

A Linear-Space Algorithm for the Substring Constrained Alignment Problem

Yoshifumi Sakai^(✉)

Graduate School of Agricultural Science, Tohoku University,
1-1, Amamiyamachi, Tsutsumidori, Aobaku, Sendai 981-8555, Japan
`sakai@biochem.tohoku.ac.jp`

Abstract. In a string similarity metric adopting affine gap penalties, we propose a quadratic-time, linear-space algorithm for the following constrained string alignment problem. The input of the problem is a pair of strings to be aligned and a pattern given as a string. Let an occurrence of the pattern in a string be a minimal substring of the string that is most similar to the pattern. Then, the output of the problem is a highest-scoring alignment of the pair of strings that matches an occurrence of the pattern in one string and an occurrence of the pattern in the other, where the score of the alignment excludes the similarity between the matched occurrences of the pattern. This problem may arise when we know that each of the strings has exactly one meaningful occurrence of the pattern and want to determine a putative pair of such occurrences based on homology of the strings.

1 Introduction

Constructing a highest-scoring alignment is a common way to analyze how two strings are similar to each other [7], because it is well known that, using the dynamic programming technique, we can obtain such an alignment of an arbitrary m -length string A and an arbitrary n -length string B in $O(mn)$ time [10]. As a more appropriate analysis of the similarity in the case where we know that a common pattern string P occurs both in A and B and that these occurrences should be matched in the alignment, Tsai [12] proposed the constrained longest common subsequence (LCS) problem. This problem consists of finding an arbitrary LCS containing P as a subsequence, where an LCS can be thought of as a highest-scoring alignment in a certain simple similarity metric. Chin et al. [4] showed that this problem is solvable in $O(mnr)$ time and $O(nr)$ space, where r is the length of P and $m \geq n \geq r$. Recently, as one of the generalized constrained LCS problems, Chen and Chao [2] proposed the STR-IC-LCS problem, which consists of finding an arbitrary LCS of A and B that contains P as a substring, instead of as a subsequence. Deorowicz [5] showed that this problem is solvable in $O(mn)$ time and $O(mn)$ space. The difference between the alignments found in these problems is whether the score of the alignment takes the similarity between the matched occurrences of P in X and Y into account or not. The STR-IC-LCS problem may arise when we know that each of the strings

has exactly one meaningful occurrence of the pattern and want to determine a putative pair of such occurrences based on homology of the strings.

In comparing strings over an alphabet set with various levels of symbol similarity, such as amino acid sequences of proteins, however, the LCS metric is sometimes too naive to adopt as a similarity metric. In the present article, we consider generalized similarity metrics, including the metric based on an amino acid substitution matrix with affine gap penalties, which is widely used to estimate the similarity between amino acid sequences [7]. This similarity metric is also adopted by another generalized constrained LCS problem, the regular expression constrained alignment problem [1, 3, 9], in which the pattern P is given as a regular expression.

The present article propose an $O(mn)$ -time, $O(n)$ -space algorithm for the problem consisting of finding a highest-scoring alignment of A and B that matches an occurrence of P in A and an occurrence of P in B . In this problem, we treat an arbitrary minimal substring of a string most similar to P as an occurrence of P in the string and ignore the similarity between the matched occurrences of P when estimating the score of the alignment. The proposed algorithm achieves the same asymptotic execution time and required space as the algorithm for the (non-constrained) alignment problem based on the divide-and-conquer technique of Hirschberg [8]. Furthermore, since the problem we consider is identical to the STR-IC-LCS problem if we adopt the LCS metric, the proposed algorithm improves space complexity of the STR-IC-LCS problem achieved by the algorithm of Deorowicz [5] from quadratic to linear.

2 Preliminaries

A string is a sequence of symbols. For any string X , $|X|$ denotes the length of X , $X[i]$ denotes the symbol in X at position i , and $X(i', i]$ denotes the substring of X at position between $i' + 1$ and i . The concatenation of string X' followed by string X'' is denoted by $X'X''$.

Let Σ be an alphabet set of a constant number of symbols. Let $-$ denote a gap symbol that does not belong to Σ . A gap is a string consisting only of more than zero gap symbols. We use $+$ and $/$ to represent the first and last gap symbols in a gap of length more than one, respectively, and $*$ to represent the only gap symbol in a gap of length one. In what follows, we use $-$ to represent a gap symbol in a gap of length more than two other than the first and last gap symbols. Let $\Gamma = \{+, -, /, *\}$ and let $\tilde{\Sigma} = \Sigma \cup \Gamma$. Let a gapped string of a string X over Σ be a string over $\tilde{\Sigma}$ obtained from X by inserting a concatenation of zero or more gaps at position between i and $i + 1$ for each index i with $0 \leq i \leq |X|$. Although concatenations of two or more gaps inserted in a string may look uncommon, we adopt this definition of a gapped string for a technical reason mentioned later. We sometimes use the index representation, denoted $I_{\tilde{X}}$, of a gapped string \tilde{X} of a substring of X , in which $X[i]$ is represented as index i and any gap symbol γ in Γ that appears in the concatenation of gaps inserted in X at position between i and $i + 1$ is represented as γ with subscript i .

For any strings X and Y over Σ , an alignment of X and Y is a pair of a gapped string \tilde{X} of X and a gapped string \tilde{Y} of Y with $|\tilde{X}| = |\tilde{Y}|$ such that $\tilde{X}[q]$ or $\tilde{Y}[q]$ is not a gap symbol in Γ for any index q with $1 \leq q \leq |\tilde{X}|$ ($= |\tilde{Y}|$). Let a symbol similarity score table s consist of values $s(a, b)$ indicating how much a is similar to b for all ordered pair (a, b) of symbols in $\tilde{\Sigma}$ other than pairs of gap symbols in Γ . A typical setting, adopted in affine gap penalty metrics, is $s(a, +) = s(a, *) = s(+, a) = s(*, a) = \text{gip} + \text{gep}$ and $s(a, -) = s(a, /) = s(-, a) = s(/, a) = \text{gep}$ for any symbol a in Σ , where gip is a gap insertion penalty representing the penalty for each insertion of a gap and gep is a gap extension penalty representing the penalty for each one-symbol extension of a gap. How well an alignment (\tilde{X}, \tilde{Y}) makes a connection between symbols in X and symbols in Y is estimated by the score $s(\tilde{X}, \tilde{Y}) = \sum_{1 \leq q \leq |\tilde{X}|} s(\tilde{X}[q], \tilde{Y}[q])$ of the alignment. For any strings X and Y over Σ , let how much X is similar to Y be defined as $\text{Sim}(X, Y) = \max_{(\tilde{X}, \tilde{Y})} s(\tilde{X}, \tilde{Y})$, where (\tilde{X}, \tilde{Y}) ranges over all alignments of X and Y . We define an occurrence of a pattern in a string as a minimal substring of the string that is most similar to the patten in the sense of the following definition.

Definition 1. For any strings X and Y over Σ , let a substring X' of X be an occurrence of Y in X if $\text{Sim}(X', Y) \geq \text{Sim}(X'', Y)$ for any substring X'' of X and $\text{Sim}(X', Y) > \text{Sim}(X'', Y)$ for any substring X'' of X' with $|X''| < |X'|$.

The present article considers the following problem.

Definition 2. Given strings, A of length m , B of length n , and P of length r , over Σ with $m \geq n \geq r$, let the substring constrained alignment (StrCA) problem consist of finding an arbitrary pair of an occurrence A_{occ} of P in A and an occurrence B_{occ} of P in B such that

$$\text{Sim}(A_{\text{pref}}, B_{\text{pref}}) + \text{Sim}(A_{\text{suff}}, B_{\text{suff}})$$

is maximum, where $A = A_{\text{pref}}A_{\text{occ}}A_{\text{suff}}$ and $B = B_{\text{pref}}B_{\text{occ}}B_{\text{suff}}$. (If arbitrary highest-scoring alignments of A_{pref} and B_{pref} and of A_{suff} and B_{suff} are necessary after the StrCA problem is solved, we can obtain such alignments in $O(mn)$ time and $O(n)$ space based on the divide-and-conquer technique of Hirschberg [8].)

3 Algorithm

This section proposes an $O(mn)$ -time, $O(n)$ -space algorithm for the StrCA problem. In order to design the proposed algorithm, we introduce several lemmas each with no proof, due to limitation of space. However, they can be proven easily in a straightforward manner.

The algorithm we propose is based on the dynamic programming technique. We use edge-weighted directed acyclic graphs (DAGs) to represent dynamic programming (DP) tables as follows.

Definition 3. Let G be an arbitrary edge-weighted DAG. For any edge e in G , let $w(e)$ denote the weight of e . We also use $w(u, v)$ to denote $w(e)$ if e is from vertex u to vertex v . For any path π in G , let the weight $w(\pi)$ of π be the sum of $w(e)$ over all edges e in π . For any vertex v in G , let $to(v)$ denote the set of all vertices u such that G has an edge from u to v . If no such vertices u exist, then v is a source vertex. Any vertex u not appearing in $to(v)$ for any vertex v in G is a sink vertex. We focus only on edge-weighted DAGs having exactly one source vertex and one sink vertex. For any vertex v in G , we use $dp(v)$ to denote the value of v in the DP table with respect to G . This value is defined recursively as $dp(v) = 0$, if v is the source vertex, or $dp(v) = \max_{u \in to(v)} (dp(u) + w(u, v))$, otherwise. Hence, $dp(v)$ represents the weight of any heaviest path from the source vertex to v .

To solve the StrCA problem, we utilize an edge-weighted DAG, called the StrCA DAG, that reduces the StrCA problem to the problem of finding an arbitrary one of certain edges through which a heaviest path from the source vertex to the sink vertex passes. Applying the same idea as the algorithm of Deorowicz [5] for the STR-IC-LCS problem to this DAG, we can immediately obtain an algorithm for the StrCA problem. However, as mentioned later, the algorithm proposed in the present article uses this DAG in a different way in order to save a great deal of space required.

The StrCA DAG is defined as a certain variant of the following edge-weighted DAG, called the alignment DAG, which is based on an idea similar to the algorithm of Gotoh [6] for the alignment problem with affine gap penalties. This DAG is designed such that any two-edge path corresponds to a pair of consecutive positions in some alignment of two strings and vice versa. The reason for the uncommon definition of a gapped string is because of a close relationship between paths in the DAG and alignments of substrings of the strings.

Definition 4. For any strings X and Y over Σ , let the alignment DAG, denoted $G(X, Y)$, for X and Y be the edge-weighted DAG consisting of vertices

- $d(i, j)$ for all index pairs (i, j) with $0 \leq i \leq |X|$ and $0 \leq j \leq |Y|$,
- $h(i, j)$ for all index pairs (i, j) with $0 \leq i \leq |X|$ and $0 < j < |Y|$, and
- $v(i, j)$ for all index pairs (i, j) with $0 < i < |X|$ and $0 \leq j \leq |Y|$

and edges

- $e(i, j)$ of weight $s(X[i], Y[j])$ from $d(i-1, j-1)$ to $d(i, j)$,
- $e(+_i, j)$ of weight $s(+, Y[j])$ from $d(i, j-1)$ to $h(i, j)$,
- $e(-_i, j)$ of weight $s(-, Y[j])$ from $h(i, j-1)$ to $h(i, j)$,
- $e(/_i, j)$ of weight $s(/, Y[j])$ from $h(i, j-1)$ to $d(i, j)$,
- $e(*_i, j)$ of weight $s(*, Y[j])$ from $d(i, j-1)$ to $d(i, j)$,
- $e(i, +_j)$ of weight $s(X[i], +)$ from $d(i-1, j)$ to $v(i, j)$,
- $e(i, -_j)$ of weight $s(X[i], -)$ from $v(i-1, j)$ to $v(i, j)$,
- $e(i, /_j)$ of weight $s(X[i], /)$ from $v(i-1, j)$ to $d(i, j)$, and
- $e(i, *_j)$ of weight $s(X[i], *)$ from $d(i-1, j)$ to $d(i, j)$

for all possible index pairs (i, j) . Let the i th row of $G(X, Y)$ consist of all vertices $d(i, j)$ with $0 \leq j \leq |Y|$, $h(i, j)$ with $0 < j < |Y|$, and $v(i, j)$ with $0 \leq j \leq |Y|$.

Lemma 1. *Any path $\pi = e(\tilde{i}_1, \tilde{j}_1)e(\tilde{i}_2, \tilde{j}_2) \cdots e(\tilde{i}_p, \tilde{j}_p)$ in $G(X, Y)$ from $d(i', j')$ to $d(i, j)$ bijectively corresponds to the alignment (\tilde{X}, \tilde{Y}) of $X[i' + 1..i]$ and $Y[j' + 1..j]$ with $I_{\tilde{X}} = \tilde{i}_1\tilde{i}_2 \cdots \tilde{i}_p$ and $I_{\tilde{Y}} = \tilde{j}_1\tilde{j}_2 \cdots \tilde{j}_p$. Furthermore, for any such pair of a path π and an alignment (\tilde{X}, \tilde{Y}) , $w(\pi) = s(\tilde{X}, \tilde{Y})$ holds.*

Before presenting the StrCA DAG, we show that all occurrences of a pattern in a string can be found in quadratic time and linear space, if we use the following variant of the alignment DAG. This DAG is based on an idea similar to the algorithm of Smith and Waterman [11] for the local alignment problem.

Definition 5. For any strings X and Y over Σ , let the occurrence DAG, denoted $G_{\text{occ}}(X, Y)$, of Y in X be the edge-weighted DAG obtained from $G(X, Y)$ by adding two vertices src and snk , bypass edges $\text{in}(i')$ of weight zero from src to $d(i', 0)$ for all indices i' with $0 \leq i' \leq |X|$, and bypass edges $\text{out}(i)$ of weight zero from $d(i, |Y|)$ to snk for all indices i with $0 \leq i \leq |X|$. For any vertex v in $G_{\text{occ}}(X, Y)$ other than src , let $i'(v)$ be the greatest index i' such that some heaviest path from src to v passes through bypass edge $\text{in}(i')$.

Lemma 2. *Substring $X(i', i]$ is an occurrence of Y in X if and only if some heaviest path in $G_{\text{occ}}(X, Y)$ from src to snk passes through $\text{out}(i)$, $i'(d(i, |Y|)) = i'$, and no substrings $X(i', i'']$ with $i' < i'' < i$ are occurrences of Y in X .*

Lemma 3. *For any vertex v in $G_{\text{occ}}(X, Y)$ other than src , $i'(v)$ is equal to the maximum of $i'(u)$ over all vertices u in $\text{to}(v)$ with $dp(v) = dp(u) + w(u, v)$, where we treat $i'(u) = i'$ if $u = \text{src}$ and $v = d(i', 0)$.*

Let $DP_{\text{occ}}(i)$ and $I'(i)$ denote the array of DP table values $dp(v)$ and the array of indices $i'(v)$ for all vertices v in the i th row of $G_{\text{occ}}(X, Y)$, respectively. It then follows from the recurrence relation of DP table value $dp(v)$ given in Definition 3 that $DP_{\text{occ}}(i)$ can be constructed in $O(|Y|)$ time from scratch, if $i = 0$, or from $DP_{\text{occ}}(i - 1)$, otherwise. Similarly, we can obtain $I'(i)$ in $O(|Y|)$ time from scratch, if $i = 0$, or from $DP_{\text{occ}}(i - 1)$, $I'(i - 1)$, and $DP(i)_{\text{occ}}$, otherwise, based on Lemma 3. Thus, we obtain Algorithm `findOcc`(X, Y) presented in Fig. 1 as an $O(|X||Y|)$ -time, $O(|Y|)$ -space algorithm that enumerates all occurrences of Y in X . In this algorithm, lines 1 through 4 prepare $dp(\text{snk})$, the weight of any heaviest path from src to snk , as the value of variable dp_{snk} . Using this value, each iteration of lines 7 through 9 applies Lemma 2, where index variable i' in line 8 is maintained so as to indicate that, if $i' \geq 0$, then some substring $X(i', i'']$ with $i' < i'' < i$ is an occurrence of Y in X .

Lemma 4. *For any strings X and Y over Σ , Algorithm `findOcc`(X, Y) enumerates all occurrences $X(i', i]$ of Y in X in ascending order with respect to i and, hence, with respect to i' in $O(|X||Y|)$ time and $O(|Y|)$ space.*

Now we present the StrCA DAG, together with the properties crucial to designing the proposed algorithm.

- 1: Let $dp_{snk} = 0$;
- 2: for each index i from 0 to $|X|$,
- 3: construct $DP_{occ}(i)$; delete $DP_{occ}(i-1)$ if $i \geq 1$;
- 4: let $dp_{snk} = \max(dp_{snk}, dp(d(i, |Y|)) + w(out(i)))$;
- 5: let $i' = -1$;
- 6: for each index i from 0 to $|X|$,
- 7: construct $DP_{occ}(i)$ and $I'(i)$; delete $DP_{occ}(i-1)$ and $I'(i-1)$ if $i \geq 1$;
- 8: if $dp(d(i, |Y|)) = dp_{snk}$ and $i' < i'(d(i, |Y|))$, then
- 9: let $i' = i'(d(i, |Y|))$; report that $X(i', i]$ is an occurrence of Y in X .

Fig. 1. Algorithm findOcc(X, Y)

- 1: Obtain all occurrences of P in B by executing Algorithm findOcc(B, P);
- 2: let $i' = 0$; let $i = 0$;
- 3: for each occurrence $A(i'_P, i_P]$ of P in A reported by Algorithm findOcc(A, P),
which is executed along with iterations of this sentence,
- 4: while $i' \leq i'_P$,
- 5: compute $DP_{pref}(i')$; delete $DP_{pref}(i' - 1)$ if $i' \geq 1$;
- 6: increase i' by one;
- 7: while $i \leq i_P$, or $i \leq m$ if $A(i'_P, i_P]$ is the last occurrence of P in A ,
- 8: compute $DP_{suff}(i)$ and $TR(i)$; delete $DP_{suff}(i-1)$ and $TR(i-1)$ if $i \geq 1$;
- 9: increase i by one;
- 10: output $(A(i', i], B(j', j])$, where the edge in $tr(d_{suff}(m, n))$ obtained as an element of $TR(m)$ is from $d_{pref}(i', j')$ to $d_{suff}(i, j)$.

Fig. 2. Algorithm solveStrCA(A, B, P)

Definition 6. Let G_{pref} and G_{suff} be copies of $G(A, B)$ and let vertices in them be indicated by subscripts *pref* and *suff*, respectively. Let the StrCA DAG, denoted G_{StrCA} , be the edge-weighted DAG obtained from G_{pref} and G_{suff} by adding a transition edge of weight zero from $d_{pref}(i', j')$ to $d_{suff}(i, j)$ for any pair of an occurrence $A(i', i]$ of P in A and an occurrence $B(j', j]$ of P in B and adding a dummy transition edge of weight $-\infty$ from $d_{pref}(0, 0)$ to $d_{suff}(0, 0)$. For any vertex v in G_{suff} , let $tr(v)$ represent an arbitrary transition edge through which some heaviest path from $d_{pref}(0, 0)$ to v passes.

Lemma 5. *Substring pair $(A(i', i], B(j', j])$ is a solution of the StrCA problem if and only if the transition edge from $d_{pref}(i', j')$ to $d_{suff}(i, j)$ is passed through by some heaviest path in G_{StrCA} from $d_{pref}(0, 0)$ to $d_{suff}(m, n)$. Hence, $tr(d_{suff}(m, n))$ gives a solution of the StrCA problem.*

Lemma 6. *For any vertex v in G_{suff} and any vertex u in $to(v)$ with $dp(v) = dp(u) + w(u, v)$, $tr(u)$ is an instance of $tr(v)$, where we treat the transition edge from u to v as $tr(u)$ if u is a vertex in G_{pref} .*

The proposed algorithm solves the StrCA problem based on Lemma 5. The key idea to achieve linear-space computation of $tr(d_{suff}(m, n))$ is to successively focus on which transition edge some heaviest path in G_{StrCA} from $d_{pref}(0, 0)$

to each vertex v in G_{suff} passes through. According to the recurrence relation of $tr(v)$ given in Lemma 6, the algorithm determines $tr(v)$ for each vertex v in G_{suff} and forget previously determined $tr(u)$ no longer in use successively. This is unlike in the case of the algorithm adopting an approach similar to the quadratic-space algorithm of Deorowicz [5] for the STR-IC-LCS problem, which simultaneously determines how much any heaviest path from $d_{\text{pref}}(0, 0)$ to $d_{\text{suff}}(m, n)$ passing through each of all transition edges weighs.

Let $DP_{\text{pref}}(i')$ denote the array of DP table values $dp(v)$ for all vertices v in the i' th row of G_{pref} and let $DP_{\text{suff}}(i)$ and $TR(i)$ denote the array of DP table values $dp(v)$ and the array of transition edges $tr(v)$ for all vertices v in the i th row of G_{suff} , respectively. Then, $DP_{\text{pref}}(i')$ can be constructed in $O(n)$ time from scratch, if $i' = 0$, or from $DP_{\text{pref}}(i' - 1)$, otherwise. Furthermore, $DP_{\text{suff}}(i)$ and $TR(i)$ can be constructed in $O(n)$ time from scratch, if $i = 0$, or otherwise from $DP_{\text{suff}}(i - 1)$ and $TR(i - 1)$, together with $DP_{\text{pref}}(i')$ if A has an occurrence $A(i', i]$ of P for some index i' . Thus, we eventually obtain Algorithm `solveStrCA`(A, B, P) presented in Fig. 2 as the proposed algorithm for the StrCA problem, which satisfies the following theorem.

Theorem 1. *The StrCA problem is solvable in $O(mn)$ time and $O(n)$ space by executing Algorithm `solveStrCA`(A, B, P).*

References

1. Arslan, A.N.: Regular expression constrained sequence alignment. *J. Discrete Algorithms* **5**, 647–661 (2007)
2. Chen, Y.-C., Chao, K.-M.: On the generalized constrained longest common subsequence problems. *J. Comb. Optim.* **21**, 383–392 (2011)
3. Chung, Y.-S., Lu, C.L., Tang, C.Y.: Efficient algorithms for regular expression constrained sequence alignment. *Inf. Process. Lett.* **103**, 240–246 (2007)
4. Chin, F.Y.L., De Santis, A., Ferrara, A.L., Ho, N.L., Kim, S.K.: A simple algorithm for the constrained sequence problems. *Inf. Process. Lett.* **90**, 175–179 (2004)
5. Deorowicz, S.: Quadratic-time algorithm for a string constrained LCS problem. *Inf. Process. Lett.* **112**, 423–426 (2012)
6. Gotoh, O.: An improved algorithm for matching biological sequences. *J. Mol. Biol.* **162**, 705–708 (1982)
7. Gusfield, D.: *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, Cambridge (1997)
8. Hirschberg, D.S.: Algorithms for the longest common subsequence problem. *J. ACM* **24**, 664–675 (1977)
9. Kucherov, G., Pinhas, T., Ziv-Ukelson, M.: Regular language constrained sequence alignment revisited. *J. Comput. Biol.* **18**, 771–781 (2011)
10. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.* **48**, 443–453 (1970)
11. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. *J. Mol. Biol.* **147**, 195–197 (1981)
12. Tsai, Y.-T.: The constrained longest common subsequence problem. *Inf. Process. Lett.* **88**, 173–176 (2003)

String Processing and Information Retrieval
23rd International Symposium, SPIRE 2016, Beppu,
Japan, October 18-20, 2016, Proceedings
Inenaga, S.; Sadakane, K.; Sakai, T. (Eds.)
2016, XVI, 273 p. 58 illus., Softcover
ISBN: 978-3-319-46048-2