BIOINFORMATICS ORIGINAL PAPER Vol. 21 no. 1 2005, pages 20–30
doi:10.1093/bioinformatics/bth468

A memory-efficient algorithm for multiple sequence alignment with constraints

Chin Lung Lu* and Yen Pin Huang

Department of Biological Science and Technology, National Chiao Tung University, Hsinchu 300, Taiwan, Republic of China

Received on April 20, 2004; revised July 16, 2004; accepted August 3, 2004
Advance Access publication August 12, 2004

ABSTRACT
Motivation: Recently, the concept of the constrained sequence alignment was proposed to incorporate the knowledge of biologists about structures/functionalities/consensuses of their datasets into sequence alignment such that the user-specified residues/nucleotides are aligned together in the computed alignment. The currently developed programs use the so-called progressive approach to efficiently obtain a constrained alignment of several sequences. However, the kernels of these programs, the dynamic programming algorithms for computing an optimal constrained alignment between two sequences, run in $\mathcal{O}(\gamma n^2)$ memory, where γ is the number of the constraints and n is the maximum of the lengths of sequences. As a result, such a high memory requirement limits the overall programs to align short sequences only.
Results: We adopt the divide-and-conquer approach to design a memory-efficient algorithm for computing an optimal constrained alignment between two sequences, which greatly reduces the memory requirement of the dynamic programming approaches at the expense of a small constant factor in CPU time. This new algorithm consumes only $\mathcal{O}(\alpha n)$ space, where α is the sum of the lengths of constraints and usually $\alpha \ll n$ in practical applications. Based on this algorithm, we have developed a memory-efficient tool for multiple sequence alignment with constraints.
Availability: http://genome.life.nctu.edu.tw/MUSICME
Contact: cllu@mail.nctu.edu.tw

1 INTRODUCTION

Multiple sequence alignment (MSA) is one of the fundamental problems in computational molecular biology that have been studied extensively, because it is a useful tool in the phylogenetic analyses among various organisms, the identification of conserved motifs and domains in a group of related proteins, the secondary and tertiary structure prediction of a protein (or RNA), and so on (Carrillo and Lipman, 1988; Chan *et al.*, 1992; Gusfield, 1997; Nicholas *et al.*, 2002; Notredame, 2002). Moreover, MSA is one of the most challenging problems in computational molecular biology because it has been shown to be NP-complete under the consideration of sum-of-pairs scoring criteria (Kececioglu, 1993; Wang and Jiang, 1994; Bonizzoni and Vedova, 2001), which means that it seems to be hard to design an efficient algorithm for finding the mathematically optimal alignment. Hence, some approximate methods (Gusfield, 1993; Pevzner, 1992; Bafna *et al.*, 1997; Li *et al.*, 2000) and heuristic methods (Feng and Doolittle, 1987; Taylor, 1987; Corpet, 1988; Higgins and Sharpe, 1988; Thompson *et al.*, 1994) were introduced to overcome this problem.

Recently, the concept of the constrained sequence alignment was proposed to incorporate the knowledge of biologists regarding the structures/functionalities/consensuses of their datasets into sequence alignment such that the user-specified residues/nucleotides are aligned together in the computed alignment (Tang *et al.*, 2003). Tang *et al.* (2003) first 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 (CMSA) tool based on the progressive approach, where each constraint considered by Tang *et al.* is a single residue/nucleotide only. Their proposed algorithm for the two sequences runs in $\mathcal{O}(\gamma n^4)$ time and consumes $\mathcal{O}(n^4)$ space, where γ is the number of constrained residues and n is the maximum lengths of the sequences. Later, this result was improved independently by two groups of researchers to $\mathcal{O}(\gamma n^2)$ time and $\mathcal{O}(\gamma n^2)$ space using the same approach of dynamic programming (Yu, 2003; Chin *et al.*, 2003). In fact, each constraint 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 residues/nucleotides, instead of a single residue/nucleotide. In other words, the constraint specified by the biologists can be a fragment of several residues/nucleotides. For some applications, biologists may further expect that some mismatches are allowed among the residues/nucleotides of the columns requested to be aligned. Hence, Tsai *et al.* (2004) studied such a kind of the constrained sequence alignment and designed an algorithm of $\mathcal{O}(\gamma n^2)$ time and $\mathcal{O}(\gamma n^2)$

*To whom correspondence should be addressed.

space for two sequences. The improvements and extension above greatly increase the performances and practical usage of the CMSA tools developed using the progressive approach. However, the requirement of $\mathcal{O}(\gamma n^2)$ memory still limits the existing CMSA tools to align a set of short sequences, at most several hundreds of residues/nucleotides. To align large genomic sequences of at least several thousands of residues/nucleotides, there is a need to design a memory-efficient algorithm for the constrained pairwise sequence alignment (CPSA) problem, which is the key limiting factor relating to the applicable extent of the progressive CMSA tools. Hence, in this paper, we adopt the so-called divide-and-conquer approach to design a memory-efficient algorithm for solving the CPSA problem, which runs in $\mathcal{O}(\gamma n^2)$ time, but consumes only $\mathcal{O}(\alpha n)$ space, where α is the sum of the lengths of constraints and usually $\alpha \ll n$ in practical applications. Based on this algorithm, we have finally developed a memory-efficient CMSA tool using the progressive approach. Note that applying the divide-and-conquer approach to memory-efficiently align two or more sequences without any constraints has been studied extensively (Myers and Miller, 1988; Chao *et al.*, 1994; Tönges *et al.*, 1996; Stoye *et al.*, 1997a,b; Stoye, 1998). In contrast to the progressive approach used here, the divide-and-conquer algorithms proposed by Stoye *et al.* (Tönges *et al.*, 1996; Stoye *et al.*, 1997a,b; Stoye, 1998) considered the input sequences simultaneously and heuristically compute the good, but not necessarily optimal, dividing positions so that the resulting total MSA is close to an optimal MSA of the original sequences. In fact, many other CMSAs have been proposed from various perspectives, even using different approaches (Schuler *et al.*, 1991; Depiereux and Feytmans, 1992; Taylor, 1994; Myers *et al.*, 1996; Notredame *et al.*, 2000; Thompson *et al.*, 2000; Sammeth *et al.*, 2003). Of these various CMSAs, it is worth mentioning that Myers *et al.* (1996) obtained their CMSA by performing progressive multiple alignment under position-based constraints that are given by users; Sammeth *et al.* (2003) got their CMSA by performing simultaneous multiple alignment under segment-based constraints (as same as we studied here) that are pre-computed via a local segmented-based algorithm (Morgenstern, 1999). We refer the reader to their papers for details.

2 PROBLEM FORMULATION

Let $\mathcal{S} = \{S_1, S_2, \dots, S_\chi\}$ be the set of χ sequences over the alphabet Σ. Then an MSA of $\mathcal{S}$ is a rectangular matrix consisting of χ 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 $1 \leq i \leq \chi$. 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 (Carrillo and Lipman, 1988; Chan *et al.*, 1992; Gusfield, 1997; Nicholas *et al.*, 2002; Notredame, 2002).

Let $\delta(T_1, T_2)$ denote the Hamming distance between two subsequences T_1 and T_2 of equal length, which is equal to the number of mismatched pairs in the alignment of T_1 and T_2 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 $\mathcal{L}'$ of $\mathcal{L}$, let $\texttt{subseq}(S_i, \mathcal{L}')$ denote the subsequence of S_i whose residues/nucleotides are all in the band $\mathcal{L}'$, where $1 \leq i \leq \chi$. A subsequence $T = t_1 t_2 \dots t_\lambda$ is said to appear in $\mathcal{L}$ if $\mathcal{L}$ contains a band $\mathcal{L}'$ of λ columns, say $\pi_1, \pi_2, \dots, \pi_\lambda$, such that the characters of column π_j, $1 \leq j \leq \lambda$, are all equal to t_j, or equivalently, $\texttt{subseq}(S_i, \mathcal{L}') = T$ for each $1 \leq i \leq \chi$. If $\delta[\texttt{subseq}(S_i, \mathcal{L}'), T] \leq \lambda \times \epsilon$ for a given error ratio $0 \leq \epsilon < 1$ [i.e. some mismatches are allowed between $\texttt{subseq}(S_i, \mathcal{L}')$ and T], then T is said to approximately appear in $\mathcal{L}$. From the biological viewpoint, T can be considered as the consensus among the subsequences in $\mathcal{L}'$ and hence T is also called as an induced consensus by the band $\mathcal{L}'$. For any two subsequences T_1 and T_2, $T_1 \prec T_2$ is used to denote that T_1 (approximately) appears strictly before T_2 in $\mathcal{L}$ (i.e. their corresponding bands do not overlap). Let $\Omega = (C_1, C_2, \dots, C_\gamma)$ be an ordered set of γ constraints (i.e. subsequences), each $C_i = c_1^i c_2^i \dots c_{\lambda_i}^i$ with length of λ_i, where $1 \leq i \leq \gamma$. Then the CMSA of $\mathcal{S}$ with respect to Ω is defined as an alignment $\mathcal{L}$ of $\mathcal{S}$ in which all the constraints of Ω approximately appear in the order $C_1 \prec C_2 \prec \cdots \prec C_\gamma$ such that $\delta(\texttt{subseq}(S_i, \mathcal{L}'_j), C_j) \leq \lambda_j \times \epsilon$ for all $1 \leq i \leq \chi$ and $1 \leq j \leq \gamma$, where $\mathcal{L}'_j$ is the band of $\mathcal{L}$ whose induced consensus is C_j. Given a set $\mathcal{S}$ of χ sequences along with an ordered set Ω of γ constraints and an error ratio ϵ, the so-called CMSA problem is to find a CMSA w.r.t. Ω with the optimal SP score. When the number of sequences in $\mathcal{S}$ is restricted to two (i.e. $\chi = 2$), the CMSA problem is called as the CPSA problem.

3 ALGORITHM

In this section, we shall first design a memory-efficient algorithm for solving the CPSA problem with two given sequences $A = a_1 a_2 \dots a_m$ and $B = b_1 b_2 \dots b_n$, a given ordered set $\Omega = (C_1, C_2, \dots, C_\gamma)$ of γ constraints, each $C_i = c_1^i c_2^i \dots c_{\lambda_i}^i$ with length of λ_i, $1 \leq i \leq \gamma$, and a given error threshold ϵ. After that, we shall use it as the kernel to heuristically solve the CMSA problem.

For any sequence T, let $\texttt{pref}(T, l)$ [respectively, $\texttt{suff}(T, l)$] phase don't change denote the prefix (respectively, suffix) of T with length l. 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 $A_i = \texttt{pref}(A, i) = a_1a_2 \ldots a_i$, $B_j = \texttt{pref}(B, j) = b_1b_2 \ldots b_j$ and $\Omega_k = (C_1, C_2, \ldots, C_k)$, where $1 \le i \le m$, $1 \le j \le n$ and $1 \le k \le \gamma$. Let $\mathcal{M}_k(i, j)$ denote the score of an optimal constrained alignment of A_i and B_j w.r.t. Ω_k. Clearly, $\mathcal{M}_\gamma(m, n)$ is the score of an optimal constrained alignment of A and B w.r.t. Ω. An alignment $\mathcal{L}$ is called as a semi-constrained alignment of A_i and B_j w.r.t. Ω_k if it is a constrained alignment of A_i and B_j w.r.t. Ω_{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. Ω_k that ends with an induced consensus equal to $\texttt{pref}(C_k, h)$. Let $\mathcal{M}_k^D(i, j)$ [respectively, $\mathcal{M}_k^I(i, j)$] be the maximum scores of all constrained alignments of A_i and B_j w.r.t. Ω_k that end with a deletion pair $(a_i, -)$ [respectively, an insertion pair $(-, b_j)$]. By definition, it is not hard to derive the recurrence of $\mathcal{M}_k(i, j)$, $1 \le i \le m$ and $1 \le j \le n$, as follows. If $k = 0$, then $\mathcal{M}_k(i, j) = \max\{\mathcal{M}_k(i-1, j-1) + \sigma(a_i, b_j), \mathcal{M}_k^D(i, j), \mathcal{M}_k^I(i, j)\}$. If $1 \le k \le \gamma$, then $\mathcal{M}_k(i, j) = \max\{\mathcal{M}_k(i-1, j-1) + \sigma(a_i, b_j), \mathcal{M}_k^D(i, j), \mathcal{M}_k^I(i, j), \mathcal{N}_k(i, j, \lambda_k)\}$. Clearly, $\mathcal{N}_k(i, j, \lambda_k) = \mathcal{M}_{k-1}(i - \lambda_k, j - \lambda_k) + \Sigma_{0 \le h \le \lambda_k - 1}\sigma(a_{i-h}, b_{j-h})$, if $\delta(\texttt{suff}(A_i, \lambda_k), C_k) \le \lambda_k \times \epsilon$ and $\delta(\texttt{suff}(B_j, \lambda_k), C_k) \le \lambda_k \times \epsilon$; 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. Ω_k that end with a substitution pair (a_i, b_j). Let $\mathcal{L}_k^D(A_i, B_j)$ denote the alignment of A_i and B_j with score $\mathcal{M}_k^D(i, j)$ that 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, $\mathcal{M}_k^D(i, j) = \max\{\mathcal{M}_k^S(i-1, j) - w_o - w_e, \mathcal{M}_k^D(i-1, j) - w_e, \mathcal{M}_k^I(i-1, j) - w_o - w_e\}$. However, by including an extra $\mathcal{M}_k^D(i-1, j) - w_o - w_e$ into the right-hand side of the above recurrence, we can reformulate the above recurrence as $\mathcal{M}_k^D(i, j) = \max\{\mathcal{M}_k(i-1, j) - w_o - w_e, \mathcal{M}_k^D(i-1, j) - w_e\}$. Similar to the discussion above, the recurrence of $\mathcal{M}_k^I(i, j)$ can be derived as $\mathcal{M}_k^I(i, j) = \max\{\mathcal{M}_k(i, j-1) - w_o - w_e, \mathcal{M}_k^I(i, j-1) - w_e\}$.

According to the recurrences above, we designed 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 depicted the recurrences of matrices $\mathcal{M}_k, \mathcal{M}_k^D, \mathcal{M}_k^I$ and $\mathcal{N}_k$ for all $0 \le k \le \gamma$ by a three-dimensional (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 1 shows 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 1, with weight $\Sigma_{0 \le h \le \lambda_k - 1}\sigma(a_{i-h}, b_{j-h})$ 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. Ω. 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.

Hirschberg (1975) had developed a linear-space algorithm for solving the longest common subsequence problem based on the divide-and-conquer technique. Since then, this strategy has been extended to yield a number of memory-efficient algorithms for aligning biological sequences (Myers and Miller, 1988; Chao *et al.*, 1994). In this paper, we generalize the Hirschberg's algorithm so that it is capable of dealing with the CPSA. As compared with others, our generalization is more complicated because the grid graph $\mathcal{G}$ dealt here is 3D, instead of 2D, and the input sequences are accompanied with several constraints that need to be considered carefully. The central idea of our memory-efficient algorithm is to determine a middle position $(i_{\text{mid}}, j_{\text{mid}}, k_{\text{mid}})$ on an optimal path from $\mathcal{M}_0(0, 0)$ to $\mathcal{M}_\gamma(m, n)$ in $\mathcal{G}$ so that we were 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 manner, and finally the optimal constrained alignment is obtained completely by merging the series of the calculated mid-points (Fig. 2).

Before describing our algorithm, some notation must be introduced as follows. Let $\overline{A}_i$ and $\overline{B}_j$ denote the suffixes $a_{i+1}a_{i+2} \ldots a_m$ and $b_{j+1}b_{j+2} \ldots b_n$ of A and B, respectively, for $1 \le i \le m$ and $1 \le j \le n$. Let $\overline{\Omega}_k$ denote the ordered subset $(C_{k+1}, C_{k+2}, \ldots, C_\gamma)$ for $1 \le k \le \gamma$.

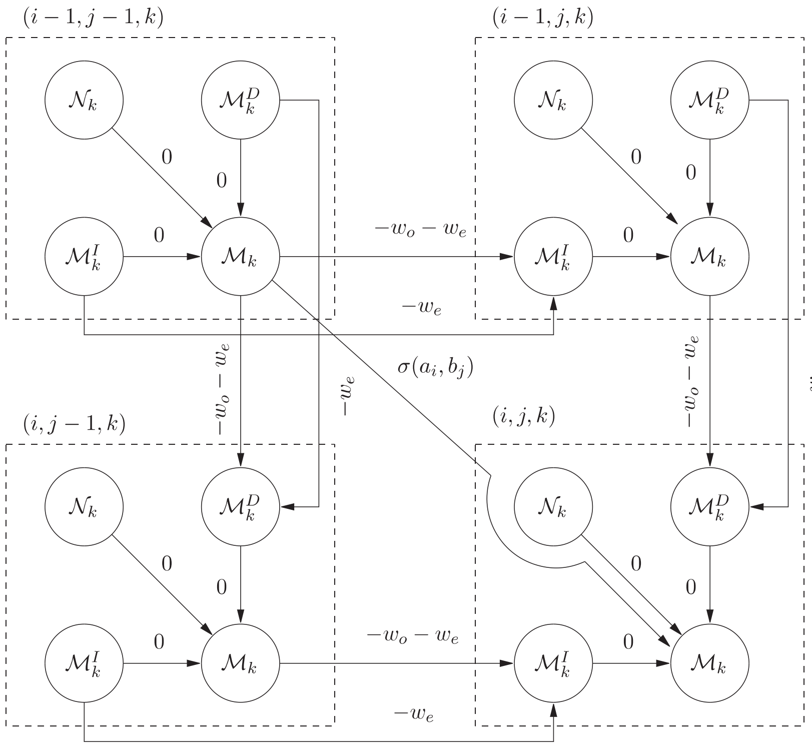


Fig. 1. The schematic diagram of four adjacent entries of $\mathcal{G}$, where entry (i,j,k) consists of four nodes $\mathcal{M}_k, \mathcal{M}_k^D, \mathcal{M}_k^I$ and $\mathcal{N}_k$ corresponding to $\mathcal{M}_k(i,j), \mathcal{M}_k^D(i,j), \mathcal{M}_k^I(i,j)$ and $\mathcal{N}_k(i,j,\lambda_k)$, respectively.

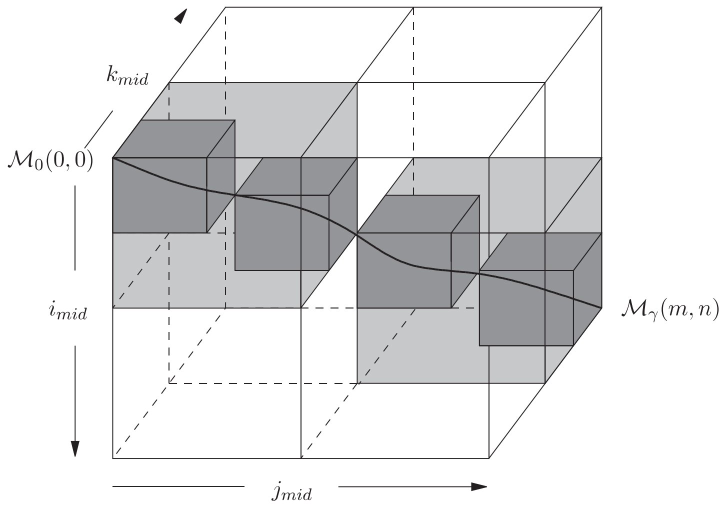


Fig. 2. Schematic diagram of divide-and-conquer approach: two light gray areas are the reduced subproblems after middle position $(i_{\text{mid}}, j_{\text{mid}}, k_{\text{mid}})$ is determined, each of which will be further divided into two subproblems of dark gray areas.

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. $\overline{\Omega}_k$, and define $\overline{\mathcal{M}}_k^S(i,j)$ ($\overline{\mathcal{M}}_k^D(i,j)$ and $\overline{\mathcal{M}}_k^I(i,j)$, respectively) to be the maximum score of all constrained alignments of $\overline{A}_i$ and $\overline{B}_j$ w.r.t. $\overline{\Omega}_k$ that begin with a substitution [deletion and insertion, respectively] pair (a_{i+1}, b_{j+1}) [$(a_{i+1}, -)$ and $(-, b_{j+1})$, respectively]. Let $\Omega_k(h) = [C_1, C_2, \ldots, C_{k-1}, \mathtt{pref}(C_k, h)]$ and $\overline{\Omega}_k(h) = [\mathtt{suff}(C_k, \lambda_k - h), C_{k+1}, \ldots, C_\gamma]$, where $1 \le h \le \lambda_k$. 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. $\overline{\Omega}_k(h)$ that begins with a band whose induced consensus is equal to $\mathtt{suff}(C_k, \lambda_k - h)$. Note that the recurrences for computing matrices $\overline{\mathcal{M}}_k, \overline{\mathcal{M}}^S_k, \overline{\mathcal{M}}^D_k$, $\overline{\mathcal{M}}^I_k$ and $\overline{\mathcal{N}}_k$ can be developed similarly as those for computing $\mathcal{M}_k$, $\mathcal{M}^S_k$, $\mathcal{M}^D_k$, $\mathcal{M}^I_k$ and $\mathcal{N}_k$, respectively. Clearly, $\mathcal{M}^S_k(i,j) = \mathcal{M}_k(i-1, j-1) + \sigma(a_i, b_j)$. If $\delta[\mathtt{suff}(A_i, \lambda_k), C_k] \le \lambda_k \times \epsilon$ and $\delta[\mathtt{suff}(B_j, \lambda_k), C_k] \le \lambda_k \times \epsilon$, 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 $1 < h \le \lambda_k$.

Next, we describe our divide-and-conquer algorithm, termed as CPSA-DC algorithm, for computing an optimal constrained alignment between A and B w.r.t. Ω as follows. The key point is to determine the middle position $(i_{\text{mid}}, j_{\text{mid}}, k_{\text{mid}})$ 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 $\mathtt{score}(\mathcal{L})$ 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. Ω and clearly $\mathtt{score}[\mathcal{L}_\gamma(A,B)] = \mathcal{M}_\gamma(m,n)$. Let $i_{\text{mid}} = \lfloor \frac{m}{2} \rfloor$. Then, we partition $\mathcal{L}_\gamma(A,B)$ into two parts by cutting it at the position immediately after $a_{i_{\text{mid}}}$ and we let $\mathcal{L}_{k_{\text{mid}}}(A_{i_{\text{mid}}}, B_{j_{\text{mid}}})$ denote the part containing $a_{i_{\text{mid}}}$ and $\overline{\mathcal{L}}_{k_{\text{mid}}}(\overline{A}_{i_{\text{mid}}}, \overline{B}_{j_{\text{mid}}})$ denote the remaining part, where $b_{j_{\text{mid}}}$ denotes the last character in $\mathcal{L}_{k_{\text{mid}}}(A_{i_{\text{mid}}}, B_{j_{\text{mid}}})$ from B, and k_{mid} denotes the largest index so that $\mathtt{pref}(C_{k_{\text{mid}}}, h_{\text{mid}})$ (approximately) appears in $\mathcal{L}_{k_{\text{mid}}}(A_{i_{\text{mid}}}, B_{j_{\text{mid}}})$. Then there are two possibilities when we consider the last aligned pair of $\mathcal{L}_{k_{\text{mid}}}(A_{i_{\text{mid}}}, B_{j_{\text{mid}}})$.

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

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

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

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

When $\mathcal{M}^D_{k_{\text{mid}}}(i_{\text{mid}}, j_{\text{mid}}) + \overline{\mathcal{M}}^D_{k_{\text{mid}}}(i_{\text{mid}}, j_{\text{mid}})$ is added to the right-hand side, the above recurrence is not changed, but can be reformulated as follows:

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

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

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

Now, we show how to use $\mathcal{O}(\alpha n)$, instead of $\mathcal{O}(\gamma mn)$, memory to determine $j_{\text{mid}}, k_{\text{mid}}$ and h_{mid}, where $\alpha = \sum_{1 \le k \le \gamma} \lambda_k$ and $\alpha \le \min\{m, n\}$ intrinsically. In fact, a single matrix E of size $(\gamma + 1) \times (n + 1)$ with each entry $E(k, j)$ of $\lambda_k + 4$ space is enough to compute $\mathcal{M}_k(i_{\text{mid}}, j)$, $\mathcal{M}^S_k(i_{\text{mid}}, j)$, $\mathcal{M}^D_k(i_{\text{mid}}, j)$ $\mathcal{M}^I_k(i_{\text{mid}}, j)$ and $\mathcal{N}_k(i_{\text{mid}}, j, h)$, for $1 \le j \le n$, $0 \le k \le \gamma$ and $1 \le h \le \lambda_k$. 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, termed as V_k 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

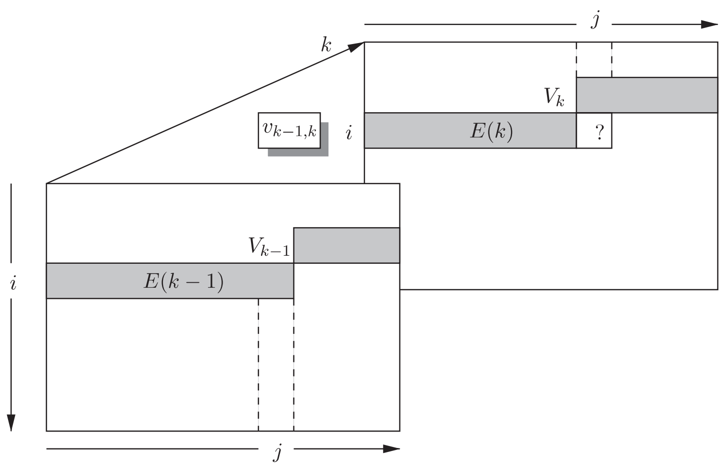


Fig. 3. The grid locations of $E(k-1)$, $E(k)$ and the values in V_{k-1} and V_k when the entry (i, j, k) of $\mathcal{G}$, marked with '?', is reached for the computation.

V_k; however, we need to first move $\mathcal{M}_k(i-1, j-1)$ in V_k into a space, named as $v_{k,k+1}$, where $1 \le i \le m$. 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 $v_{k-1,k}$; compute $\mathcal{N}_k(i, j, h)$ for each $2 \le h \le \lambda_k$, 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 were able to compute $\mathcal{M}_k(i, j)$. Figure 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 k-th row of E. Hence, the total needed space for computing and storing all $\mathcal{M}_k(i_{\text{mid}}, j)$, $\mathcal{M}_k^S(i_{\text{mid}}, j)$, $\mathcal{M}_k^D(i_{\text{mid}}, j)$ $\mathcal{M}_k^I(i_{\text{mid}}, j)$ and $\mathcal{N}_k(i_{\text{mid}}, j, h)$ is the sum of the space of matrix E, the space of all V_k and the space of all $v_{k,k+1}$, where $1 \le j \le n$, $0 \le k \le \gamma$ and $1 \le h \le \lambda_k$, which is equal to $\mathcal{O}(\alpha n)$. Similarly, the required matrix, denoted by $\overline{E}$, for computing all $\overline{\mathcal{M}}_k(i_{\text{mid}}, j)$, $\overline{\mathcal{M}}_k^S(i_{\text{mid}}, j)$, $\overline{\mathcal{M}}_k^D(i_{\text{mid}}, j)$ $\overline{M}_k^I(i_{\text{mid}}, j)$ and $\overline{\mathcal{N}}_k(i_{\text{mid}}, j, h)$ still needs $\mathcal{O}(\alpha n)$ space. Hence, the determination of $j_{\text{mid}}, k_{\text{mid}}$ and h_{mid} can be performed in $\mathcal{O}(\alpha n)$ space. The details of CPSA-DC algorithm are described as follows. Note that the program code of BestScoreRev is similar to that of BestScore and hence is omitted here. In the codes, the variable $E(\mathcal{M}_k(i_{\text{mid}}, j))$ is used to denote the value of $\mathcal{M}_k(i_{\text{mid}}, j)$ in $E(k, j)$ and others are analogous. The global variables $\mathcal{H}_A(k,h) = \delta(\texttt{suff}(A_{i_{\text{mid}}}, h), \texttt{pref}(C_k, h))$, $\overline{\mathcal{H}}_A(k,h) = \delta(\texttt{pref}(\overline{A}_{i_{\text{mid}}}, \lambda_k - h), \texttt{suff}(C_k, \lambda_k - h))$, $\mathcal{H}_B(j,k,h) = \delta(\texttt{suff}(B_j, h), \texttt{pref}(C_k, h))$, and $\overline{\mathcal{H}}_B(j,k,h) = \delta(\texttt{pref}(\overline{B}_j, \lambda_k - h), \texttt{suff}(C_k, \lambda_k - h))$ are computed in Algorithm BestScore so that they can be used directly in Algorithm CPSA-DC.

Algorithm CPSA-DC$(i_{\text{start}}, i_{\text{end}}, j_{\text{start}}, j_{\text{end}}, k_{\text{start}}, k_{\text{end}})$
Input: Sequences $a_{i_{\text{start}}} \cdots a_{i_{\text{end}}}$ and $b_{j_{\text{start}}} \cdots b_{j_{\text{end}}}$ with
  constraints $(C_{k_{\text{start}}}, \ldots, C_{k_{\text{end}}})$

```
1: if $(i_{\text{start}} > i_{\text{end}})$ or $(j_{\text{start}} > j_{\text{end}})$ then
     Align the nonempty sequence with spaces;
   else
     $i_{\text{mid}} = \lfloor \frac{i_{\text{start}} + i_{\text{end}}}{2} \rfloor$;
     BestScore$(i_{\text{start}}, i_{\text{mid}}, j_{\text{start}}, j_{\text{end}}, k_{\text{start}}, k_{\text{end}})$;
     BestScoreRev$(i_{\text{mid}} + 1, i_{\text{end}}, j_{\text{start}}, j_{\text{end}}, k_{\text{start}}, k_{\text{end}})$;
   end if
2: max $= -\infty$;
    for $j = j_{\text{start}} - 1$ to $j_{\text{end}}$ do
      for $k = k_{\text{start}} - 1$ to $k_{\text{end}}$ do
        if $E(\mathcal{M}_k^D(i_{\text{mid}}, j)) + \overline{E}(\overline{\mathcal{M}}_k(i_{\text{mid}}, j)) >$ max then
          max $= E(\mathcal{M}_k^D(i_{\text{mid}}, j)) + \overline{E}(\overline{\mathcal{M}}_k(i_{\text{mid}}, j))$;
          $j_{\text{mid}} = j$; $k_{\text{mid}} = k$; $type =$ case 1;
        end if
        if $E(\mathcal{M}_k^D(i_{\text{mid}}, j)) + \overline{E}(\overline{\mathcal{M}}_k^D(i_{\text{mid}}, j)) + w_o >$ max then
          max $= E(\mathcal{M}_k^D(i_{\text{mid}}, j)) + \overline{E}(\overline{\mathcal{M}}_k^D(i_{\text{mid}}, j)) + w_o$;
          $j_{\text{mid}} = j$; $k_{\text{mid}} = k$; type $=$ case 2;
        end if
        if $E(\mathcal{M}_k^S(i_{\text{mid}}, j)) + \overline{E}(\overline{\mathcal{M}}_k(i_{\text{mid}}, j)) >$ max then
          max $= E(\mathcal{M}_k^S(i_{\text{mid}}, j)) + \overline{E}(\overline{\mathcal{M}}_k(i_{\text{mid}}, j))$;
          $j_{\text{mid}} = j$; $k_{\text{mid}} = k$; type $=$ case 3;
        end if
        if $k \ge 1$ then
          for $h = 1$ to $\lambda_k - 1$ do
            if $\left(\frac{\mathcal{H}_A(k,h) + \overline{\mathcal{H}}_A(k,h)}{\lambda_k} \le \epsilon\right)$ and
               $\left(\frac{\mathcal{H}_B(j,k,h) + \overline{\mathcal{H}}_B(j,k,h)}{\lambda_k} \le \epsilon\right)$ then
              if $E(\mathcal{N}_k(i_{\text{mid}}, j, h)) + \overline{E}(\overline{\mathcal{N}}_k(i_{\text{mid}}, j, h)) >$
                max then
                max $= E(\mathcal{N}_k(i_{\text{mid}}, j, h))$
                      $+ \overline{E}(\overline{\mathcal{N}}_k(i_{\text{mid}}, j, h))$;
                $j_{\text{mid}} = j$; $k_{\text{mid}} = k$; $h_{\text{mid}} = h$; type $=$ case 4;
              end if
            end if
          end for
        if $\left(\frac{\mathcal{H}_A(k,\lambda_k)}{\lambda_k} \le \epsilon\right)$ and $\left(\frac{\mathcal{H}_B(j,k,\lambda_k)}{\lambda_k} \le \epsilon\right)$ then
          if $E(\mathcal{N}_k(i_{\text{mid}}, j, \lambda_k)) + \overline{E}(\overline{\mathcal{M}}_k(i_{\text{mid}}, j)) >$
            max then
            max $= E(\mathcal{N}_k(i_{\text{mid}}, j, \lambda_k)) + \overline{E}(\overline{\mathcal{M}}_k(i_{\text{mid}}, j))$;
            $j_{\text{mid}} = j$; $k_{\text{mid}} = k$; $h_{\text{mid}} = h$; type = case 5;
          end if
        end if
      end if
    end for
  end for
3: if type = case 1 then
    CPSA-DC($i_{\text{start}}, i_{\text{mid}} - 1, j_{\text{start}}, j_{\text{mid}}, k_{\text{start}}, k_{\text{mid}}$);
    Align $a_{i_{\text{mid}}}$ with a space;
    CPSA-DC($i_{\text{mid}} + 1, i_{\text{end}}, j_{\text{mid}} + 1, j_{\text{end}}, k_{\text{mid}} + 1, k_{\text{end}}$);
  end if
  if type = case 2 then
    CPSA-DC($i_{\text{start}}, i_{\text{mid}} - 1, j_{\text{start}}, j_{\text{mid}}, k_{\text{start}}, k_{\text{mid}}$);
    Align $a_{i_{\text{mid}}} a_{i_{\text{mid}}+1}$ with two spaces;
    CPSA-DC($i_{\text{mid}} + 2, i_{\text{end}}, j_{\text{mid}} + 1, j_{\text{end}}, k_{\text{mid}} + 1, k_{\text{end}}$);
  end if
  if type = case 3 then
    CPSA-DC($i_{\text{start}}, i_{\text{mid}} - 1, j_{\text{start}}, j_{\text{mid}} - 1, k_{\text{start}}, k_{\text{mid}}$);
    Align $a_{i_{\text{mid}}}$ with $b_{j_{\text{mid}}}$;
    CPSA-DC($i_{\text{mid}} + 1, i_{\text{end}}, j_{\text{mid}} + 1, j_{\text{end}}, k_{\text{mid}} + 1, k_{\text{end}}$);
  end if
  if type = case 4 then
    CPSA-DC($i_{\text{start}}, i_{\text{mid}} - h_{\text{mid}}, j_{\text{start}}, j_{\text{mid}} - h_{\text{mid}}, k_{\text{start}},$
      $k_{\text{mid}} - 1$);
    Align $a_{i_{\text{mid}}-h_{\text{mid}}+1} \cdots a_{i_{\text{mid}}+\lambda_k-h_{\text{mid}}}$ with $b_{j_{\text{mid}}-h_{\text{mid}}+1} \cdots$
      $b_{j_{\text{mid}}+\lambda_k-h_{\text{mid}}}$;
    CPSA-DC($i_{\text{mid}} + \lambda_k - h_{\text{mid}} + 1, i_{\text{end}}, j_{\text{mid}} + \lambda_k - h_{\text{mid}}$
      $+ 1, j_{\text{end}}, k_{\text{mid}} + 1, k_{\text{end}}$);
  end if
  if type = case 5 then
    CPSA-DC($i_{\text{start}}, i_{\text{mid}} - \lambda_k, j_{\text{start}}, j_{\text{mid}} - \lambda_k, k_{\text{start}},$
      $k_{\text{mid}} - 1$);
    Align $a_{i_{\text{mid}}-\lambda_k+1} \cdots a_{i_{\text{mid}}}$ with $b_{j_{\text{mid}}-\lambda+1} \cdots b_{j_{\text{mid}}}$;
    CPSA-DC($i_{\text{mid}} + 1, i_{\text{end}}, j_{\text{mid}} + 1, j_{\text{end}}, k_{\text{mid}} + 1, k_{\text{end}}$);
  end if
```

Algorithm BestScore($i_{\text{start}}, i_{\text{end}}, j_{\text{start}}, j_{\text{end}}, k_{\text{start}}, k_{\text{end}}$)
Input: Sequences $a_{i_{\text{start}}} \cdots a_{i_{\text{end}}}$ and $b_{j_{\text{start}}} \cdots b_{j_{\text{end}}}$ with constraints $(C_{k_{\text{start}}}, \ldots, C_{k_{\text{end}}})$

```
1: /* Reindex */
  $m = i_{\text{start}} - i_{\text{end}} + 1$; $n = j_{\text{start}} - j_{\text{end}} + 1$;
  $\gamma = k_{\text{start}} - k_{\text{end}} + 1$;
2: /* Initialization */
  for $j = 0$ to $n$ do
    for $k = 0$ to $\gamma$ do
      $E(\mathcal{M}_k^S(i_{\text{mid}}, j)) = E(\mathcal{M}_k^D(i_{\text{mid}}, j)) = -\infty$;
      if $(j = 0)$ or $(k > 0)$ then $E(\mathcal{M}_k^I(i_{\text{mid}}, j)) = -\infty$;
        else $E(\mathcal{M}_k^I(i_{\text{mid}}, j)) = -w_{\text{o}} - j w_{\text{e}}$;
      if $(j = 0)$ and $(k = 0)$ then $E(\mathcal{M}_k(i_{\text{mid}}, j)) = 0$;
        else $E(\mathcal{M}_k(i_{\text{mid}}, j)) = -\infty$;
      if $k \ge 1$ then
        for $h = 1$ to $\lambda_k$ do $E(\mathcal{N}_k(i_{\text{mid}}, j, h)) = -\infty$;
      end if
    end for
  end for
3: /* Computation */
  for $i = 1$ to $m$ do
    for $k = 0$ to $\gamma$ do /* For the case of $j = 0$ */
      $V_k(\mathcal{M}_k(i_{\text{mid}}, 0)) = E(\mathcal{M}_k(i_{\text{mid}}, 0))$;
      if $k \ge 1$ then
        for $h = 1$ to $\lambda_k$ do $V_k(\mathcal{N}_k(i_{\text{mid}}, 0, h))$
            $= E(\mathcal{N}_k(i_{\text{mid}}, 0, h)))$;
      end if
      $E(\mathcal{M}_k^S(i_{\text{mid}}, 0)) = E(\mathcal{M}_k^I(i_{\text{mid}}, 0)) = -\infty$;
      $E(\mathcal{M}_k(i_{\text{mid}}, 0)) = E(\mathcal{M}_k^D(i_{\text{mid}}, 0)) = -w_{\text{o}} - j w_{\text{e}}$;
    end for
    for $j = 1$ to $n$ do /* For the case of $j > 0$ */
    for $k = 0$ to $\gamma$ do
      temp$_k(\mathcal{M}_k(i_{\text{mid}}, j)) = E(\mathcal{M}_k(i_{\text{mid}}, j))$ ;
      if $k \ge 1$ then
        for $h = 1$ to $\lambda_k$ do temp$_k(\mathcal{N}_k(i_{\text{mid}}, j, h))$
            $= E(\mathcal{N}_k(i_{\text{mid}}, j, h))$;
      end if
      $E(\mathcal{M}_k^S(i_{\text{mid}}, j)) = V_k(\mathcal{M}_k(i_{\text{mid}}, j))$
                    $+ \sigma(a_{i_{\text{start}}+i-1}, b_{j_{\text{start}}+j-1})$;
      $E(\mathcal{M}_k^D(i_{\text{mid}}, j)) = \max\{E(\mathcal{M}_k^D(i_{\text{mid}}, j))$
                    $- w_{\text{e}}, E(\mathcal{M}_k(i_{\text{mid}}, j)) - w_{\text{o}} - w_{\text{e}}\}$;
      $E(\mathcal{M}_k^I(i_{\text{mid}}, j)) = \max\{E(\mathcal{M}_k^I(i_{\text{mid}}, j - 1))$
                    $- w_{\text{e}}, E(\mathcal{M}_k(i_{\text{mid}}, j - 1))$
                    $- w_{\text{o}} - w_{\text{e}}\}$;
      if $k \ge 1$ then
        for $h = 1$ to $\lambda_k$ do
          if $h = 1$ then
            $E(\mathcal{N}_k(i_{\text{mid}}, j, h)) = v_{k-1,k} + \sigma(a_{i_{\text{start}}+i-\lambda_k},$
                          $b_{j_{\text{start}}+j-\lambda_k})$;
          else
            $E(\mathcal{N}_k(i_{\text{mid}}, j, h)) = V_k(\mathcal{N}_k(i_{\text{mid}}, j, h - 1))$
                          $+ \sigma(a_{i_{\text{start}}+i-\lambda_k+h-1},$
                          $b_{j_{\text{start}}+j-\lambda_k+h-1})$;
          end if
        end for
      end if
      $E(\mathcal{M}_k(i_{\text{mid}}, j))$
        $= \max\left\{\begin{array}{l} E(\mathcal{M}_k^D(i_{\text{mid}}, j)), E(\mathcal{M}_k^I(i_{\text{mid}}, j)), \\ E(\mathcal{N}_k(i_{\text{mid}}, j, \lambda_k)) \\ V_k(\mathcal{M}_k(i_{\text{mid}}, j)) + \sigma(a_{i_{\text{start}}+i-1}, \\ \quad b_{j_{\text{start}}+j-1}), \end{array}\right\}$;
      $v_{k,k+1} = V_k(M_k(i_{\text{mid}}, j))$;
      $V_k(\mathcal{M}_k(i_{\text{mid}}, j)) = \text{temp}_k(\mathcal{M}_k(i_{\text{mid}}, j))$;
      if $k \ge 1$ then
        for $h = 1$ to $\lambda_k$ do
          $V_k(\mathcal{N}(i_{\text{mid}}, j, h)) = \text{temp}_k(\mathcal{N}_k(i_{\text{mid}}, j, h))$;
        end for
      end if
```

```
            if $i = m$ and $k \geq 1$ then
              for $h = 1$ to $\lambda_k$ do
                if $h = 1$ then
                  if $j = 1$ and $a_{i_{\text{start}}+i-\lambda_k} \neq c_h^k$ then
                    $\mathcal{H}_A(k,h) = 1$; else $\mathcal{H}_A(k,h) = 0$;
                  if $b_{j_{\text{start}}+j-\lambda_k} \neq c_h^k$ then
                    $\mathcal{H}_B(j,k,h) = 1$; else $\mathcal{H}_B(j,k,h) = 0$;
                else
                  if $j = 1$ and $a_{i_{\text{start}}+i-\lambda_k+h-1} \neq c_h^k$ then
                    $\mathcal{H}_A(k,h) = \mathcal{H}_A(k,h-1) + 1$;
                  if $b_{j_{\text{start}}+j-\lambda_k+h-1} \neq c_h^k$ then $\mathcal{H}_B(j,k,h)$
                    $= \mathcal{H}_B(j,k,h-1) + 1$;
                end if
              end for
            end if
          end for
        end for
      end for
```

Now, we analyze the time-complexity of our CPSA-DC algorithm for solving the CPSA. As shown in Figure 2, after determining the middle position $(i_{\text{mid}}, j_{\text{mid}}, k_{\text{mid}})$ 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 $(i_{\text{mid}}, j_{\text{mid}}, k_{\text{mid}})$, 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 entries in $\mathcal{G}$. It 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 Ψ denote the size of the original problem (i.e. $\Psi = \gamma mn$). Then the total time-complexity of our CPSA-DC algorithm is equal to $\Psi + \frac{\Psi}{2} + \frac{\Psi}{4} + \cdots = 2\Psi$, which is twice as high as the CPSA-DP algorithm. Using the CPSA-DC algorithm as a kernel, we were able to design a memory-efficient algorithm, termed CMSA-DC, for progressively aligning multiple input sequences into a CMSA according to the branching order of a guide tree. The above progressive method we adopted was proposed by Tang *et al.* (2003). Owing to space limitation, we refer the reader to their paper for the details of its implementation.

4 EXPERIMENTAL RESULTS

We use Java language to implement the CMSA-DC algorithm as a web server, called as MuSiC-ME (Memory-Efficient tool for Multiple Sequence Alignment with Constraints). The input of the MuSiC-ME system consists of a set of protein/DNA/RNA sequences and a set of user-specified constraints, each with a fragment of residue/nucleotide that (approximately) appears in all input sequences. The output of MuSiC-ME is a CMSA in which the fragments of the input sequences whose residues/nucleotides exhibit a given degree of similarity to a constraint are aligned together. For its biological applications, we refer the reader to other related papers (Tang *et al.*, 2003; Tsai *et al.*, 2004).

Table 1. The information of the tested families and their constraints

Family	#SEQ	MAXSEQ	#CON	MAXCON
Protease	6	123	4	1
Globin	6	146	7	2
RNase	6	185	3	1
Kinase	6	353	10	3
CoV-3′-UTR	6	422	12	2

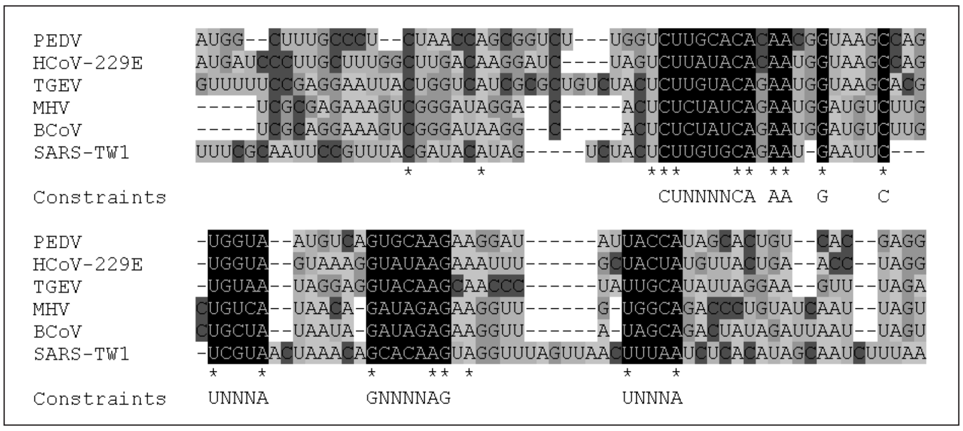
#SEQ is the number of sequences, MAXSEQ is the maximum length of sequences, #CON is the number of constraints and MAXCON is the maximum length of constraints.

Table 2. The comparison of CPU time and memory usage between MuSiC and MuSiC-ME

Family	MuSiC		MuSiC-ME	
	CPU Time (s)	Memory (MB)	CPU Time (s)	Memory (MB)
Protease	6	25.4	6	15.5
Globin	23	42.0	18	15.5
RNase	11	32.0	8	15.5
Kinase	131	160.8	96	15.9
CoV-3′-UTR	—	—	165	17.4

The memory usage includes JVM (Java Virtual Machine), code (MuSiC/MuSiC-ME) and data, and MuSiC cannot deal with the case of CoV-3′-UTR due to running out of memory.

In the following, we evaluate our memory-efficient MuSiC-ME system and compare its running time and memory to the original MuSiC system (Tsai *et al.*, 2004), whose kernel CPSA algorithm was implemented by the dynamic programming approach. We chose five families of protein/RNA sequences as our testing datasets, each of which has been shown to contain an ordered series of conserved motifs related to the structures/functionalities/consensuses of the family (McClure *et al.*, 1994; Chin *et al.*, 2003; Tang *et al.*, 2003; Tsai *et al.*, 2004): (1) the aspartic acid protease family (Protease), (2) the hemoglobins family (Globin), (3) the ribonuclease family (RNase), (4) the kinase family (Kinase) and (5) the 3′- untranslated region of the coronaviruses (CoV-3′-UTR). From each family, we have selected a representative set of sequences and adopted the ordered series of conserved motifs as the constraints. Table 1 lists the information of the tested families and their constraints. All tests were run with default parameters on IBM PC with 1.26 GHz processor and 512 MB RAM under Linux system. Table 2 lists the CPU time and memory usage of our experiments using MuSiC and MuSiC-ME. It shows that the memory usage of MuSiC-ME is much smaller than that of MuSiC for large-scale sequences, and the CPU time required by MuSiC-ME is smaller than that required by MuSiC for



```
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
```

Fig. 4. 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.

short sequences, since we have simplified the recurrences of the dynamic programming here.

It is worth mentioning that in MuSiC-ME system, the letters representing the constraints are not just the individual residues/nucleotides, but also the IUPAC (International Union of Pure and Applied Chemistry) codes. For example, nucleotides `N` and `R` have the meanings of any nucleotides and purine (i.e. `A` or `G`), respectively. This enhanced improvement will enable the user to define more flexible constraints or combine several small constraints with fixed distances into a large one. For example, consider our fifth experiment above related to the 3′-UTRs of the coronavirus sequences, including HCV-229E (human coronavirus), PEDV (porcine epidemic diarrhea virus), TGEV (porcine transmissible gastroenteritis virus), BCV (bovine coronavirus), MHV (mouse hepatitis virus) and SARS-TW1 (severe acute respiratory syndrome virus). All the 12 adopted constraints appear in the fragment sequences that were able to fold themselves into a stable pseudoknot structure (Williams *et al.*, 1999; Tsai *et al.*, 2004). However, these adopted constraints are too short to correctly align the truly conserved motifs of sequences together, since the short constraints occur frequently in the large genomic sequences that led to the difficulty in identifying the true occurrences. In fact, four pairs of two consecutive constraints appear in the stem regions (containing no loops) of pseudoknots and each paired constraints is separated by a non-conserved subsequence of fixed length. Hence, we can combine each pair of constraints into a new and larger constraint by representing the non-conserved part with `N`. Consequently, we got eight new constraints with the order of (`CUNNNNC, A, AA, G, C, UNNNA, GNNNNAG, UNNNA`) for this dataset. After running MuSiC-ME, a satisfied CMSA was found (Figure 4), where the band of the resulting CMSA corresponding to a constraint is black and its corresponding constraint is displayed beneath it. 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 (Pleij, 1994; Deiman and Pleij, 1997). In fact, this fragment is the pseudoknot sequence of SART-TW1 that was found by Tsai *et al.* (2004) using MuSiC to align the 3′-UTR of SARS-TW1 with the pseudoknot sequences, instead of 3′-UTRs, of other coronaviruses. The input sequences of the above experiment were also tested by Clustal W 1.82, the most commonly used MSA tool. According to its resulting MSA as shown in Figure 5, the fragments of all pseudoknots, including our detected pseudoknot for SARS-TW1, were not able to align well so that it is difficult for us to identify the exact fragment of the SARS-TW1 pseudoknot from this MSA.

5 CONCLUSIONS

In this paper, we designed a memory-efficient program for performing the CMSA, which can incorporate the knowledge of biologists about the structures/functionalities/consensuses of their datasets into sequence alignment such that the user-specified residues/nucleotides are aligned together. We first used the divide-and-conquer approach to design a memory-efficient algorithm for optimally aligning two sequences with constraints, and then based on this algorithm, we used the progressive method to develop a memory-efficient tool, called MuSiC-ME, for heuristically aligning multiple sequences with constraints. The proposed MuSiC-ME system makes it possible to align several large-scale protein/DNA/RNA 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 more flexible constraints or combine several small constraints with fixed distances into a large one. It is worth mentioning that the A* algorithm, a heuristic search method in Artificial Intelligence, has been extensively used to time- and/or memory-efficiently solve the

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

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

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

Fig. 5. 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.

general MSA problem without constraints (Ikeda and Imai, 1994, 1999; Kobayashi and Imai, 1999; Lermen and Reinert, 2000). Hence, it is interesting to study whether or not the A* algorithm can still be applied to the CMSA problem.

ACKNOWLEDGEMENTS

The authors would like to thank the anonymous referees for their constructive comments to the presentation of this paper. This work was supported in part by National Science Council of Republic of China under grant NSC93-2213-E-009-113.

REFERENCES

Bafna,V., Lawler,E.L. and Pevzner,P.A. (1997) Approximation algorithms for multiple sequence alignment. *Theoret. Comput. Sci.*, **182**, 233–244.

Bonizzoni,P. and Vedova,G.D. (2001) The complexity of multiple sequence alignment with SP-score that is a metric. *Theoret. Comput. Sci.*, **259**, 63–79.

Carrillo,H. and Lipman,D. (1988) The multiple sequence alignment problem in biology. *SIAM J. Appl. Math.*, **48**, 1073–1082.

Chan,S.C., Wong,A.K.C. and Chiu,D.K.Y. (1992) A survey of multiple sequence comparison methods. *Bull. Math. Biol.*, **54**, 563–598.

Chao,K.M., Hardison,R.C. and Miller,W. (1994) Recent developments in linear-space alignment methods: a survey. *J. Comput. Biol.*, **1**, 271–291.

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

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

Deiman,B. and Pleij,C.W.A. (1997) Pseudoknots: a vital feature in viral RNA. *Semin. Virol.*, **8**, 166–175.

Depiereux,E. and Feytmans,E. (1992) MATCH-BOX: a fundamentally new algorithm for the simultaneous alignment of several protein sequences. *Comput. Appl. Biosci.*, **8**, 501–509.

Feng,D.F. and Doolittle,R.F. (1987) Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *J. Mol. Evol.*, **25**, 351–360.

Gusfield,D. (1993) Efficient methods for multiple sequence alignment with guaranteed error bounds. *Bull. Math. Biol.*, **55**, 141–154.

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

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

Hirschberg,D.S. (1975) A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, **18**, 341–343.

Ikeda,T. and Imai,H. (1994) Fast A* algorithms for multiple sequence alignment. In *Proceedings of the Genome Informatics Workshop*. Universal Academy Press, Tokyo, pp. 90–99.

Ikeda,T. and Imai,H. (1999) Enhanced A* algorithms for multiple alignments: optimal alignments for several sequences and k-opt approximate alignments for large cases. *Theoret. Comput. Sci.*, **210**, 341–374.

Kececioglu,J.D. (1993) The maximum weight trace problem in multiple sequence alignment. In *Proceedings of the Fourth Annual Symposium on Combinatorial Pattern Matching (CPM 2004)*, LNCS **684**. Springer-Verlag, Heidelberg, Germany, pp. 106–119.

Kobayashi,H. and Imai,H. (1999) Improvement of the A* algorithm for multiple sequence alignment. In *Proceedings of the Genome Informatics Workshop*. Universal Academy Press, Tokyo, pp. 120–130.

Lermen,M. and Reinert,K. (2000) The practical use of the A* algorithm for exact multiple sequence alignment. *J. Comput. Biol.*, **7**, 655–672.

Li,M., Ma,B. and 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). ACM Press, Portland, OR, pp. 425–434.

McClure,M.A., Vasi,T.K. and Fitch,W.M. (1994) Comparative analysis of multiple protein-sequence alignment methods. *Mol. Biol. Evol.*, **11**, 571–592.

Morgenstern,B. (1999) DIALIGN 2: improvement of the segment-to-segment approach to multiple sequence alignment. *Bioinformatics*, **15**, 211–218.

Myers,E.W. and Miller,W. (1988) Optimal alignment in linear space. *Comput. Appl. Biosci.*, **4**, 11–17.

Myers,G., Selznick,S., Zhang,Z. and Miller,W. (1996) Progressive multiple alignment with constraints. *J. Comput. Biol.*, **3**, 563–572.

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

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

Notredame,C., Higgins,D.G. and Heringa,J. (2000) T-Coffee: a novel method for fast and accurate multiple sequence alignment. *J. Mol. Biol.*, **302**, 205–217.

Pevzner,P.A. (1992) Multiple alignment, communication cost, and graph matching. *SIAM J. Appl. Math.*, **52**, 1763–1779.

Pleij,C.W.A. (1994) RNA pseudoknots. *Curr. Opin. Struct. Biol.*, **4**, 337–344.

Sammeth,M., Morgenstern,B. and Stoye,J. (2003) Divide-and-conquer multiple alignment with segment-based constraints. *Bioinformatics*, **19** (Suppl. 2), ii189–ii195.

Schuler,G.D., Altschul,S.F. and Lipman,D.J. (1991) A workbench for multiple alignment construction and analysis. *Proteins*, **9**, 180–190.

Stoye,J. (1998) Multiple sequence alignment with the divide-and-conquer method. *Gene*, **211**, GC45–GC56.

Stoye,J., Moulton,V. and Dress,A.W.M. (1997a) DCA: an efficient implementation of the divide-and-conquer approach to simultaneous multiple sequence alignment. *Comput. Appl. Biosci.*, **13**, 625–626.

Stoye,J., Perrey,S.W. and Dress,A.W.M. (1997b) Improving the divide-and-conquer approach to sum-of-pairs multiple sequence alignment. *Appl. Math. Lett.*, **10**, 67–73.

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. and Chou,W.I. (2003) Constrained multiple sequence alignment tool development and its application to RNase family alignment. *J. Bioinform. Comput. Biol.*, **1**, 267–287.

Taylor,W.R. (1987) Multiple sequence alignment by a pairwise algorithm. *Comput. Appl. Biosci.*, **3**, 81–87.

Taylor,W.R. (1994) Motif-biased protein sequence alignment. *J. Comput. Biol.*, **1**, 297–310.

Thompson,J.D., Higgs,D.G. and 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 Res.*, **22**, 4673–4680.

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

Tönges,U., Perrey,S.W., Stoye,J. and Dress,A.W.M. (1996) A general method for fast multiple sequence alignment. *Gene*, **172**, GC33–GC41.

Tsai,Y.T., Huang,Y.P., Yu,C.T. and Lu,C.L. (2004) MuSiC: a tool for multiple sequence alignment with constraints. *Bioinformatics*, (in press).

Wang,L. and Jiang,T. (1994) On the complexity of multiple sequence alignment. *J. Comput. Biol.*, **1**, 337–348.

Williams,G.D., Chang,R.-Y. and Brian,D.A. (1999) A phylogenetically conserved hairpin-type 39 untranslated region pseudoknot functions in coronavirus RNA replication. *J. Virol.*, **73**, 8349–8355.

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