Time and Space Efficient Algorithms for Constrained Sequence Alignment

Z.S. Peng and H.F. Ting

Department of Computer Science,
The University of Hong Kong, Hong Kong
{zspeng, hfting}@cs.hku.hk

Abstract. In this paper, we study the constrained sequence alignment problem, which is a generalization of the classical sequence alignment problem with the additional constraint that some characters in the alignment must be positioned at the same columns. The problem finds important applications in Bioinformatics. Our major result is an $O(\ell n^2)$-time and $O(\ell n)$-space algorithm for constructing an optimal constrained alignment of two sequences where n is the length of the longer sequence and ℓ is the length of the constraint. Our algorithm matches the best known time complexity and reduces the best known space complexity by a factor of n for solving the problem. We also apply our technique to design time and space efficient heuristic and approximation algorithm for aligning multiple sequences.

1 Introduction

In Bioinformatics, sequence alignment is a useful method for measuring the similarity of DNA sequences. By constructing an alignment of the DNA sequences of different species, Biologists may obtain important information on the evolutionary history of these species or discover conserved regions in their genomes. For the Pairwise Sequence Alignment problem (PSA), which asks for aligning only two sequences, there are polynomial-time algorithms for constructing optimal solutions [3, 11]. For the Multiple Sequence Alignment problem (MSA), which aligns more than two sequences, we know that the problem is NP-complete [10, 8], and there are many heuristics [5, 4, 7] and approximation algorithms [12, 2, 9] for constructing good, but not necessary optimal, solutions.

Existing sequence alignment programs do not allow users to use their biological knowledge to improve the quality of the alignment. For example, it is generally agreed that in the alignment of RNase sequences, three active-site residues `His12`, `Lys41` and `His119` should be aligned in the same columns. However, most sequence alignment programs mis-align these important residues. To solve this problem, Tang *et al.* [1] formulated the Constrained Sequence Alignment problem, which is a natural generalization of the classical sequence alignment problem. The new problem has an additional input of a constrained sequence, which imposes a structure on the alignment by requiring every character in the constrained sequence to appear in the same column of the alignment. They gave an algorithm for

M. Domaratzki et al. (Eds.): CIAA 2004, LNCS 3317, pp. 237–246, 2005.

the Constrained Pairwise Sequence Alignment problem (CPSA) (i.e., constructing constrained alignment of two sequences) that runs in $O(\ell n^4)$) time and uses $O(\ell n^4)$ space where n is the length of the longer input sequence and ℓ is the length of the constrained sequence. They also proposed the *progressive alignment* heuristic for the Constrained Multiple Sequence Alignment (CMSA) problem, which basically uses an optimal algorithm for CPSA to align pairs of input sequences progressively. Using the $O(\ell n^4)$-time and $O(\ell n^4)$-space algorithm for CPSA to implement the heuristic gives an $O(\ell k n^4)$ time and $O(\ell n^4)$ space complexity, where k is the number of sequences to be aligned.

In [6], Chin *et al.* gave an $O(\ell n^2)$-time and $O(\ell n^2)$-space algorithm for CPSA. When applying this algorithm to the progressive alignment heuristic, they reduced both the time and space complexity of the heuristic by a factor of n^2. They also gave an approximation algorithm, called Center-star, for CMSA which runs in $O(\ell C k^2 n^2)$ time and uses $O(\ell k^2 n^2)$ memory space, and guarantees that the distance score of the alignment returned by the algorithm is always at most $(2 - 2/k)$ times the distance score of the optimal alignment. Here, C is a constant depending on the input sequences. Experiments showed that the quality of the alignment returned by Center-Star is 15%–30% better than that of the progressive alignment heuristic.

Both the progressive alignment heuristic and Center-star are not practical because of their huge $O(\ell k^2 n^2)$ memory space requirement. The DNA sequences that we study in Bioinformatics are usually more than 1M characters long. Thus, to align four such sequences with a constrained sequence of 4 characters, we need at least $2^{16} Gb = 65536$Gb of memory. Note that a typical workstation is equipped with at most 4Gb memory.

The major result of this paper is an $O(\ell n^2)$-time and $O(\ell n)$-space algorithm for the CPSA problem. Note that we have reduced the space requirement by a factor of n without increasing the time complexity. This algorithm immediately enables us to reduce the space requirement of the progressive alignment heuristic from $O(\ell n^2)$ to $O(\ell n)$. Furthermore, we adapt our space-saving technique so as to reduce the space complexity of Center-star to $O(\ell k^2 n)$ without increasing its time complexity. These improvements are very important practically. Now, to align four DNA sequences of 1M long with a constrained sequence of 4 characters, we need only 64Mb of memory, which is well within the capability of a typical workstation.

The organization of this paper is as follows. In Section 2, we give the definitions and notations that are used in our discussion. In Sections 3 and 4, we describe our algorithm for the CPSA problem and analyze its time and space complexity. We show how to adapt our technique to reduce the space complexity of Center-Star in Section 5.

2 Definitions and Notations

Let Σ be a finite set of characters which does not include '␣', the space character. We are given a distance function $\delta : (\Sigma \cup \{␣\}) \times (\Sigma \cup \{␣\}) \rightarrow \Re$ such

that any two characters a, b in $\Sigma \cup \{\_\}$ have distance $\delta(a, b)$. We assume that $\delta(\_, \_) = 0$. (Intuitively, the distance $\delta(a, b)$ measures the *mutation* distance between characters a, b.) For any sequence $S = S[1]S[2] \cdots S[n]$ over Σ, let $|S|$ denote its length n, and for any $1 \leq i \leq j \leq n$, let $S[i..j]$ denote the substring $S[i]S[i+1] \cdots S[j]$. To simplify our discussion, we let $S[i..j]$ be the empty sequence if $i > j$.

Let S and S' be any two sequences over Σ. A *sequence alignment* of S and S' is given by an alignment matrix, which has two rows and $w \geq \max\{|S|, |S'|\}$ columns, such that when we remove all the spaces in the first (resp. second) row, we get S (resp. S'). Let P be a common subsequence of S and S' (i.e., P is a subsequence of S and is also a subsequence of S'). A *constrained sequence alignment* (CSA) of S and S' with respect to P is an alignment A of S and S' with the following additional property: there are $|P|$ columns $c_1, c_2, \ldots c_{|P|}$ in A such that for all $1 \leq j \leq |P|$, we have $A[1, c_j] = A[2, c_j] = P[j]$. Figure 1 gives an example.

```
S1: IN-YRWRCKNQN--LRTTFANV-N-CGNQSIRCPHNRT--NCHRSR--VPLLHCDL--P
S2: INNYQ-RCKNQNTFLL-TF-NVVNVCGNPN--CPSNKTRKNCH-SGSQVP--HCNLTTP

P: CKCCC
```

Fig. 1. An example on constrained sequence alignment

Define $\delta(A) = \sum_{1 \leq j \leq w} \delta(A[1, j], A[2, j])$ to be the *score* of the alignment A. We say that A is optimal if it has the smallest score among all CSAs of S and S' with respect to P. We let $\delta_{\mathtt{opt}}(S, S', P)$ denote the score of an optimal CSA. To unify our discussion, we let $\delta_{\mathtt{opt}}(S, S', P) = \infty$ if P is not a common subsequence of S and S'.

We generalize sequence alignment to multiple sequences naturally. Consider $k > 2$ sequences $S_1, S_2, \ldots, S_k$ over Σ. A sequence alignment of $S_1, S_2, \ldots, S_k$ is given by an alignment matrix of k rows and $w \geq \max\{|S_1|, |S_2|, \ldots, |S_k|\}$ columns such that for any $1 \leq i \leq k$, if we remove all the space characters in row i, we get the sequence S_i. Let P be a common subsequence of $S_1, S_2, \ldots, S_k$. A constrained sequence alignment (CSA) of $S_1, S_2, \ldots, S_k$ with respect to P is an alignment A of $S_1, S_2, \ldots, S_k$ with the following property: there are $|P|$ columns $c_1, c_2, \ldots c_{|P|}$ in A such that for all $1 \leq j \leq |P|$, we have $A[1, c_j] = A[2, c_j] = \cdots = A[k, c_j] = P[j]$. Define the *sum-of-pair* score of A, or simply the score of A, to be

$$\delta(A) = \sum_{1 \leq p < q \leq k} \sum_{1 \leq j \leq w} \delta(A[p, j], A[q, j]).$$

We say that A is optimal if $\delta(A)$ has the smallest value. The score of an optimal CSA is denoted as $\delta_{\mathtt{opt}}(S_1, S_2, \ldots, S_k; P)$.

3 Two Useful Formulas

Let $S = S[1..m]$ and $S' = S'[1..n]$ be two sequences of m and n characters over Σ. Let $P = P[1..\ell]$ be a common subsequence of S and S'. In the next two sections, we describe an algorithm for computing an optimal CSA of S and S' with respect to P. Our algorithm is recursive and it needs to compute all optimal CSAs of $S[1..i]$ and $S'[1..j]$ with respect to $P[1..k]$ for some i, j, k. It also needs to compute all optimal CSAs of $S[i..m]$ and $S'[j..n]$ with respect to $P[k..\ell]$. Below, we state two useful formulas that are important for us to find these optimal alignments efficiently.

For any $0 \le i \le m$, $0 \le j \le n$, and $0 \le k \le \ell$, let $D(i, j, k)$ be the score of an optimal alignment of $S[1..i]$ and $S'[1..j]$ with respect to $P[1..k]$. In other words, $D(i, j, k) = \delta_{\mathtt{opt}}(S[1..i], S'[1..j]; P[1..k])$. Recall that $\delta_{\mathtt{opt}}(S[1..i], S'[1..j]; P[1..k]) = \infty$ if $P[1..k]$ is not a common subsequence of $S[1..i]$ and $S'[1..j]$. In [6], Chin *et al.* gave a formula that relates with different $D(i, j, k)$:
Formula I:

$$D(i,j,k) = \min \begin{cases} D(i-1,j-1,k-1) + \delta(S[i], S'[j]) & \text{if } S[i] = S'[j] = P[k], \\ D(i-1,j-1,k) + \delta(S[i], S'[j]) & \text{if } i, j \ge 1, \\ D(i-1,j,k) + \delta(S[i], \text{␣}) & \text{if } i \ge 1, \\ D(i,j-1,k) + \delta(\text{␣}, S'[j]) & \text{if } j \ge 1. \end{cases}$$

Furthermore, we have $D(0, 0, 0) = 0$, and $D(i, 0, k) = \infty$ and $D(0, j, k) = \infty$ for all $0 \le k \le \ell$, $0 \le i \le m$, and $0 \le j \le n$. (Recall that ␣ is the space character.)

The above formula is useful for computing the score of alignment for sequences $S[1..i]$, $S'[1..j]$, $P[1..k]$. To handle alignments for sequences $S[i..m]$, $S'[j..n]$, $P[k..\ell]$, we need the following lemma.

Lemma 1. *For any* $1 \le i \le m+1$, $1 \le j \le n+1$, *and* $1 \le k \le \ell+1$, *let* $Q(i, j, k) = \delta_{\mathtt{opt}}(S[i..m], S'[j..n]; P[k..\ell])$. *We have*
Formula II:

$$Q(i,j,k) = \min \begin{cases} Q(i+1,j+1,k+1) + \delta(S[i], S'[j]) & \text{if } S[i] = S'[j] = P[k], \\ Q(i+1,j+1,k) + \delta(S[i], S'[j]) & \text{if } i \le m \text{ and } j \le n, \\ Q(i+1,j,k) + \delta(S[i], \text{␣}) & \text{if } i \le m, \\ Q(i,j+1,k) + \delta(\text{␣}, S'[j]) & \text{if } j \le n, \end{cases}$$

with the boundary conditions (i) $Q(m+1, n+1, \ell+1) = 0$, *and (ii)* $Q(i, n+1, k) = \infty$ *and* $Q(m+1, j, k) = \infty$ *for all* $1 \le k \le \ell+1$, $1 \le i \le m+1$, *and* $1 \le j \le n+1$.

Proof. We consider the construction of an optimal constrained sequence alignment A of $S[i..m]$ and $S'[j..n]$ with respect to $P[k..\ell]$. Note that $\delta(A) = Q[i, j, k]$. There are four possibilities:

- If $S[i] = S'[j] = P[k]$, A can align $S[i], S'[j]$ and $P[k]$ at the same column, and the remaining columns of A form an optimal alignment of $S[i + 1..m]$ and $S'[j + 1..n]$ with respect to $P[k + 1..\ell]$. In this case, $\delta(A) = Q[i + 1, j + 1, k + 1] + \delta(S[i], S'[j])$.

- If $i \leq m, j \leq n$, A can align $S[i]$ and $S'[j]$ at the same column, and the remaining columns of A form an optimal alignment of $S[i{+}1..m]$ and $S'[j{+}1..n]$ with respect to $P[k..\ell]$; in this case, $\delta(A) = Q[i+1, j+1, k] + \delta(S[i], S'[j])$.
- If $i \leq m$, then A can align $S[i]$ with a space, and the remaining columns of A form an optimal alignment of $S[i+1..m]$ and $S'[j..n]$ with respect to $P[k..\ell]$; in this case, $\delta(A) = Q(i+1, j, k) + \delta(S[i], \textvisiblespace)$.
- If $j \leq n$, then A can align $S'[j]$ with a space, and the remaining columns of A form an optimal alignment of $S[i..m]$ and $S'[j+1..n]$ with respect to $P[k..\ell]$; in this case, $\delta(A) = Q(i, j+1, k) + \delta(\textvisiblespace, S'[j])$.

Obviously, $\delta(A)$ must be equal to the minimum of these four values. □

4 An Optimal CSA Algorithm for Two Sequences

In this section, we describe an algorithm for constructing an optimal alignment of $S = S[1..m]$ and $S' = S'[1..n]$ with respect to $P = P[1..\ell]$. We need the following lemma, which gives a structural property about the alignment.

Lemma 2. *Let i be any integer in $[1, m]$, the set of integers between 1 and m. Then $\delta_{\mathtt{opt}}(S[1..m], S'[1..n], P[1..\ell])$ is equal to*

$$\min_{\substack{0 \leq j \leq n \\ 0 \leq k \leq \ell}} \{\delta_{\mathtt{opt}}(S[1..i], S'[1..j]; P[1..k]) + \delta_{\mathtt{opt}}(S[i+1..m], S[j+1..n]; P[k+1..\ell])\}$$

Proof. Consider an optimal CSA A of S and S' with respect to P. Recall that A has two rows, and after throwing all the space characters in the first row, we get S. Suppose the ith character of S, i.e., $S[i]$, is at the pth column of the first row. In other words, $S[i] = A[1, p]$. Now, we consider S' and the second row of A. Suppose that $S'[1..j]$ falls into the first p columns of the second row. Furthermore, suppose that $c_1, c_2, \ldots, c_\ell$ are the ℓ columns in the CSA A such that $A[1, c_h] = A[2, c_h] = P[h] (1 \leq h \leq \ell)$, and that either $k = 0$ or $1 \leq k \leq p \leq \ell$. Then, the first p columns of A form a CSA A_1 of $S[1..i]$ and $S'[1..j]$ with respect to $P[1..k]$, and the remaining columns form a CSA A_2 of $S[i{+}1..m]$ and $S[j{+}1..n]$ with respect to $P[k+1..\ell]$. Because of the optimality of A, we conclude that A_1 and A_2 must be optimal. □

We give below our recursive algorithm for finding an optimal CSA of S and S' with respect to P.

```
Algorithm CSA(S[1..m], S'[1..n], P[1..ℓ])
begin
S1: Find the pair (j, k) ∈ [1, n] × [1, ℓ] such that δ_opt(S[1..m/2], S'[1..j]; P[1..k])
    +δ_opt(S[m/2 + 1..m], S'[j + 1..n]; P[k + 1..ℓ]) is minimum;
S2: Call recursively CSA(S[1..m/2], S'[1..j], P[1..k]) to find an optimal alignment
    A_1 of S[1..m/2] and S'[1..j] with respect to P[1..k].
S3: Call recursively CSA(S[m/2+1..m], S'[j+1..n], P[k+1..ℓ]) to find an optimal
    alignment A_2 of S[m/2 + 1..m] and S'[j + 1..n] with respect to P[k + 1..ℓ].
S4: Return A_1A_2.
end
```

Note that Steps S2, S3 and S4 are straightforward. The following lemma gives the details of Step S1.

Lemma 3. *We can finish Step 1 using $O(\ell mn)$ time and $O(\ell n)$ space.*

Proof. Recall that in Section 3, we have $D(i,j,k) = \delta_{\mathtt{opt}}(S[1..i], S'[1..j]; P[1..k])$ and $Q(i,j,k) = \delta_{\mathtt{opt}}(S[i..m], S'[j..n]; P[k..\ell])$. For any $0 \le p \le m$, let

$$D[p,*,*] = \{D[i,j,k] \mid i = p, 0 \le j \le n, 0 \le k \le \ell\}.$$

For any $1 \le p \le m+1$, let

$$Q[p,*,*] = \{Q[i,j,k] \mid i = p, 1 \le j \le n+1, 1 \le k \le \ell+1\}.$$

It is easy to see that if we are given $D[m/2,*,*]$ and $Q[m/2+1,*,*]$, we can find the pair (j,k) required in Step 1 using $O(\ell n)$ time.

Note that we know all the values in $D[0,*,*]$, and from Formula I, we know that for any $p > 0$, we can compute $D[p,j,k]$ from $D[p-1,j-1,k-1]$, $D[p-1,j-1,k]$, $D[p,j-1,k]$ and $D[p-1,j,k]$ in constant time. Thus, we can compute $D[1,*,*]$ from $D[0,*,*]$, and in general, $D[p,*,*]$ from $D[p-1,*,*]$ by applying Formula I to find sequentially $D[p,0,0]$, $D[p,2,0], \ldots, D[p,1,1], D[p,1,2]$, $\ldots, D[p,n,\ell]$. The total time taken is $O(\ell n)$. It follows that we can compute $D[m/2,*,*]$ iteratively from $D[0,*,*]$, $D[1,*,*], \ldots, D[m/2-1,*,*]$ using totally $O(\ell mn)$ time. Since we can reuse the space after each iteration, the total space needed is $O(\ell n)$.

Similarly, we can apply Formula II to find $Q[m/2+1,*,*]$ using the same time and space complexity. The lemma follows. □

Now, we are ready to analyze the time and space complexity of our algorithm.

Theorem 1. *The algorithm* `CSA` *runs in $O(\ell mn)$ time and uses $O(\ell n)$ space.*

Proof. Let $T(m,n,\ell)$ and $S(m,n,\ell)$ be the worst case time and space complexity of `CSA` for finding an optimal constrained sequence alignment of two sequences with length m and n with respect to a common subsequence with length ℓ. First, we analyse the space complexity. We will prove by induction that $S(m,n,\ell) = O(\ell n)$. By Lemma 3, Step 1 uses $O(\ell n)$ space. Step 2 and Step 3 use respectively $S(m/2,j,k)$ and $S(m/2,n-j,\ell-k)$ space. Note that we can reuse the space after each step. Hence, we have

$$S(m,n,\ell) \le \max\{O(\ell n), O(kj), O((\ell-j)(n-j))\} = O(\ell n).$$

Now, we consider the time. By Lemma 3, Step 1 takes at most $c\ell mn$ time for some constant c. We prove below by induction that the running time of the whole algorithm is at most double the running time of Step 1. In other words, $T(m,n,\ell) \le 2cmn\ell$.

As mentioned above, Step 1 runs in $c\ell mn$ time. Steps 2 and 3 call `CSA` recursively, each take $T(m/2, n, \ell)$ time. Hence, we have the following recurrence:

$$\begin{aligned} T(m,n,\ell) &\le c\ell mn + T(m/2, j, k) + T(m/2, n-j, \ell-k) \\ &\le c\ell mn + 2c \cdot \frac{m}{2} jk + 2c \cdot \frac{m}{2}(n-j)(\ell-k) \\ &\le (c + 2 \cdot \frac{c}{2}) mn\ell \\ &\le 2cmn\ell. \end{aligned}$$

The theorem is proved. □

5 An Approximating CSA Algorithm for Multiple Sequences

In this section, we describe how to adapt our space-saving technique to reduce the space complexity of the Center-star approximation algorithm of Chin *et al.*, which constructs a constrained sequence alignment of $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ with respect to P in $O(\ell C k^2 n^2)$ time and $O(\ell k^2 n^2)$ space, where $\ell = |P|$, C is some constant depending on input, and $n = \max\{|S_1|, |S_2|, \ldots, |S_k|\}$. The alignment constructed by the algorithm is guaranteed to have a score no greater than $(2 - 2/k)$ times that of an optimal alignment. We show below how to implement this algorithm such that the space complexity is reduced to $O(\ell k^2 n)$ while the time complexity is still $O(\ell C k^2 n^2)$.

5.1 The Center-Star Algorithm

To describe the Center-star algorithm, we need some definitions. Let S be any sequence in $\mathcal{S}$. Let S' be another sequence, and A be a constrained sequence alignment of S and S' with respect to P. We say that in A, the ith character of S is aligned with the jth character of P if at some column p of A, $S[i] = A[1,p] = A[2,p] = P[j]$ and after removing all spaces in the first row of A, $A[1,p]$ becomes the ith character of S. We say that A aligns S to P at positions $1 \le c_1 < c_2 < \cdots < c_\ell$ if for all $1 \le j \le \ell$, the c_jth character of S is aligned with the jth character of P in A. We let $A_{\mathtt{opt}}(S, S', P, (c_1, c_2, \ldots, c_\ell))$ denote the optimal CSA of S and S' with respect to P that aligns S with P at $(c_1, c_2, \ldots, c_\ell)$.

Now, we are ready to describe the Center-star algorithm. To find a CSA of $S_1, S_2, \ldots, S_k$ with respect to P, it executes the following two steps:

1. Find $S^* \in \mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ and list of positions $(c_1, c_2, \ldots, c_\ell)$ such that $\sum_{S' \in \mathcal{S}} \delta(A_{\mathtt{opt}}(S^*, S', P, (c_1, c_2, \ldots, c_\ell)))$ is minimum.
2. Merging the $k-1$ alignments $A_{\mathtt{opt}}(S^*, S', P, (c_1, c_2, \ldots, c_\ell))$ $(S' \in \mathcal{S} - S^*)$ into an alignment of sequences $S_1, S_2, \ldots, S_k$ with respect to P by adding spaces at the appropriate positions.

It is easy to see that the most time and space consuming computation in the algorithm is to find $A_{\mathtt{opt}}(S, S', P, (c_1, c_2, \ldots, c_\ell))$. Chin *et al.* showed that

such optimal alignment can be found in $O(\ell n^2)$ time and $O(\ell n^2)$ space. (Recall that $n = \max\{|S_1|, |S_2|, \ldots, |S_k|\}$.) In the next section, we apply our technique to reduce the space complexity from $O(\ell n^2)$ to $O(\ell n)$, while keeping the time complexity to be $O(\ell n^2)$. It follows that our implementation reduces the overall space complexity of the Center-star algorithm from $O(\ell k^2 n^2)$ to $O(\ell k^2 n)$ without increasing the time complexity. We refer to [6] for more details on the complexity analysis of the Center-star algorithm.

5.2 Reducing the Space Complexity

Let $S = S[1..m]$, $S' = S'[1..n]$ and $P = P[1..\ell]$. Let $\delta_{\mathtt{opt}}(S, S', P, (c_1, c_2, \ldots, c_\ell))$ denote the score of the optimal constrained sequence alignment of S and S' with respect to P that aligns S to P at position $(c_1, c_2, \ldots, c_\ell)$. To simplify discussion, we let $\delta_{\mathtt{opt}}(S, S', P, (c_1, c_2, \ldots, c_\ell)) = \infty$ if no such alignment is possible. We need the following lemma.

Lemma 4. *Let* $i \in [1, m]$. $\delta_{\mathtt{opt}}(S[1..m], S[1..n], P[1..\ell], (c_1, c_2, \ldots, c_\ell))$ *equals*

$$\min_{0 \le j \le n, 0 \le k \le \ell} \delta_{\mathtt{opt}}(S[1..i], S[1..j], P[1..k], (c_1, c_2, \ldots, c_k)) + \delta_{\mathtt{opt}}(S[i+1..m], S[j+1..n], P[k+1..\ell], (c_{k+1}, c_{k+2}, \ldots, c_\ell))$$

Proof. Similar to the proof of Lemma 2. □

The above lemma suggests immediately the a recursive algorithm for finding $A_{\mathtt{opt}}(S, S', P, (c_1, c_2, \ldots, c_\ell))$, which is given in Figure 2. To execute Step S1 effi-

```
Algorithm CSAP(S[1..m], S'[1..n], P[1..ℓ], (c_1, c_2, ..., c_ℓ))
begin
S1: Find the j, k such that the sum of δ_opt(S[1..m/2], S'[1..j], P[1..k], (c_1, ..., c_k))
    and δ_opt(S[m/2 + 1..m], S'[j + 1..n], P[k + 1..ℓ], (c_{k+1}, ..., c_ℓ)) is minimum;
S2: Call recursively CSAP(S[1..m/2], S'[1..j], P[1..k], (c_1, ..., c_k)) to find the optimal alignment A_1
S3: Call recursively CSAP(S[m/2+1..m], S'[j+1..n], P[k+1..ℓ], (c_{k+1}, ..., c_ℓ)) to find the optimal alignment A_2
S4: Return A_1 A_2.
end
```

Fig. 2. Algorithm for constructing an optimal CSA with position constraints

ciently, we need two formulas similar to Formulas I and II. The first one is given in [6].

For any $0 \le i \le m$, $0 \le j \le n$, $0 \le k \le \ell$, let

$$D'(i, j, k) = \delta_{\mathtt{opt}}(S[1..i], S'[1..j], P[1..k], (c_1, c_2, \ldots, c_k)).$$

For any $1 \leq i \leq m+1$, $1 \leq j \leq n+1$, $1 \leq k \leq \ell+1$, let

$$Q'(i,j,k) = \delta_{\mathtt{opt}}(S[i..m], S'[j..n], P[k..\ell], (c_k, c_{k+1}, \ldots, c_\ell)).$$

We have

$$D'(i,j,k) = \min \begin{cases} D'(i-1,j-1,k-1) + \delta(S[i], S'[j]) & \text{if } c_k = i \text{ and} \\ & S[i] = S'[j] = P[k], \\ D'(i-1,j-1,k) + \delta(S[i], S'[j]) & \text{if } i,j \geq 1, \\ D'(i-1,j,k) + \delta(S[i], \sqcup) & \text{if } i \geq 1, \\ D'(i,j-1,k) + \delta(\sqcup, S'[j]) & \text{if } j \geq 1. \end{cases}$$

and

$$Q'(i,j,k) = \min \begin{cases} Q'(i+1,j+1,k+1) + \delta(S[i], S'[j]) & \text{if } c_k = i \text{ and} \\ & S[i] = S'[j] = P[k], \\ Q'(i+1,j+1,k) + \delta(S[i], S'[j]) & \text{if } i \leq m \text{ and } j \leq n, \\ Q'(i+1,j,k) + \delta(S[i], \sqcup) & \text{if } i \leq m, \\ Q'(i,j+1,k) + \delta(\sqcup, S'[j]) & \text{if } j \leq n, \end{cases}$$

with the boundary conditions

- $D'(0,0,0) = 0$, $D'(i,0,k) = \infty$ and $D'(0,j,k) = \infty$ for all $0 \leq k \leq \ell$, $0 \leq i \leq m$, $0 \leq j \leq n$, and
- $Q'(m+1,n+1,\ell+1) = 0$, and $Q'(i,n+1,k) = \infty$ and $Q'(m+1,j,k) = \infty$ for all $1 \leq k \leq \ell+1$, $1 \leq i \leq m+1$, $1 \leq j \leq n+1$.

Given these formulas, we can apply the technique used in Section 4 to implement Step 1 of `CSAP` efficiently.

Theorem 2. *We can finish Step 1 of* `CSAP` *in* $O(\ell n^2)$ *time and* $O(\ell n)$ *space. Furthermore,* `CSAP` *runs in* $O(\ell n^2)$ *time and used* $O(\ell n)$ *space.*

Proof. Similar to the proofs of Lemma 3 and Theorem 1. □

References

1. C.Y. Tang, C.L. Lu, M.D.T. Chang, Y.T. Tsai, Y.J. Sun, K.M. Chao, J.M. Chang, Y.H. Chiou, C.M. Wu, H.T. Chang, and W.I. Chou. Constrained multiple sequence alignment tool development and its application to RNase family alignment. In *Proceedings of the First IEEE Computer Society Bioinformatics Conference*, pages 127–137, 2002.
2. D. Gusfield. Efficient methods for multiple sequence alignment with guaranteed error bounds. *Bulletin of Mathematical Biology*, 30:141–154, 1993.
3. D. Gusfield. *Algorithms on strings, trees, and sequence.* Cambridge University Press, British, 1999.
4. D. Higgins and P. Sharpe. CLUSTAL: a package for performing multiple sequence alignment on a microcomputer. *Gene*, 73:237–244, 1988.

5. F. Corpet. Multiple sequence alignment with hierarchical clustering. *Nucleic Acids Research*, 16:10881–10890, 1988.
6. F.Y.L. Chin, N.L. Ho, T.W. Lam, W.H. Wong, and M.Y. Chan. Efficient constrained multiple sequence alignment with performance guarantee. In *Proceedings of the IEEE Computational Systems Bioinformatics Conference*, pages 337–346, 2003.
7. J.D. Thompson, D.G. Higgins, and T.J. Gibson. 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(22):4673–4680, 1994.
8. L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1:337–348, 1994.
9. P.A. Pevzner. Multiple alignment, communication cost, and graph matching. *SIAM Journal on Applied Mathematics*, 52:1763–1779, 1992.
10. P. Bonizzoni and G.D. Vedova. The complexity of multiple sequence alignment with *SP*-score that is a metric. *Theoretical Computer Science*, 259:63–79, 2001.
11. S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Evolution*, 48:443–453, 1970.
12. V. Bafna, E.L. Lawler, and P.A. Pevzner. Approximation algorithms for multiple sequence alignment. *Theoretical Computer Science*, 182:233–244, 1997.