations. It is not hard to see that $m = \mathcal{O}(k)$ and hence the construction of $\mathcal{T}_{\mathcal{K}}$ takes $\mathcal{O}(k)$ time. Therefore, the total time complexity of constructing $\mathcal{T}_{\mathcal{K}}$ is $\mathcal{O}(k^2 \log k)$.

Chapter 3

Algorithms

Here, we describe our algorithms to efficiently solve the CMSA problem based on the progressive approach adopted by Tang *et al.* [34]. The ideas behind this progressive approach are first to design an efficient algorithm to optimally solve the constrained pairwise sequence alignment (CPSA) and then use it as a kernel to progressively align the input sequences into a CMSA according to the branching order of a guide tree. The main different part of our progressive algorithm from Tang's is the algorithms for solving the CPSA problem. In the following, we first use the dynamic programming technique to design a time-efficient algorithm for optimally solving the CPSA problem, then use the divide-and-conquer technique to design a memory-efficient algorithm for the CPSA problem, and finally use the progressive approach to develop two CMSA programs.

3.1 Constrained pairwise sequence alignment

3.1.1 Dynamic programming method

In this section, we shall use dynamic programming method to design a time-efficient algorithm for solving the CPSA problem with two given sequences $A = a_1a_2 \ldots a_m$ and $B = b_1b_2 \ldots b_n$, a given order set $\mathcal{C} = (C_1, C_2, \ldots, C_\gamma)$ of γ constraints and a given error threshold ϵ.

For any two characters $a, b \in \Sigma$, let $\sigma(a, b)$ denote the score of aligning a with b.

The gap penalty adopted here is the so-called *affine gap penalty* that penalizes a gap of length l with $w_o + l \times w_e$, where $w_o > 0$ is the gap-open penalty and $w_e > 0$ is the gap-extension penalty. For convenience, let $A_i = a_1a_2 \ldots a_i$, $B_j = b_1b_2 \ldots b_j$ and $\mathcal{C}_k = (C_1, C_2, \ldots, C_k)$, where $1 \leq i \leq m$, $1 \leq j \leq n$, $1 \leq k \leq \gamma$. Let $\mathcal{M}_k(i,j)$ denote the score of an optimal constrained alignment of A_i and B_j w.r.t. $\mathcal{C}_k$. Clearly, $\mathcal{M}_\gamma(m,n)$ is the score of an optimal constrained alignment of A and B w.r.t. $\mathcal{C}$. $\mathcal{L}$ is called as a *semi-constrained alignment* of A_i and B_j w.r.t. $\mathcal{C}_k$ if it is a constrained alignment of A_i and B_j w.r.t. $\mathcal{C}_{k-1}$ and also ends (or begins) with a band whose induced consensus is equal to a prefix of C_k (or a suffix of C_1). $\mathcal{N}_k(i,j,h)$ is defined to be the score of an optimal semi-constrained alignment of A_i and B_j w.r.t. $\mathcal{C}_k$ that ends with an induced consensus equal to $C_{k,h}$, where $C_{k,h} = c_{k,1}c_{k,2} \ldots c_{k,h}$. Let $\mathcal{M}_k^D(i,j)$ and $\mathcal{M}_k^I(i,j)$ be the maximum scores of all constrained alignments of A_i and B_j w.r.t. $\mathcal{C}_k$ that end with a deletion pair (i.e., $(a_i, -)$) and an insertion pair (i.e., $(-, b_j)$), respectively. By the definition, it is not hard to derive the recurrence of $\mathcal{M}_k(i,j)$, where $1 \leq i \leq m$ and $1 \leq j \leq n$, as follows. If $k = 0$, then

$$\mathcal{M}_k(i,j) = \max \begin{cases} \mathcal{M}_k(i-1, j-1) + \sigma(a_i, b_j), \\ \mathcal{M}_k^D(i,j), \\ \mathcal{M}_k^I(i,j). \end{cases}$$

If $1 \leq k \leq \gamma$, then

$$\mathcal{M}_k(i,j) = \max \begin{cases} \mathcal{M}_k(i-1, j-1) + \sigma(a_i, b_j), \\ \mathcal{M}_k^D(i,j), \\ \mathcal{M}_k^I(i,j), \\ \mathcal{N}_k(i,j,\lambda_k). \end{cases}$$

Clearly, $\mathcal{N}_k(i,j,\lambda_k) = \mathcal{M}_{k-1}(i-\lambda_k, j-\lambda_k) + \Sigma_{0 \leq h \leq \lambda_k - 1}\sigma(a_{i-h}, b_{i-h})$, if $d_H(A_i(\lambda_k), C_k) \leq \lambda_k \times \epsilon$ and $d_H(B_j(\lambda_k), C_k) \leq \lambda_k \times \epsilon$, where $A_i(\lambda_k) = a_{i-\lambda_k+1} \ldots a_i$ and $B_j(\lambda_k) = b_{j-\lambda_k+1} \ldots b_j$. Otherwise, $\mathcal{N}_k(i,j,\lambda_k) = -\infty$. To simply describe the computation of $\mathcal{M}_k^D(i,j)$ and $\mathcal{M}_k^I(i,j)$, we introduce another notation $\mathcal{M}_k^S(i,j)$ which is defined to be the maximum score of all constrained alignments of A_i and B_j w.r.t. $\mathcal{C}_k$ that end with a substitution pair (i.e., (a_i, b_j)). Let $\mathcal{L}_k^D(A_i, B_j)$ denote the alignment of A_i and B_j

with score $\mathcal{M}_k^D(i,j)$ which ends with a deletion pair $(a_i, -)$. Let $\mathcal{L}'$ be the portion of $\mathcal{L}_k^D(A_i, B_j)$ before the last aligned pair $(a_i, -)$. Then there are three possibilities when we consider the last aligned pair of $\mathcal{L}'$.

Case 1: The last aligned pair of $\mathcal{L}'$ is a substitution pair. Then the score of $\mathcal{L}'$ is $\mathcal{M}_k^S(i-1,j)$ and $(a_i, -)$ is charged by a gap-open penalty and a gap-extension penalty in $\mathcal{M}_k^D(i,j)$. Hence, $\mathcal{M}_k^D(i,j) = \mathcal{M}_k^S(i-1,j) - w_o - w_e$.

Case 2: The last aligned pair of $\mathcal{L}'$ is a deletion pair. Then the score of $\mathcal{L}'$ is $\mathcal{M}_k^D(i-1,j)$ and $(a_i, -)$ is charged by only one gap-extension penalty in $\mathcal{M}_k^D(i,j)$. Hence, $\mathcal{M}_k^D(i,j) = \mathcal{M}_k^D(i-1,j) - w_e$.

Case 3: The last aligned pair of $\mathcal{L}'$ is an insertion pair. Then the score of $\mathcal{L}'$ is $\mathcal{M}_k^I(i-1,j)$ and $(a_i, -)$ is charged by a gap-open penalty and a gap-extension penalty in $\mathcal{M}_k^D(i,j)$. Hence, $\mathcal{M}_k^D(i,j) = \mathcal{M}_k^I(i-1,j) - w_o - w_e$.

In summary, we have

$$\mathcal{M}_k^D(i,j) = \max \begin{cases} \mathcal{M}_k^S(i-1,j) - w_o - w_e, \\ \mathcal{M}_k^D(i-1,j) - w_e, \\ \mathcal{M}_k^I(i-1,j) - w_o - w_e. \end{cases}$$

However, by including an extra $\mathcal{M}_k^D(i-1,j) - w_o - w_e$ into the right-hand site of the above recurrence, we can reformulate the above recurrence as

$$\mathcal{M}_k^D(i,j) = \max \begin{cases} \mathcal{M}_k(i-1,j) - w_o - w_e, \\ \mathcal{M}_k^D(i-1,j) - w_e. \end{cases}$$

Similar to the discussion above, the recurrence of $\mathcal{M}_k^I(i,j)$ can be derived as

$$\mathcal{M}_k^I(i,j) = \max \begin{cases} \mathcal{M}_k(i,j-1) - w_o - w_e, \\ \mathcal{M}_k^I(i,j-1) - w_e. \end{cases}$$

The initializations of $\mathcal{M}_k(i,j), \mathcal{M}_k^D(i,j)$ and $\mathcal{M}_k^I(i,j)$ for all $0 \le k \le \gamma$ are as follows.

- If $k = 0$, then $\mathcal{M}_k(0,0) = 0$, $\mathcal{M}_k^D(0,0) = \mathcal{M}_k^I(0,0) = -\infty$, $\mathcal{M}_k(i,0) = \mathcal{M}_k^D(i,0) = -w_o - iw_e$ and $\mathcal{M}_k^I(i,0) = -\infty$ for all $1 \le i \le m$, and $\mathcal{M}_k(0,j) = \mathcal{M}_k^I(0,j) = -w_o - jw_e$ and $\mathcal{M}_k^D(0,j) = -\infty$ for all $1 \le j \le n$.

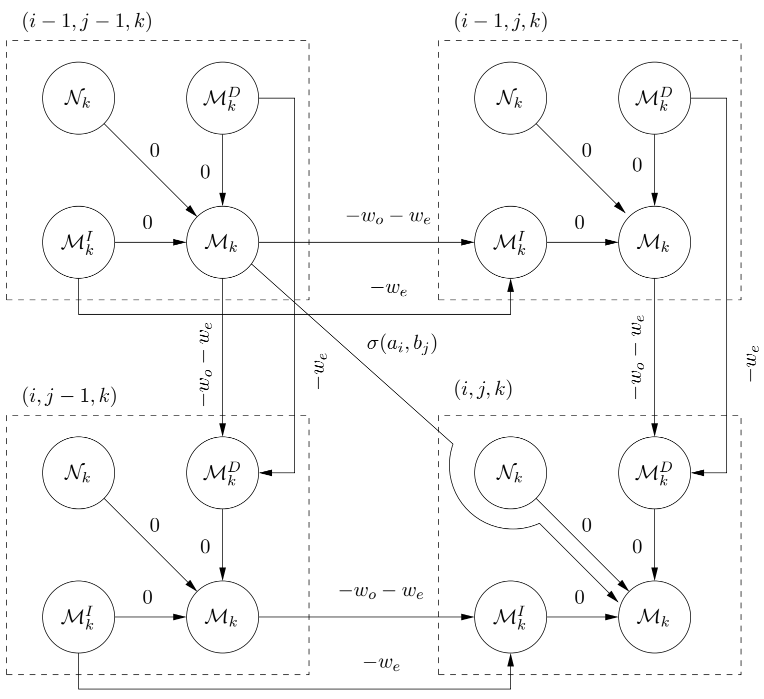


Figure 3.1: The schematic diagram of four adjacent entries of $\mathcal{G}$, where entry (i, j, k) consists of three nodes $\mathcal{M}_k, \mathcal{M}_k^D$ and $\mathcal{M}_k^I$ corresponding to $\mathcal{M}_k(i, j), \mathcal{M}_k^D(i, j), \mathcal{M}_k^I(i, j)$, respectively.

- If $1 \leq k \leq \gamma$, then $\mathcal{M}_k(0, 0) = 0$, $\mathcal{M}_k^D(0, 0) = \mathcal{M}_k^I(0, 0) = -\infty$, $\mathcal{M}_k(i, j) = \mathcal{M}_k^D(i, j) = \mathcal{M}_k^I(i, j) = -\infty$ for all $1 \leq i \leq m$ and $1 \leq j \leq n$.

According to the recurrences above, we can design an algorithm to compute $\mathcal{M}_\gamma(m, n)$ and its corresponding constrained alignment using the technique of dynamic programming as follows. For convenience, we can depict the recurrences of matrices $\mathcal{M}_k, \mathcal{M}_k^D, \mathcal{M}_k^I$ and $\mathcal{N}_k$ for all $0 \leq k \leq \gamma$ by a 3D grid graph $\mathcal{G}$, which consists of $(m + 1) \times (n + 1) \times (\gamma + 1)$ entries and each entry (i, j, k) consists of four nodes $\mathcal{M}_k, \mathcal{M}_k^D$, $\mathcal{M}_k^I$ and $\mathcal{N}_k$ corresponding to $\mathcal{M}_k(i, j), \mathcal{M}_k^D(i, j), \mathcal{M}_k^I(i, j)$, and $\mathcal{N}_k(i, j, \lambda_k)$, respectively. Figure 3.1 illustrates the relationship of four adjacent entries $(i, j, k), (i-1, j, k), (i, j-1, k)$ and $(i-1, j-1, k)$ of $\mathcal{G}$ for each fixed k. Note that there is a directed edge, which is not shown in Figure 3.1, with weight $\Sigma_{0 \leq h \leq \lambda_k - 1} \sigma(a_{i-h}, b_{j-h})$ from the $\mathcal{M}_{k-1}$ node of the entry $(i - \lambda_k, j - \lambda_k, k - 1)$ to the $\mathcal{N}_k$ node of the entry (i, j, k). Then each path from $\mathcal{M}_0(0, 0)$ node of entry $(0, 0, 0)$ to $\mathcal{M}_\gamma(m, n)$ node of entry (m, n, γ) corresponds to a constrained alignment of A and B w.r.t. $\mathcal{C}$. As a result, an optimal constrained alignment of A and B can be obtained by backtracking a shortest path from $\mathcal{M}_\gamma(m, n)$ to $\mathcal{M}_0(0, 0)$ in $\mathcal{G}$. It is not hard to see that the algorithm costs both computer time and memory in the order of $\mathcal{O}(\gamma mn)$. We call the above algorithm based on the dynamic programming approach as CPSA-DP algorithm.

3.1.2 Divide and conquer method

In this section, we shall use divide-and-conquer method to design a memory-efficient algorithm for solving the CPSA problem with two given sequences $A = a_1a_2 \dots a_m$ and $B = b_1b_2 \dots b_n$, a given order set $\mathcal{C} = (C_1, C_2, \dots, C_\gamma)$ of γ constraints and a given error threshold ϵ.

Recall that Hirschberg [17] developed a linear-space algorithm for solving the longest common subsequence problem based on the technique of divide and conquer. Since then, this strategy has been extended to yield a number of memory-efficient algorithms for aligning biological sequences [6, 21]. In this paper, we generalize the Hirschberg's algorithm so that it is able to deal with the constrained pairwise sequence alignment. As compared with others, our generalization is more complicated because the grid graph $\mathcal{G}$ we deal with here is 3D, instead of 2D, and the input sequences are accompanied with several constraints which need to be considered carefully. The central idea of our memory-efficient algorithm is to determine a middle position $(i_{mid}, j_{mid}, k_{mid})$ on an optimal path from $\mathcal{M}_0(0, 0)$ to $\mathcal{M}_\gamma(m, n)$ in $\mathcal{G}$ so that we are able to divide the constrained alignment problem into two smaller constrained alignment problems, then these smaller constrained alignment problems are continued to be divided in the same way, and finally the optimal constrained alignment is obtained completely by merging the series of the calculated mid-points (see Figure 3.2 for an illustration).

Before describing our algorithm, some notation must be introduced as follows. Let $\overline{A}_i$ and $\overline{B}_j$ denote the suffixes $a_{i+1}a_{i+2} \dots a_m$ and $b_{j+1}b_{j+2} \dots b_n$ of A and B, respectively, for $1 \le i \le m$ and $1 \le j \le n$. Let $\overline{\mathcal{C}}_k$ denote the ordered subset $(C_{k+1}, C_{k+2}, \dots, C_\gamma)$ for $1 \le k \le \gamma$. Define $\overline{\mathcal{M}}_k(i, j)$ to be the score of an optimal constrained alignment of $\overline{A}_i$ and $\overline{B}_j$ w.r.t. $\overline{\mathcal{C}}_k$, and define $\overline{\mathcal{M}}_k^S(i, j)$, $\overline{\mathcal{M}}_k^D(i, j)$ and $\overline{\mathcal{M}}_k^I(i, j)$ to be the maximum score of all constrained alignments of $\overline{A}_i$ and $\overline{B}_j$ w.r.t. $\overline{\mathcal{C}}_k$ that begin with a substitution pair

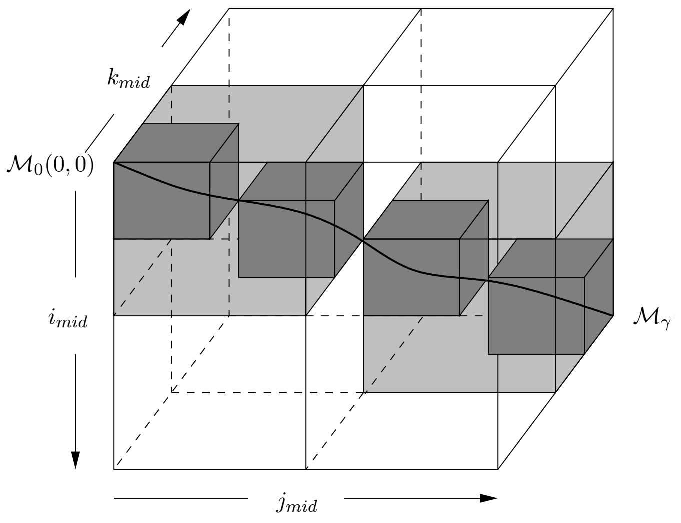


Figure 3.2: Schematic diagram of divide and conquer approach: two light gray areas are the reduced subproblems after middle position $(i_{mid}, j_{mid}, k_{mid})$ is determined, each of which will be further divided into two subproblems of dark gray areas.

(i.e, (a_{i+1}, b_{j+1})), a deletion pair (i.e., $(a_{i+1}, -)$) and an insertion pair (i.e., $(-, b_{j+1})$), respectively. Let $\mathcal{C}_k(h) = (C_1, C_2, \ldots, C_{k-1}, C_{k,h})$ and $\overline{\mathcal{C}}_k(h) = (\overline{C}_{k,h}, C_{k+1}, \ldots, C_\gamma)$, where $\overline{C}_{k,h} = c_{k,h+1}c_{k,h+2} \ldots c_{k,\lambda_k}$. Let $\overline{\mathcal{N}}_k(i,j,h)$ denote the score of an optimal semi-constrained alignment $\overline{\mathcal{L}}$ of $\overline{A}_i$ and $\overline{B}_j$ w.r.t. $\overline{\mathcal{C}}_k(h)$ that begins with a band whose induced consensus is equal to $\overline{C}_{k,h}$. Note that the recurrences for computing matrices $\overline{\mathcal{M}}_k, \overline{\mathcal{M}}_k^S, \overline{\mathcal{M}}_k^D$, $\overline{\mathcal{M}}_k^I$ and $\overline{\mathcal{N}}_k$ can be developed similarly as those for computing $\mathcal{M}_k$, $\mathcal{M}_k^S$, $\mathcal{M}_k^D$, $\mathcal{M}_k^I$ and $\mathcal{N}_k$, respectively. Clearly, $\mathcal{M}_k^S(i,j) = \mathcal{M}_k(i-1,j-1)+\sigma(a_i,b_j)$. For simplicity, let $A_i(h)$ (respectively, $B_j(h)$) denote the suffix of A_i (respectively, B_j) with length of h (i.e., $A_i(h) = a_{i-h+1}\ldots a_i$ and $B_j(h) = b_{j-h+1}\ldots b_j$). If $d_H(A_i(\lambda_k), C_k) \leq \lambda_k \times \epsilon$ and $d_H(B_j(\lambda_k), C_k) \leq \lambda_k \times \epsilon$, then we can reformulate the recurrence of $\mathcal{N}_k$ as follows: $\mathcal{N}_k(i,j,1) = \mathcal{M}_{k-1}(i-1,j-1)+\sigma(a_i,b_j)$ and $\mathcal{N}_k(i,j,h) = \mathcal{N}_k(i-1,j-1,h-1)+\sigma(a_i,b_j)$ for each $1 < h \leq \lambda_k$.

Next, we describe our divide-and-conquer algorithm, called as CPSA-DC algorithm, for computing an optimal constrained alignment between A and B w.r.t. $\mathcal{C}$ as follows. The key point is to determine the middle position $(i_{mid}, j_{mid}, k_{mid})$ of the optimal path in $\mathcal{G}$ to divide the problem into two subproblems, each of which is recursively divided into two smaller subproblems using the same way. Given an alignment $\mathcal{L}$, we use

$\texttt{score}(\mathcal{L})$ to denote the score of $\mathcal{L}$. Let $\mathcal{L}_\gamma(A,B)$ be an optimal constrained alignments of A and B w.r.t. $\mathcal{C}$ and clearly $\texttt{score}(\mathcal{L}_\gamma) = \mathcal{M}_\gamma(m,n)$. Let $i_{mid} = \lfloor \frac{m}{2} \rfloor$. Then we partition $\mathcal{L}_\gamma(A,B)$ into two parts by cutting it at the position immediately after $a_{i_{mid}}$ and we let $\mathcal{L}^1_\gamma(A,B)$ denote the part containing $a_{i_{mid}}$ and $\mathcal{L}^2_\gamma(A,B)$ denote the remaining part. Let $b_{j_{mid}}$ be the last character in $\mathcal{L}^1_\gamma(A,B)$ from B, and let k_{mid} be the largest index so that a prefix of $C_{k_{mid}}$ with length h_{mid} appears in $\mathcal{L}^1_\gamma(A,B)$. Then there are two possibilities when we consider the last aligned pair of $\mathcal{L}^1_\gamma(A,B)$.

Case 1: The last aligned pair of $\mathcal{L}^1_\gamma(A,B)$ is a substitution pair (i.e., $(a_{i_{mid}}, b_{j_{mid}})$). In this case, we have $\mathcal{M}_\gamma(m,n) = \texttt{score}(\mathcal{L}_\gamma(A,B)) = \texttt{score}(\mathcal{L}^1_\gamma(A,B)) + \texttt{score}(\mathcal{L}^2_\gamma(A,B))$. If $(a_{i_{mid}}, b_{j_{mid}})$ is not a constrained column in $\mathcal{L}_\gamma(A,B)$, then $\mathcal{L}^1_\gamma(A,B)$ is an optimal constrained alignment of $A_{i_{mid}}$ and $B_{j_{mid}}$ w.r.t. $\mathcal{C}_{k_{mid}}$ ending with a substitution pair $(a_{i_{mid}}, b_{j_{mid}})$, and $\mathcal{L}^2_\gamma(A,B)$ is an optimal constrained alignment of $\overline{A}_{i_{mid}}$ and $\overline{B}_{j_{mid}}$ w.r.t. $\overline{\mathcal{C}}_k$. Hence, $\mathcal{M}_\gamma(m,n) = \mathcal{M}^S_{k_{mid}}(i_{mid}, j_{mid}) + \overline{\mathcal{M}}_{k_{mid}}(i_{mid}, j_{mid})$. If $(a_{i_{mid}}, b_{j_{mid}})$ is a constrained column in $\mathcal{L}_\gamma(A,B)$, then $\mathcal{L}^1_\gamma(A,B)$ is an optimal semi-constrained alignment of $A_{i_{mid}}$ and $B_{j_{mid}}$ w.r.t. $\mathcal{C}_{k_{mid}}(h_{mid})$ ending with a band $\mathcal{B}_1$ whose induced consensus is equal to $C_{k_{mid},h_{mid}}$. If $h_{mid} < \lambda_{k_{mid}}$, then $\mathcal{L}^2_\gamma(A,B)$ is an optimal semi-constrained alignment of $\overline{A}_{i_{mid}}$ and $\overline{B}_{j_{mid}}$ w.r.t. $\overline{\mathcal{C}}_{k_{mid}}(h_{mid})$ beginning with a band $\mathcal{B}_2$ whose induced consensus is equal to $\overline{C}_{k_{mid},h_{mid}}$. Moreover, the induced consensus of the merge of $\mathcal{B}_1$ and $\mathcal{B}_2$ have to be equal to $C_{k_{mid}}$. In this case, we have $\mathcal{M}_\gamma(m,n) = \mathcal{N}_{k_{mid}}(i_{mid}, j_{mid}, h_{mid}) + \overline{\mathcal{N}}_{k_{mid}}(i_{mid}, j_{mid}, h_{mid})$. If $h_{mid} = \lambda_{k_{mid}}$, then $\mathcal{L}^2_\gamma(A,B)$ is an optimal constrained alignment of $\overline{A}_{i_{mid}}$ and $\overline{B}_{j_{mid}}$ w.r.t. $\overline{\mathcal{C}}_{k_{mid}}(h_{mid})$, and hence $\mathcal{M}_\gamma(m,n) = \mathcal{N}_{k_{mid}}(i_{mid}, j_{mid}, \lambda_{k_{mid}}) + \overline{\mathcal{M}}_{k_{mid}}(i_{mid}, j_{mid})$.

Case 2: The last aligned pair of $\mathcal{L}^1_\gamma(A,B)$ is a deletion pair (i.e., $(a_{i_{mid}}, -)$). If the first aligned pair in $\mathcal{L}^2_\gamma(A,B)$ is not a deletion pair, then $\mathcal{M}_\gamma(m,n) = \max\{\mathcal{M}^D_{k_{mid}}(i_{mid}, j_{mid}) + \overline{\mathcal{M}}^S_{k_{mid}}(i_{mid}, j_{mid}), \mathcal{M}^D_{k_{mid}}(i_{mid}, j_{mid}) + \overline{\mathcal{M}}^I_{k_{mid}}(i_{mid}, j_{mid})\}$. If the first aligned pair in $\mathcal{L}^2_\gamma(A,B)$ is a deletion pair, then $\mathcal{M}_\gamma(m,n) = \mathcal{M}^D_{k_{mid}}(i_{mid}, j_{mid}) + \overline{\mathcal{M}}^D_{k_{mid}}(i_{mid}, j_{mid}) + w_o$. Since the open penalty of the gap containing $a_{i_{mid}}$ and $a_{i_{mid}+1}$ in $\mathcal{L}_\gamma(A,B)$ is charged twice by $\mathcal{M}^D_{k_{mid}}(i_{mid}, j_{mid})$ and $\overline{\mathcal{M}}^D_{k_{mid}}(i_{mid}, j_{mid})$, we need to compensate it by adding w_o.

In summary, the recurrence of $\mathcal{M}_\gamma(m,n)$ is derived as follows.

$$\mathcal{M}_\gamma(m,n) = \max \left\{ \begin{array}{l} \mathcal{M}^D_{k_{mid}}(i_{mid}, j_{mid}) + \overline{\mathcal{M}}^S_{k_{mid}}(i_{mid}, j_{mid}), \\ \mathcal{M}^D_{k_{mid}}(i_{mid}, j_{mid}) + \overline{\mathcal{M}}^I_{k_{mid}}(i_{mid}, j_{mid}), \\ \mathcal{M}^D_{k_{mid}}(i_{mid}, j_{mid}) + \overline{\mathcal{M}}^D_{k_{mid}}(i_{mid}, j_{mid}) + w_o, \\ \mathcal{M}^S_{k_{mid}}(i_{mid}, j_{mid}) + \overline{\mathcal{M}}_{k_{mid}}(i_{mid}, j_{mid}), \\ \mathcal{N}_{k_{mid}}(i_{mid}, j_{mid}, h_{mid}) + \overline{\mathcal{N}}_{k_{mid}}(i_{mid}, j_{mid}, h_{mid}), \\ \mathcal{N}_{k_{mid}}(i_{mid}, j_{mid}, \lambda_{k_{mid}}) + \overline{\mathcal{M}}_{k_{mid}}(i_{mid}, j_{mid}) \end{array} \right\}$$

By adding extra $\mathcal{M}^D_{k_{mid}}(i_{mid}, j_{mid}) + \overline{\mathcal{M}}^D_{k_{mid}}(i_{mid}, j_{mid})$ into the right-hand side, the above recurrence is not changed, but can be reformulated as follows.

$$\mathcal{M}_\gamma(m,n) = \max \left\{ \begin{array}{l} \mathcal{M}^D_{k_{mid}}(i_{mid}, j_{mid}) + \overline{\mathcal{M}}_{k_{mid}}(i_{mid}, j_{mid}), \\ \mathcal{M}^D_{k_{mid}}(i_{mid}, j_{mid}) + \overline{\mathcal{M}}^D_{k_{mid}}(i_{mid}, j_{mid}) + w_o, \\ \mathcal{M}^S_{k_{mid}}(i_{mid}, j_{mid}) + \overline{\mathcal{M}}_{k_{mid}}(i_{mid}, j_{mid}), \\ \mathcal{N}_{k_{mid}}(i_{mid}, j_{mid}, h_{mid}) + \overline{\mathcal{N}}_{k_{mid}}(i_{mid}, j_{mid}, h_{mid}), \\ \mathcal{N}_{k_{mid}}(i_{mid}, j_{mid}, \lambda_{k_{mid}}) + \overline{\mathcal{M}}_{k_{mid}}(i_{mid}, j_{mid}) \end{array} \right\}$$

In other words, j_{mid}, k_{mid} and h_{mid} are the indices j, k and h, where $1 \leq j \leq n$, $0 \leq k \leq \gamma$ and $1 \leq h < \lambda_k$, such that the following maximal value is the maximum.

$$\max \left\{ \begin{array}{l} \mathcal{M}^D_k(i_{mid}, j) + \overline{\mathcal{M}}_k(i_{mid}, j), \\ \mathcal{M}^D_k(i_{mid}, j) + \overline{\mathcal{M}}^D_k(i_{mid}, j) + w_o, \\ \mathcal{M}^S_k(i_{mid}, j) + \overline{\mathcal{M}}_k(i_{mid}, j), \\ \mathcal{N}_k(i_{mid}, j, h) + \overline{\mathcal{N}}_k(i_{mid}, j, h), \\ \mathcal{N}_k(i_{mid}, j, \lambda_k) + \overline{\mathcal{M}}_k(i_{mid}, j) \end{array} \right\}$$

Now, we show how to use $\mathcal{O}(\gamma\lambda n)$, instead of $\mathcal{O}(\gamma m n)$, memory to determine j_{mid}, k_{mid} and h_{mid}, where $\lambda = \max_{1 \leq k \leq \gamma} \lambda_k$ and usually $\lambda << m$. In fact, a single matrix E of size $(\gamma+1) \times (n+1)$ with each entry $E(k,j)$ of $(\lambda+4)$ space is enough to compute $\mathcal{M}_k(i_{mid}, j)$, $\mathcal{M}^S_k(i_{mid}, j)$, $\mathcal{M}^D_k(i_{mid}, j)$ $\mathcal{M}^I_k(i_{mid}, j)$ and $\mathcal{N}_k(i_{mid}, j, h)$, where $1 \leq j \leq n$, $0 \leq k \leq \gamma$, $1 \leq h \leq \lambda_k$. When reaching the entry (i, j, k) of 3D grid graph $\mathcal{G}$, we use entry $E(k, j)$ of E to hold the most recently computed values of $\mathcal{M}_k(i,j)$, $\mathcal{M}^S_k(i,j)$, $\mathcal{M}^D_k(i,j)$ $\mathcal{M}^I_k(i,j)$ and $\mathcal{N}_k(i,j,h)$, which clearly needs a total of $\lambda_k + 4$ space. Note that the old values in entry $E(k,j)$ will be moved into an extra entry, called as V_k

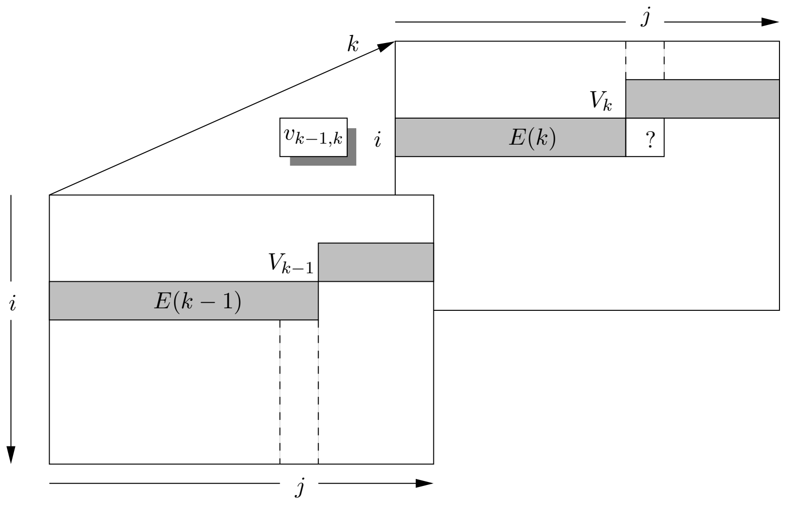


Figure 3.3: The grid locations of $E(k-1)$, $E(k)$ and the values in V_{k-1} and V_k when the entry (i, j, k) of $\mathcal{G}$, marked with "?", is reached for the computation.

whose space is equal to $E(k, j)$, before they are overwritten by their newly computed values. Before moving the old values in $E(k, j)$ into V_k, however, we need to first move $\mathcal{M}_k(i-1, j-1)$ in V_k into a space, called as $v_{k,k+1}$. The mechanism above will enable us to compute $\mathcal{N}_k(i, j, 1)$, which needs to refer to $\mathcal{M}_{k-1}(i-1, j-1)$ that is kept in $v_{k-1,k}$, and compute $\mathcal{N}_k(i, j, h)$ for each $2 \leq h \leq \lambda_k$, which needs to refer to $\mathcal{N}_k(i-1, j-1, h-1)$ that is kept in V_k, compute $\mathcal{M}_k^S(i, j)$ which needs to refer $\mathcal{M}_k(i-1, j-1)$ that is kept in V_k, and finally we are able to compute $\mathcal{M}_k(i, j)$. Figure 3.3 shows the grid locations of $E(k-1)$, $E(k)$ and the values in V_{k-1} and V_k when we reach the entry (i, j, k) of $\mathcal{G}$ for the computation, where $E(k)$ denotes the the kth row of E. Hence, the totally needed space for computing and storing all $\mathcal{M}_k(i_{mid}, j)$, $\mathcal{M}_k^S(i_{mid}, j)$, $\mathcal{M}_k^D(i_{mid}, j)$ $\mathcal{M}_k^I(i_{mid}, j)$ and $\mathcal{N}_k(i_{mid}, j, h)$ is the sum of the space of matrix E, the space of all V_k, $0 \leq k \leq \gamma$, and the space of all $v_{k,k+1}$, $0 \leq k < \gamma$, which is equal to $\mathcal{O}(\gamma\lambda n)$. Similarly, the process of computing and storing all $\overline{\mathcal{M}}_k(i_{mid}, j)$, $\overline{\mathcal{M}}_k^S(i_{mid}, j)$, $\overline{\mathcal{M}}_k^D(i_{mid}, j)$ $\overline{M}_k^I(i_{mid}, j)$ and $\overline{\mathcal{N}}_k(i_{mid}, j, h)$ still needs $\mathcal{O}(\gamma\lambda n)$ space. Hence, the determination of j_{mid}, k_{mid} and h_{mid} can be done in $\mathcal{O}(\gamma\lambda n)$ space. The details of CPSA-DC algorithm are described as follows, where the program code of BestScoreRev is similar to that of BestScore and hence is omitted.

```
Algorithm CPSA-DC($i_{start}, i_{end}, j_{start}, j_{end}, k_{start}, k_{end}$)
Input: Sequences $a_{i_{start}} \dots a_{i_{end}}$ and $b_{j_{start}} \dots b_{j_{end}}$ with constraints $(C_{k_{start}}, \dots, C_{k_{end}})$
Step 1: if $(i_{start} > i_{end})$ or $(j_{start} > j_{end})$ then
            Align the nonempty sequence with spaces;
        else
            $i_{mid} = \lfloor \frac{i_{start}+i_{end}}{2} \rfloor$;
            BestScore($i_{start}, i_{mid}, j_{start}, j_{end}, k_{start}, k_{end}$);
            BestScoreRev($i_{mid}+1, i_{end}, j_{start}, j_{end}, k_{start}, k_{end}$);
        end if
Step 2: $max = -\infty$;
        for $j = j_{start}-1$ to $j_{end}$ do
            for $k = k_{start}-1$ to $k_{end}$ do
                if $\mathcal{M}^D_k(\cdot, j) + \overline{\mathcal{M}}_k(\cdot, j) > max$ then
                    $max = \mathcal{M}^D_k(\cdot, j) + \overline{\mathcal{M}}_k(\cdot, j)$;
                    $j_{mid} = j$; $k_{mid} = k$; $type = case\ 1$;
                end if
                if $\mathcal{M}^D_k(\cdot, j) + \overline{\mathcal{M}}^D_k(\cdot, j) > max$ then
                    $max = \mathcal{M}^D_k(\cdot, j) + \overline{\mathcal{M}}^D_k(\cdot, j)$;
                    $j_{mid} = j$; $k_{mid} = k$; $type = case\ 2$;
                end if
                if $\mathcal{M}^S_k(\cdot, j) + \overline{\mathcal{M}}_k(\cdot, j) > max$ then
                    $max = \mathcal{M}^S_k(\cdot, j) + \overline{\mathcal{M}}_k(\cdot, j)$;
                    $j_{mid} = j$; $k_{mid} = k$; $type = case\ 3$;
                end if
                if $k \geq 1$ then
                    for $h = 1$ to $\lambda_k - 1$ do
                        if $(\frac{\mathcal{H}_1(k,h)+\overline{\mathcal{H}}_1(k,h)}{\lambda_k} \leq \epsilon)$ and $(\frac{\mathcal{H}_2(k,h)+\overline{\mathcal{H}}_2(k,h)}{\lambda_k} \leq \epsilon)$ then
                            if $\mathcal{N}_k(\cdot, j, h) + \overline{\mathcal{N}}_k(\cdot, j, h) > max$ then
                                $max = \mathcal{N}_k(\cdot, j, h) + \overline{\mathcal{N}}_k(\cdot, j, h)$;
                                $j_{mid} = j$; $k_{mid} = k$; $h_{mid} = h$; $type = case\ 4$;
                end if
            end if
        end for
        if $(\frac{\mathcal{H}_1(k,\lambda_k)}{\lambda_k} \le \epsilon)$ and $(\frac{\mathcal{H}_2(k,h)}{\lambda_k} \le \epsilon)$ then
            if $\mathcal{N}_k(\cdot, j, \lambda_k) + \overline{\mathcal{M}}_k(\cdot, j) > max$ then
                $max = \mathcal{N}_k(\cdot, j, \lambda_k) + \overline{\mathcal{M}}_k(\cdot, j)$;
                $j_{mid} = j$; $k_{mid} = k$; $h_{mid} = h$; $type = case\ 5$;
            end if
        end if
      end if
    end for
  end for
Step 3: if $type = case\ 1$ then
    CPSA-DC($i_{start}, i_{mid} - 1, j_{start}, j_{mid}, k_{start}, k_{mid}$);
    Align $a_{i_{mid}}$ with a space;
    CPSA-DC($i_{mid} + 1, i_{end}, j_{mid} + 1, j_{end}, k_{mid} + 1, k_{end}$);
  end if
  if $type = case\ 2$ then
    CPSA-DC($i_{start}, i_{mid} - 1, j_{start}, j_{mid}, k_{start}, k_{mid}$);
    Align $a_{i_{mid}} a_{i_{mid}+1}$ with two spaces;
    CPSA-DC($i_{mid} + 2, i_{end}, j_{mid} + 1, j_{end}, k_{mid} + 1, k_{end}$);
  end if
  if $type = case\ 3$ then
    CPSA-DC($i_{start}, i_{mid} - 1, j_{start}, j_{mid} - 1, k_{start}, k_{mid}$);
    Align $a_{i_{mid}}$ with $b_{j_{mid}}$;
    CPSA-DC($i_{mid} + 1, i_{end}, j_{mid} + 1, j_{end}, k_{mid} + 1, k_{end}$);
  end if
  if $type = case\ 4$ then
    CPSA-DC($i_{start}, i_{mid} - h_{mid}, j_{start}, j_{mid} - h_{mid}, k_{start}, k_{mid} - 1$);
    Align $a_{i_{mid}-h_{mid}+1} \ldots a_{i_{mid}+\lambda_k-h_{mid}}$ with $b_{j_{mid}-h_{mid}+1} \ldots b_{j_{mid}+\lambda_k-h_{mid}}$;
```

```
        CPSA-DC($i_{mid} + \lambda_k - h_{mid} + 1, i_{end}, j_{mid} + \lambda_k - h_{mid} + 1, j_{end}, k_{mid} + 1, k_{end}$);
    end if
    if $type = case\ 5$ then
        CPSA-DC($i_{start}, i_{mid} - \lambda_k, j_{start}, j_{mid} - \lambda_k, k_{start}, k_{mid} - 1$);
        Align $a_{i_{mid}-\lambda_k+1} \ldots a_{i_{mid}}$ with $b_{j_{mid}-\lambda+1} \ldots b_{j_{mid}}$;
        CPSA-DC($i_{mid} + 1, i_{end}, j_{mid} + 1, j_{end}, k_{mid} + 1, k_{end}$);
    end if
```

```
Algorithm BestScore($i_{start}, i_{end}, j_{start}, j_{end}, k_{start}, k_{end}$)
Input: Sequences $a_{i_{start}} \ldots a_{i_{end}}$ and $b_{j_{start}} \ldots b_{j_{end}}$ with constraints $(C_{k_{start}}, \ldots, C_{k_{end}})$
Output:
Step 1: /* Reindex */
    $m = i_{start} - i_{end} + 1$; $n = j_{start} - j_{end} + 1$; $\gamma = k_{start} - k_{end} + 1$;
Step 2: /* Initialization */
    for $j = 0$ to $n$ do
        for $k = 0$ to $\gamma$ do
            if $(j = 0)$ and $(k = 0)$ then $\mathcal{M}_k(\cdot, j) = 0$; else $\mathcal{M}_k(\cdot, j) = -\infty$;
            if $(j = 0)$ or $(k > 0)$ then $\mathcal{M}_k^I(\cdot, j) = -\infty$; else $\mathcal{M}_k^I(\cdot, j) = -w_o - jw_e$;
            $\mathcal{M}_k^S(\cdot, j) = \mathcal{M}_k^D(\cdot, j) = -\infty$;
            if $k \geq 1$ then
                for $h = 1$ to $\lambda_k$ do
                    $\mathcal{N}_k(\cdot, j, h) = -\infty$;
                end for
            end if
        end for
    end for
Step 3: /* Computation */
    for $i = 1$ to $m$ do
        for $k = 0$ to $\gamma$ do /* For the case of $j = 0$ */
        $V_k(\mathcal{M}_k(\cdot,0)) = \mathcal{M}_k(\cdot,0)$;
        **if** $k \geq 1$ **then**
            **for** $h = 1$ to $\lambda_k$ **do**
                $V_k(\mathcal{N}_k(\cdot,0,h)) = \mathcal{N}_k(\cdot,0,h))$;
                $V_k(\mathcal{H}_1(k,h)) = \mathcal{H}_1(k,h)$;
                $V_k(\mathcal{H}_2(k,h)) = \mathcal{H}_2(k,h)$;
            **end for**
        **end if**
        $\mathcal{M}^S_k(\cdot,0) = \mathcal{M}^I_k(\cdot,0) = -\infty$;
        $\mathcal{M}_k(\cdot,0) = \mathcal{M}^D_k(\cdot,0) = -w_o - jw_e$;
    **end for**
    **for** $j = 1$ to $n$ **do** /* **For the case of** $j > 0$ */
        **for** $k = 0$ to $\gamma$ **do**
            $temp_k(\mathcal{M}_k(\cdot,j)) = \mathcal{M}_k(\cdot,j)$ ;
            **if** $k \geq 1$ **then**
                **for** $h = 1$ to $\lambda_k$ **do**
                    $temp_k(\mathcal{N}_k(\cdot,j,h)) = \mathcal{N}_k(\cdot,j,h)$;
                    $temp_k(\mathcal{H}_1(k,h)) = \mathcal{H}_1(k,h)$;
                    $temp_k(\mathcal{H}_2(k,h)) = \mathcal{H}_2(k,h)$;
                **end for**
            **end if**
            $\mathcal{M}^S_k(\cdot,j) = V(\mathcal{M}_k(\cdot,j)) + \sigma(a_{i_{start}+i-1}, b_{j_{start}+j-1})$;
            $\mathcal{M}^D_k(\cdot,j) = \max\{\mathcal{M}^D_k(\cdot,j) - w_e, \mathcal{M}_k(\cdot,j) - w_o - w_e\}$;
            $\mathcal{M}^I_k(\cdot,j) = \max\{\mathcal{M}^I_k(\cdot,j-1) - w_e, \mathcal{M}_k(\cdot,j-1) - w_o - w_e\}$;
            **if** $k \geq 1$ **then**
                **for** $h = 1$ to $\lambda_k$ **do**
                    **if** $h = 1$ **then**
                        $\mathcal{N}_k(\cdot,j,h) = v_{k-1,k} + \sigma(a_{i_{start}+i-1}, b_{j_{start}+j-1})$;
                        **if** $a_{i_{start}+i-1} \neq c_{k,h}$ **then** $\mathcal{H}_1(k,h) = 1$; **else** $\mathcal{H}_1(k,h) = 0$;
                        **if** $b_{j_{start}+j-1} \neq c_{k,h}$ **then** $\mathcal{H}_2(k,h) = 1$; **else** $\mathcal{H}_2(k,h) = 0$;
                    else
                        $\mathcal{N}_k(\cdot, j, h) = V_k(\mathcal{N}_k(\cdot, j, h-1)) + \sigma(a_{i_{start}+i-1}, b_{j_{start}+j-1})$;
                        if $a_{i_{start}+i-1} \neq c_{k,h}$ then $\mathcal{H}_1(k, h) = \mathcal{H}_1(k, h-1) + 1$;
                        if $b_{j_{start}+j-1} \neq c_{k,h}$ then $\mathcal{H}_2(k, h) = \mathcal{H}_2(k, h-1) + 1$;
                    end if
                end for
            end if
            $\mathcal{M}_k(\cdot, j) = \max \left\{ \begin{array}{l} \mathcal{M}_k^D(\cdot, j), \mathcal{M}_k^I(\cdot, j), \mathcal{N}_k(\cdot, j, \lambda_k) \\ V_k(\mathcal{M}_k(\cdot, j)) + \sigma(a_{i_{start}+i-1}, b_{j_{start}+j-1}), \end{array} \right\}$;
            $v_{k,k+1} = V_k(M_k(\cdot, j))$;
            $V_k(\mathcal{M}_k(\cdot, j)) = temp_k(\mathcal{M}_k(\cdot, j))$;
            if $k \geq 1$ then
                for $h = 1$ to $\lambda_k$ do
                    $V_k(\mathcal{N}(\cdot, j, h)) = temp_k(\mathcal{N}_k(\cdot, j, h))$;
                    $\mathcal{H}_1(k, h) = temp_k(\mathcal{H}_1(k, h))$;
                    $\mathcal{H}_2(k, h) = temp_k(\mathcal{H}_2(k, h))$;
                end for
            end if
        end for
    end for
```

Now, we analyze the time complexity of our CPSA-DC algorithm for solving the constrained pairwise sequence alignment. As illustrated in Figure 3.2, after determining the middle position $(i_{mid}, j_{mid}, k_{mid})$ of the optimal path in $\mathcal{G}$, we can divide the original problem into two subproblems, each of which further can be recursively divided into two smaller subproblems using the same way. Note that regardless of where the optimal path passes through $(i_{mid}, j_{mid}, k_{mid})$, the total size of the two reduced subproblems is just half the size of the original problem, where the size is measured by the number of the entries in $\mathcal{G}$. In is not hard to see that the time complexity of determining the middle position of each subproblem at each recursive stage is proportional to the size of the subproblem. Let T denote the size of the original problem (i.e., $T = \gamma mn$). Then the total time complexity of our CPSA-DC algorithm is equal to $T + \frac{T}{2} + \frac{T}{4} + \cdots = 2T$, which is twice as high as the CPSA-DP algorithm.

3.2 Constrained multiple sequence alignment

In this section, we use Algorithm CPSA-DP and CPSA-DC in previous section as kernels to design two CMSA algorithms, called Algorithm CMSA-DP and CMSA-DC respectively, for progressively aligning the input sequences into a CMSA according to the branching order of a guide tree, where the guide tree we use here is the so-called Kruskal merging order tree. We refer the reader to Section 2 for the details of constructing the Kruskal merging order tree. Except for the adopted CPSA kernel, CMSA-DP and CMSA-DC have the same execution steps, which are described as follows.

1. Compute the distance matrix D by globally aligning all pairs of sequences using Algorithm CPSA-DP or CPSA-DC , where $D(i, j)$ denotes the distance between sequences S_i and S_j.

2. Create a complete graph G from the distance matrix D and then compute the Kruskal merging order tree $\mathcal{T}_k$ from G to serve as the guide tree.

3. Progressively align the sequences according to the branching order of the guide tree $\mathcal{T}_k$ in a way that the currently two closest pre-aligned groups of sequences are joined by applying Algorithm CPSA-DP or CPSA-DC to these two groups of sequences, where the score between any two positions in these two groups is the arithmetic average of the scores for all possible character comparisons at those positions.

In the following, we analyze the time complexity of Algorithm CMSA-DP/CMSA-DC. It is not hard to see that step 1 costs $\mathcal{O}(\gamma k^2 n^2)$ time, where n is the maximum of the lengths of k sequences. According to the paper of Tang *et al.* [34] , step 2 can be done in $\mathcal{O}(k^2 \log k)$ time. In step 3, there are at most $\mathcal{O}(k)$ iterations for calling

Algorithm CPSA-DP/CPSA-DC, whose time complexity is $\mathcal{O}(\gamma n^2)$, to join two pre-aligned groups of sequences. Hence, the time complexity of step 3 is $\mathcal{O}(\gamma k n^2)$. Clearly, the cost of Algorithm CMSA-DP/CMSA-DC is dominated by step 1 and hence its time complexity is $\mathcal{O}(\gamma k^2 n^2)$.

Chapter 4

Implementation and Discussion

4.1 MuSiC

We use Java language to implement the CMSA-DP algorithm as a web server, called as MuSiC, which is a short for Multiple Sequence Alignment with Constraints. It can be easily accessed via a simple web interface (see Figure 4.1). The input of the MuSiC system consists of a set of protein/DNA/RNA sequences and a set of user-specified constraints, each with a fragment of bases that (approximately) appears in all input sequences. The output of MuSiC is a constrained multiple sequence alignment in which the fragments of the input sequences whose bases exhibit a given degree of similarity to a constraint are aligned together. The use of the proposed MuSiC system is illustrated below to help to detect a fragment of an RNA sequence in the $3'$ untranslated region (UTR) of the SARS-TW1 sequence, which can fold itself into a pseudoknot structure. The structural elements in the $5'$ and $3'$ UTRs of a plus-strand RNA virus have been postulated to be involved in RNA replication, transcription and translation by interacting with viral or cellular proteins. Much evidence supports the fact that the pseudoknots in $3'$ UTRs among coronaviruses participate in the replication of RNA [43]. The SARS (Severe Acute Respiratory Syndrome) virus, which caused several hundreds of deaths since its outbreak in early 2003, is a novel type of coronavirus, so a pseudoknot is expected to be observed in its $3'$ UTR. By comparing the sequences of the phylogenetically conserved pseudoknots among many coronaviruses,

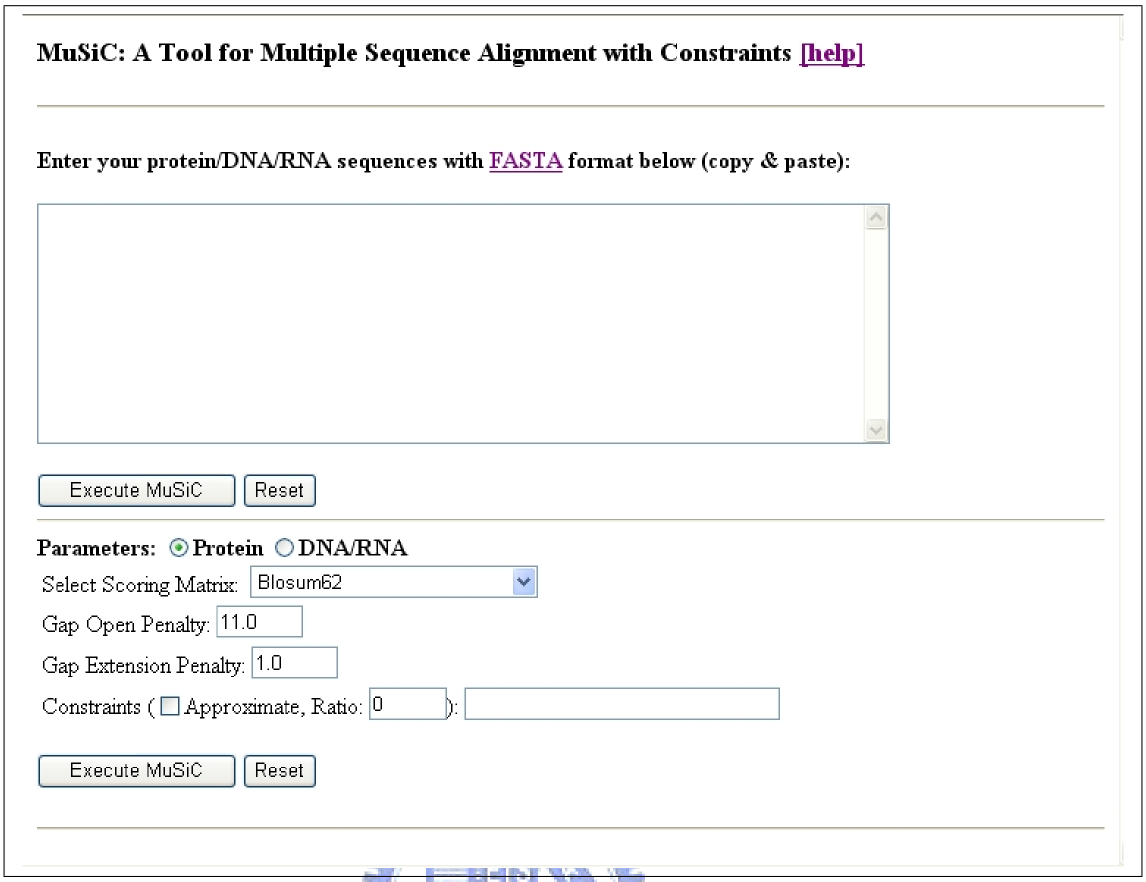
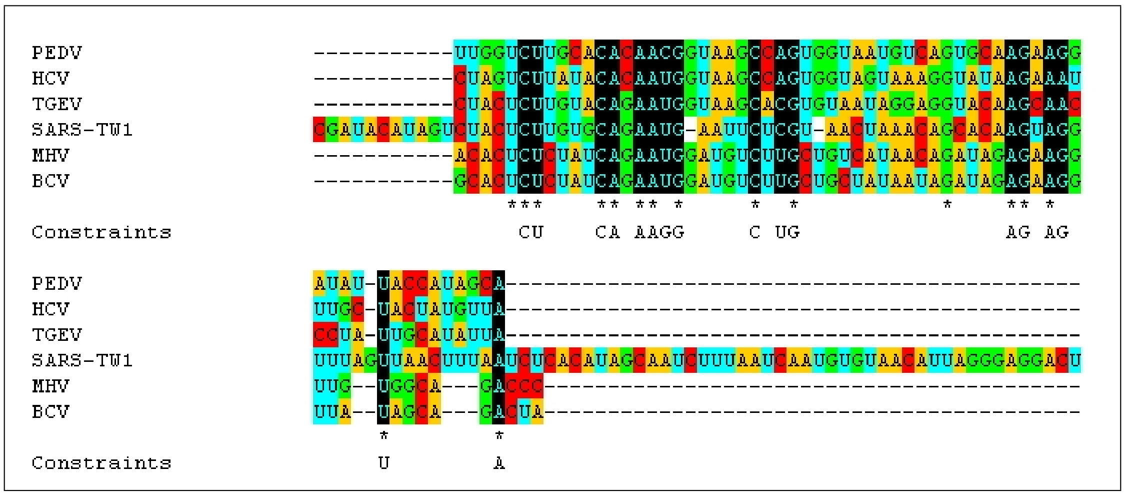

MuSiC: A Tool for Multiple Sequence Alignment with Constraints [help]
Enter your protein/DNA/RNA sequences with FASTA format below (copy & paste):
Execute MuSiC
Reset
Parameters: [✓ Protein] DNA/RNA
Select Scoring Matrix: Blosum62
Gap Open Penalty: 11.0
Gap Extension Penalty: 1.0
Constraints (Approximate, Ratio: 0):
Execute MuSiC
Reset

Figure 4.1: The interface of MuSiC.

Williams *et al.* found that these sequences contain several fragments of conserved nucleotides (consensuses). They found 12 consensuses, say `CU, CA, AA, GG, C, UG, A, G, AG, U` and `A`, among coronaviruses including HCV-229E (human coronavirus), PEDV (porcine epidemic diarrhea virus), TGEV (porcine transmissible gastroenteritis virus), BCV (bovine coronavirus) and MHV (mouse hepatitis virus). To determine whether or not the $3'$ UTR of SARS has a pseudoknot, SARS-TW1 (AY291451) was chosen as the test subject and the MuSiC system was used to align the sequence of the $3'$ UTR of SARS-TW1 with those of the detected pseudoknots in the $3'$ UTRs of BCV, MHV, PEDV, TGEV and HCV-229E coronaviruses. The consensuses described above were used as the constrained sequences in the proposed MuSiC system and then the default scoring matrix and gap penalties were chosen with the initial setting $\epsilon = 0$. As a result, no CMSA was found to satisfy the requirement, because the postulated pseudoknot in the $3'$ UTR of SARS-TW1 may comprise the fragments that are only partially, rather than completely, similar to the constraints. Hence, this



```
PEDV           -----------UUGGUCUUGCACACAACGGUAAGCCAGUGGUAAUGUCAGUGCAAGAAGG
HCV            -----------CUAGUCUUAUACACAAUGGUAAGCCAGUGGUAGUAAAGGUAUAAGAAAU
TGEV           -----------CUACUCUUGUACAGAAUGGUAAGCACGUGUAAUAGGAGGUACAAGCAAC
SARS-TW1       CGAUACAUAGUCUACUCUUGUGCAGAAUG-AAUUCUCGU-AACUAAAACAGCACAAGUAGG
MHV            -----------ACACUCUCUAUCAGAAUGGAUGUCUUGCUGUCAUAAACAGAUAGAGAAGG
BCV            -----------GCACUCUCUAUCAGAAUGGAUGUCUUGCUGCUAUAAUAGAUAGAGAAGG
                              ***    ** ** *     *  *           ** *
Constraints                    CU    CA AAGG     C UG                AG AG

PEDV           AUAU-UACCAUAGCA---------------------------------------------
HCV            UUGC-UACUAUGUUA---------------------------------------------
TGEV           CCUA-UUGCAUAUUA---------------------------------------------
SARS-TW1       UUUAGUUAACUUUAAUCUCACAUAGCAAUCUUUAAUCAAUGUGUAACAUUAGGGAGGACU
MHV            UUG--UGGCA---GACCC------------------------------------------
BCV            UUA--UAGCA---GACUA------------------------------------------
                    *        *
Constraints         U        A
```

Figure 4.2: The partial display of the resulting CMSA of MuSiC by aligning the sequences of SARS-TW1 3′ UTR with those of other five coronaviruses.

case was tested again with letting $\epsilon = 0.5$ so that in the band of the resulting CMSA, of length two or three, no more than one mismatch may exist between the fragment of each input sequence and the constraint. Consequently, as indicated in Figure 4.2 , a satisfied CMSA was found. Note that, the band of the resulting CMSA that corresponds to a constraint is black and its corresponding constraint is displayed beneath it. In some bands of this resulting CMSA, such as the fourth, sixth and ninth, at most one mismatch exists between the fragment of each input sequence and the corresponding constraint. Moreover, the part of SARS-TW1 aligned with the pseudoknot sequences of other coronaviruses is interspersed with only two gaps of length one. This finding suggests that this part of SARS-TW1 may fold itself into a pseudoknot structure and possibly be involved in replicating SARS viruses. Therefore, this SARS-TW1 fragment is further validated by applying PKNOTS, developed by the Eddy group [30], to determine whether it can fold itself into a pseudoknot structure with a stable free energy. Consequently, this fragment of SARS-TW1 indeed folds itself into a stable pseudoknot whose base pairings have a topology, as indicated in Figure 4.3 , that is very similar to those of other coronaviruses described in the literature [43]. However, whether or not this SARS-TW1 fragment participates in replicating the RNA of SARS must be investigated experimentally in the laboratory.

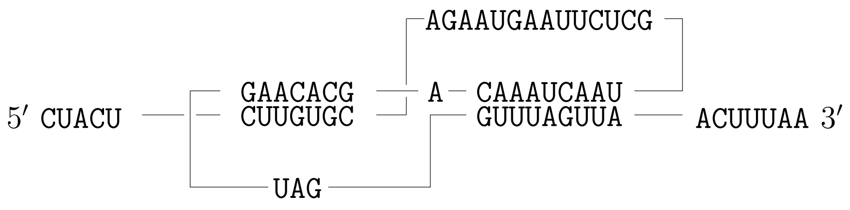


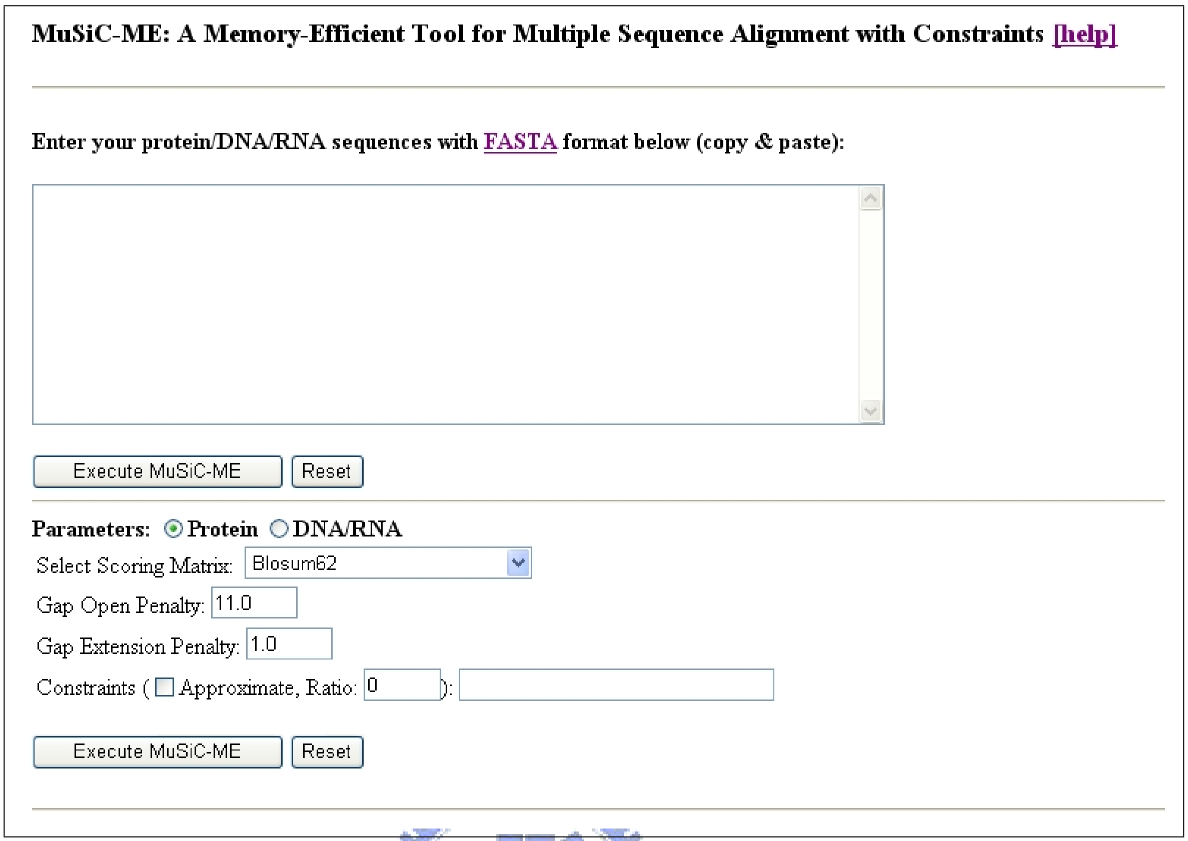
Figure 4.3: The diagram of the predicted pseudoknot in the 3′ UTR of SARS-TW1 ranging from 29460 to 29521 bp.

4.2 MuSiC-ME

We also use Java language to implement the CMSA-DC algorithm as a web server, called as MuSiC-ME that is short for Memory-Efficient tool for Multiple Sequence Alignment with Constraints (see Figure 4.4). It is worth mentioning that for MuSiC-ME, the letters representing the constraints are not just the individual bases, but also the IUPAC (International Union of Pure and Applied Chemistry) codes. For example, nucleotides N and R have the meanings of any bases and purine (i.e., A or G), respectively.

To demonstrate the practicability of our MuSiC-ME system, we also use it to detect a fragment of an RNA sequence in the 3′ UTR of the SARS-TW1 sequence being able to fold into a stable pseudoknot.

In this test, we aligned the sequence of the 3′ UTR of SARS-TW1 with those of the 3′ UTRs of BCV, MHV, PEDV, TGEV and HCV-229E coronaviruses, and used the consensuses as described before as the constraints. Since these constraints are too short, they occur frequently in the large genomic sequences. For our purpose, we further combine a few of constraints into a new and larger constraint as follows. Among the consensuses above, the first and second (respectively, ninth and tenth) consensuses are located in the 5′ (respectively, 3′) end of stem 1 and they are separated by other 4 (respectively, 4) bases, and the seventh and eighth (respectively, eleventh and twelfth) consensuses are located in the 5′ (respectively, 3′) end of stem 2 and they are separated by other 3 (respectively, 3) bases. Since both stems 1 and 2 contain no loops, we are able to combine the consensuses in the same stem into a new and larger constraint. Hence, we have new constraints like CUNNNNC for the 5′ end of stem 1, GNNNNAG for the


MuSiC-ME: A Memory-Efficient Tool for Multiple Sequence Alignment with Constraints [help]

Enter your protein/DNA/RNA sequences with FASTA format below (copy & paste):

Parameters: [✓ Protein] DNA/RNA

Select Scoring Matrix: Blosum62

Gap Open Penalty: 11.0

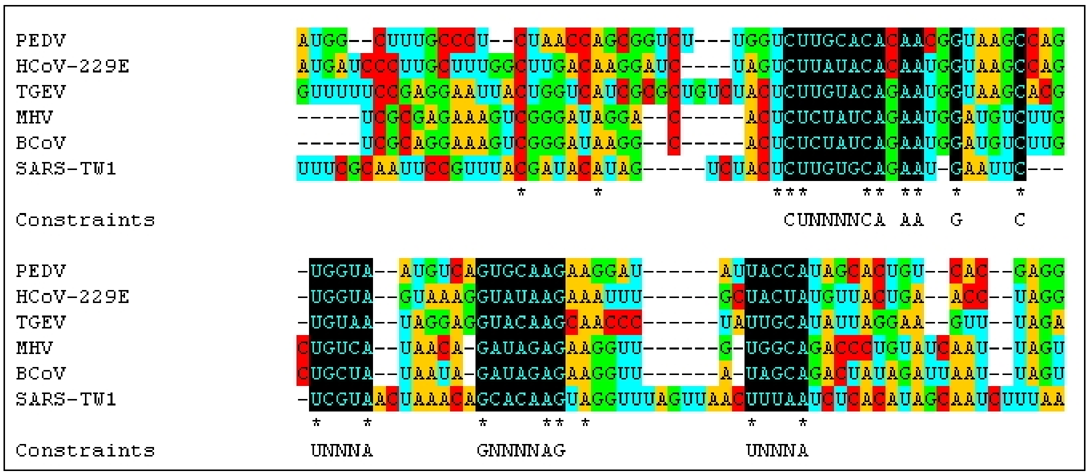
Gap Extension Penalty: 1.0

Constraints (Approximate, Ratio: 0):

Figure 4.4: The interface of MuSiC-ME.

$3'$ end of stem 1, `UNNNA` for the $5'$ end of stem 2, and `UNNNA` for the $3'$ end of stem 2. Finally, we got eight constraints with the order of (`CUNNNNC, A, AA, G, C, UNNNA, GNNNNAG, UNNNA`) for this test. After running MuSiC-ME, a satisfied CMSA was found as shown in Figure 4.5. This resulting CMSA implies that the fragment of SARS-TW1 between the first band and the last band may fold into a pseudoknot structure that is possibly involved in replicating SARS viruses. In fact, the fragment is the one we found in the test with MuSiC and hence it can fold into a stable pseudoknot as shown in Figure 4.3.

Note that above test was run on IBM PC with 1.26 GHz processor and 128 MB RAM under Linux system. Under this limited memory environment, the instance can not be executed by the MuSiC system whose kernel, the CPSA-DP algorithm, was implemented by the dynamic programming approach, due to running out of memory. The input sequences above were also tested by Clustal W 1.82, the most commonly used MSA tool. According to its resulting MSA as shown in Figure 4.6, the fragments

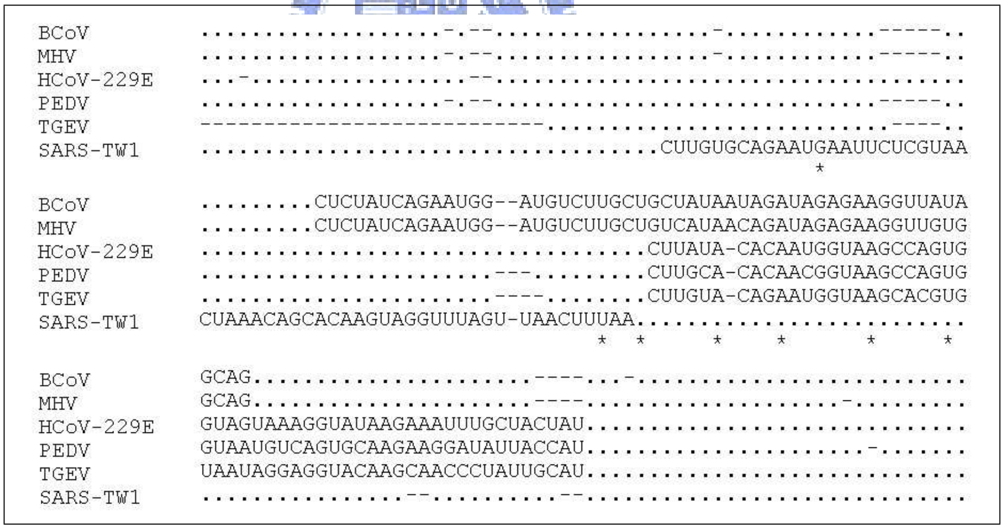


```
PEDV            AUGG--CUUUGCCCU--CUAACCAGCGGUCU---UGGUCUUGCACACAACGGUAAGCCAG
HCoV-229E       AUGAUCCCUUGCUUUGGCUUGACAAGGAUC----UAGUCUUAUACACAAUGGUAAGCCAG
TGEV            GUUUUUCCGAGGAAUUACUGGUCAUCGCGCUGUCUACUCUUGUACAGAAUGGUAAGCACG
MHV             -----UCGCGAGAAAGUCGGGAUAGGA--C-----ACUCUCUAUCAGAAUGGAUGUCUUG
BCoV            -----UCGCAGGAAAGUCGGGAUAAGG--C-----ACUCUCUAUCAGAAUGGAUGUCUUG
SARS-TW1        UUUCGCAAUUCCGUUUACGAUACAUAG-----UCUACUCUUGUGCAGAAU-GAAUUC---
                                 *     *             ***    ** **  *    *
Constraints                                           CUNNNNCA AA  G    C

PEDV            -UGGUA--AUGUCAGUGCAAGAAGGAU------AUUACCAUAGCACUGU--CAC--GAGG
HCoV-229E       -UGGUA--GUAAAGGUAUAAGAAAUUU------GCUACUAUGUUACUGA--ACC--UAGG
TGEV            -UGUAA--UAGGAGGUACAAGCAACCC------UAUUGCAUAUUAGGAA--GUU--UAGA
MHV             CUGUCA--UAACA-GAUAGAGAAGGUU------G-UGGCAGACCCUGUAUCAAU--UAGU
BCoV            CUGCUA--UAAUA-GAUAGAGAAGGUU------A-UAGCAGACUAUAGAUUAAU--UAGU
SARS-TW1        -UCGUAACUAAACAGCACAAGUAGGUUUAGUUAACUUUAAUCUCACAUAGCAAUCUUUAA
                 *   *        *    ** *            *   *
Constraints      UNNNA        GNNNNAG              UNNNA
```

Figure 4.5: The partial display of the resulting CMSA of MuSiC-ME by aligning the sequences of SARS-TW1 3′ UTR with those of other five coronaviruses.

of all pseudoknots, including our detected pseudoknot for SARS-TW1, are not able to be aligned well so that it is difficult for us to identify the exact fragment of the SARS-TW1 pseudoknot from this MSA.



```
BCoV        ...................-.--.................-............-----..
MHV         ...................-.--.................-............-----..
HCoV-229E   ...-.................--.....................................
PEDV        ...................-.--..............................-----..
TGEV        ---------------------------...........................----..
SARS-TW1    ....................................CUUGUGCAGAAUGAAUUCUCGUAA
                                                *

BCoV        .........CUCUAUCAGAAUGG--AUGUCUUGCUGCUAUAAUAGAUAGAGAAGGUUAUA
MHV         .........CUCUAUCAGAAUGG--AUGUCUUGCUGUCAUAACAGAUAGAGAAGGUUGUG
HCoV-229E   ...................................CUUAUA-CACAAUGGUAAGCCAGUG
PEDV        .......................---.........CUUGCA-CACAACGGUAAGCCAGUG
TGEV        .......................----........CUUGUA-CAGAAUGGUAAGCACGUG
SARS-TW1    CUAAACAGCACAAGUAGGUUUAGU-UAACUUUAA..........................
                                           *  *     *    *      *     *

BCoV        GCAG......................----...-..........................
MHV         GCAG......................----....................-.........
HCoV-229E   GUAGUAAAGGUAUAAGAAAUUUGCUACUAU..............................
PEDV        GUAAUGUCAGUGCAAGAAGGAUAUUACCAU......................-.......
TGEV        UAAUAGGAGGUACAAGCAACCCUAUUGCAU..............................
SARS-TW1    ................--..........--..............................
```

Figure 4.6: The partial display of the resulting MSA of Clustal W 1.82 by aligning the 3′ UTR sequences of six coronaviruses, where the bases not in the pseudoknots are marked with dots.

Chapter 5

Conclusions

In this thesis, we studied a generalized CMSA problem, which is to find a CMSA for the input sequences with several user-specified constraints such that the fragments of the input sequences whose bases exhibit a given degree of similarity to a constraint are aligned together. In this model, each of the user-specified constraint may be a fragment of bases, instead of a single base only, and the adopted gap scoring is the so-called affine gap penalty that penalizes a gap once for opening and then proportionally to its length.

First, we adopted the dynamic programming and divide-and-conquer techniques to design a time-efficient algorithm and a memory-efficient algorithm respectively for optimally solving the CPSA problem. Based on these two kernel algorithms, we developed two CMSA programs, called as MuSiC and MuSiC-ME respectively, using the progressive approach. The MuSiC program generates a good CMSA efficiently, but its high memory requirement limits it to align a set of short sequences, at most several hundreds of bases. The MuSiC-ME program made it possible to align several large-scale sequences with constraints through the desktop PC with the limited memory. In this system, moreover, the letters allowed to represent the constraints are the IUPAC codes, which will enable the user to define a more flexible constraints or combine several small constraints with fixed distances into a large one. The practicabilities of MuSiC and MuSiC-ME were also demonstrated by helping us to detect the fragment of the $3'$ UTR of SARS that is able to fold itself into a stable pseudoknot for possibly participating in replicating the RNA of SARS.

References

[1] Altschul, S. & Lipman, D. (1989) Trees stars and multiple biological sequence alignment. *SIAM Journal on Applied Mathematics,* **49**, 197–209.

[2] Bafna, V., Lawler, E. L. & Pevzner, P. A. (1997) Approximation algorithms for multiple sequence alignment. *Theoretical Computer Science,* **182**, 233–244.

[3] Bonizzoni, P. & Vedova, G. D. (2001) The complexity of multiple sequence alignment with SP-score that is a metric. *Theoretical Computer Science,* **259**, 63–79.

[4] Carrillo, H. & Lipman, D. (1988) The multiple sequence alignment problem in biology. *SIAM Journal on Applied Mathematics,* **48**, 1073–1082.

[5] Chan, S. C., Wong, A. K. C. & Chiu, D. K. Y. (1992) A survey of multiple sequence comparison methods. *Bulletin of Mathematical Biology,* **54**, 563–598.

[6] Chao, K. M., Hardison, R. C. & W, W. M. (1994) Recent developments in linear-space alignment methods: a survey. *Journal of Computational Biology,* **1**, 271–291.

[7] Chin, F. Y. L., Ho, N. L., Lamy, T. W., Wong, P. W. H. & Chan, M. Y. (2003) Efficient constrained multiple sequence alignment with performance guarantee. In *Proceedings of the IEEE Computer Society Bioinformatics Conference (CSB 2003)* pp. 337–346 IEEE, Los Alamitos, CA.

[8] Corpet, F. (1988) Multiple sequence alignment with hierarchical clustering. *Nucleic Acids Research,* **16**, 10881–10890.

[9] Deiman, B. & Pleij, C. W. A. (1997) Pseudoknots: a vital feature in viral RNA. *Seminars in Virology,* **8**, 166–175.

[10] Depiereux, E. & Feytmans, E. (1992) MATCH-BOX: a fundamentally new algorithm for the simultaneous alignment of several protein sequences. *Computer Applications in the Biosciences,* **8**, 501–509.

[11] Feng, D. F. & Doolittle, R. F. (1987) Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *Journal of Molecular Evolution,* **25**, 351–360.

[12] Gabow, H. & Tarjan, R. (1985) A linear-time algorithm for a special case of disjoint set union. *Journal of Computer and System Sciences,* **30**, 209–221

[13] Gusfield, D. (1993) Efficient methods for multiple sequence alignment with guaranteed error bounds. *Bulletin of Mathematical Biology,* **55**, 141–154.

[14] Gusfield, D. (1997) *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press.

[15] Hein, J. (1989) A new method that simultaneously aligns and reconstruct ancestral sequences for any number of homologous sequences, when the phylogeny is given. *Molecular Biology and Evolution,* **6**, 649–668.

[16] Higgins, D. & Sharpe, P. (1988) CLUSTAL: a package for performing multiple sequence alignment on a microcomputer. *Gene,* **73**, 237–244.

[17] Hirschberg, D. (1975) A linear space algorithm for computing maximal common subseqeuences. *Communications of the ACM,* **18**, 341–343.

[18] Kruskal, J. (1956) On the shrtest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society,* **7**, 48–50.

[19] Li, M., Ma, B. & Wang, L. (2000) Near optimal multiple alignment within a band in polynomial time. In *Proceedings of the Thirty Second Annual ACM Symposium on Theory of Computing (STOC 2000)* pp. 425–434 ACM Press, Portland.

[20] Lipman, D. & Pearson, W. (1985) Rapid and sensitive protein simularity search. *Science,* **227**, 1435–1411.

[21] Myers, E. W. & Miller, W. (1988) Optimal alignment in linear space. *CABIOS,* **4**, 11–17.

[22] Needleman, S. & Wunsch, C. (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Evolution,* **48**, 443–453

[23] Nicholas, H. B., Ropelewski, A. J. & Deerfield, D. W. (2002) Strategies for multiple sequence alignment. *Biotechniques,* **32**, 592–603.

[24] Notredame, C. (2002) Recent progresses in multiple sequence alignment: a survey. *Pharmacogenomics,* **3**, 131–144.

[25] Notredame, C., Higgins, D. G. & Heringa, J. (2000) T-Coffee: a novel method for fast and accurate multiple sequence alignment. *Journal of Molecular Biology,* **302**, 205–217.

[26] Pevzner, P. A. (1992) Multiple alignment, communication cost, and graph matching. *SIAM Journal on Applied Mathematics,* **52**, 1763–1779.

[27] Pearson, W. (1991) Searching protein sequence libraries: Computation of the sensitivity and selectivity of the Smith-Waterman and FASTA algorithm. *Genomics,* **11**, 635–650.

[28] Pleij, C. W. A. (1994) RNA pseudoknots. *Current Opinion in Structural Biology,* **4**, 337–344.

[29] Ravi, R. & Kececioglu, J. (1998) Approximation algorithms for multiple sequence alignment under a fixed evolutionary tree. *Discrete Applied Mathematics,* **88**, 355–366.

[30] Rivas, E. & Eddy, S. (1999) A dynamic programming algorithm for RNA structure prediction including pseudoknots. *Journal of Molecular Biology,* **285**, 2053–2068.

[31] Saitou, N. & Nei, M. (1987) The neighbor-joining method: a new mothod for reconstructing phylogenetic trees. *Molecular Biology and Evolution,* **4**, 406–425.

[32] Schuler, G. D., Altschul, S. F. & Lipman, D. J. (1991) A workbench for multiple alignment construction and analysis. *Proteins: Structure, Function and Genetics,* **9**, 180–190.

[33] Sneath, P. & Sokal, R. (1973) *Numerical Taxonomy.* Freeman, San Francisco, CA.

[34] Tang, C. Y., Lu, C. L., Chang, M. D. T., Tsai, Y. T., Sun, Y. J., Chao, K. M., Chang, J. M., Chiou, Y. H., Wu, C. M., Chang, H. T. & Chou, W. I. (2003) Constrained multiple sequence alignment tool development and its application to RNase family alignment. *Journal of Bioinformatics and Computational Biology,* **1**, 267–287.

[35] Taylor, W. R. (1987) Multiple sequence alignment by a pairwise algorithm. *CABIOS,* **3**, 81–87.

[36] Taylor, W. R. (1994) Motif-biased protein sequence alignment. *Journal of Computational Biology,* **1**, 297–310.

[37] Thompson, J. D., Higgs, D. G. & Gibson, T. J. (1994) CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position specific gap penalties, and weight matrix choice. *Nucleic Acids Research,* **22**, 4673–4680.

[38] Thompson, J. D., Plewniak, F., Thierry, J.-C. & Poch, O. (2000) DbClustal: rapid and reliable global multiple alignments of protein sequences detected by database searches. *Nucleic Acids Research,* **28**, 2919–2926.

[39] Wang, L. & Gusfield, D. (1997) Improved approximation algorithms for tree alignment. *Journal of Algorithms,* **25**, 255-273.

[40] Wang, L. & Jiang, T. (1994) On the complexity of multiple sequence alignment. *Journal of Computational Biology,* **1**, 337–348.

[41] Wang, L., Jiang, T. & Gusfield, D. (2000) A more efficient approximation scheme for tree alignment. *SIAM Journal on Computing,* **30**, 283–299.

[42] Wang, L., Jiang, T. & Lawler, E. (1996) Approximation algorithms for tree alignment with a givn phylogeny. *Algorithmica,* **16**, 302–315.

[43] Williams, G. D., Chang, R.-Y. & Brian, D. A. (1999) A phylogenetically conserved hairpin-type 39 untranslated region pseudoknot functions in coronavirus RNA replication. *Journal of Virology,* **73**, 8349–8355.

[44] Yu, C. T. (2003). Efficient algorithms for constrained sequence alignment problems. Master's thesis Department of Computer Science and Information Management, Providence University.