Available online at www.sciencedirect.com
ScienceDirect
Information Processing Letters 103 (2007) 240–246
Information Processing Letters
www.elsevier.com/locate/ipl

Efficient algorithms for regular expression constrained sequence alignment ☆

Yun-Sheng Chung [a], Chin Lung Lu [b], Chuan Yi Tang [a,*]

[a] *Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan 300, ROC*
[b] *Department of Biological Science and Technology, National Chiao Tung University, Hsinchu, Taiwan 300, ROC*

Received 16 October 2006; received in revised form 18 March 2007; accepted 23 April 2007

Available online 29 April 2007

Communicated by F.Y.L. Chin

Abstract

Imposing constraints is an effective means to incorporate biological knowledge into alignment procedures. As in the PROSITE database, functional sites of proteins can be effectively described as regular expressions. In an alignment of protein sequences it is natural to expect that functional motifs should be aligned together. Due to this motivation, Arslan introduced the regular expression constrained sequence alignment problem and proposed an algorithm which, if implemented naïvely, can take time and space up to $O(|\Sigma|^2|V|^4n^2)$ and $O(|\Sigma|^2|V|^4n)$, respectively, where Σ is the alphabet, n is the sequence length, and V is the set of states in an automaton equivalent to the input regular expression. In this paper we propose a more efficient algorithm solving this problem which takes $O(|V|^3n^2)$ time and $O(|V|^2n)$ space in the worst case. If $|V| = O(\log n)$ we propose another algorithm with time complexity $O(|V|^2 \log |V|n^2)$. The time complexity of our algorithms is independent of Σ, which is desirable in protein applications where the formulation of this problem originates; a factor of $|\Sigma|^2 = 400$ in the time complexity of the previously proposed algorithm would significantly affect the efficiency in practice.

Keywords: Constrained sequence alignment; Algorithms; Regular expression; Dynamic programming

1. Introduction

Sequence alignment is one of the most fundamental problems in computational biology. Due to its importance, it has been extensively studied in the past (see, e.g., [8]). As biological knowledge and predictions grow, it is often desirable if one can incorporate more information into the alignment procedure in hope that the alignment result can be more biologically meaningful and reasonable. In particular, when the input sequences share some properties, one then expects that the resulting alignment should not violate these properties. Such kind of preservation is in its nature a satisfaction of properly defined constraints. Due to this need, Tang et al. [11] defined the constrained sequence alignment problem (CSA, for short). They considered the alignment of RNase sequences which share a conserved sequence of residues H, K, H. It is expected that the

☆ A preliminary version of this work was presented in the 17th Annual Symposium on Combinatorial Pattern Matching, CPM 2006.

* Corresponding author.
E-mail addresses: yschung@algorithm.cs.nthu.edu.tw (Y.-S. Chung), cllu@mail.nctu.edu.tw (C.L. Lu), cytang@cs.nthu.edu.tw (C.Y. Tang).

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2007.04.007

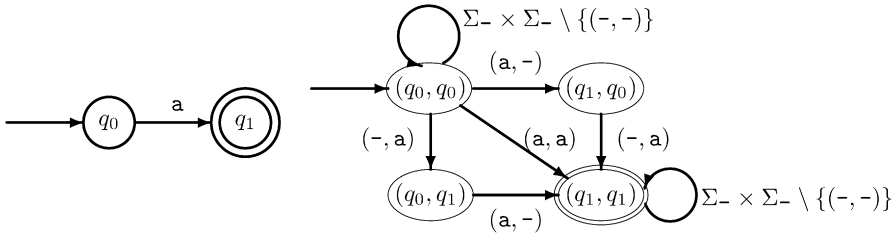

Fig. 1. Left: automaton A, equivalent to $R = \mathtt{a}$. Right: weighted automaton M, constructed by $A \times A$.

alignment result should have these conserved residues aligned together. A constraint is then defined in [11] as a sequence of characters. The problem requires that the alignment result must contain a sequence of columns, with length the same as the constraint, such that each character in the constraint appears in exactly one of these columns and that each column consists solely of a character. The goal is to find the best such alignment.

Later, Chin et al. [3] proposed an efficient algorithm for the pairwise version of CSA, along with a 2-approximation algorithm for the multiple alignment version with scoring function satisfying triangle inequality. Tsai et al. [13] generalized the definition of a constraint from a sequence of characters to a sequence of strings (patterns), and the occurrences of each pattern in the sequences need not be identical to the pattern specified in the input. Instead, the Hamming distances between each pattern and its occurrences need only to be within a user-specified threshold. This formulation enables the user to align sequences so that some known motifs are required to be aligned together. Lu and Huang [9] then reduced the memory requirement of the algorithm in [13], which significantly improves the applicability of the tool. The constrained longest common subsequence problem, a special case of CSA, was studied in [12,4].

As indicated by Arslan [2], it is well known that proteins with similar functions often share some motifs. The PROSITE database [7] collects biologically significant sites and motifs of proteins. In particular, these motifs can be conveniently represented as regular expressions. Hence an alignment tool able to incorporate patterns in regular expressions would be useful. To fulfill this need, Arslan introduced the regular expression constrained sequence alignment problem (RECSA, for short) [2]. A feasible solution of RECSA is an alignment containing a run of contiguous columns such that both of the two substrings corresponding to these columns match the regular expression. The following example, constructed by Arslan [2], clearly illustrates the definition and its difference with a standard alignment without constraint:

```
T G F P S V G K T K D D - - - - A
|   |   | |         | | |           |
T - F - S V A - - K D D D G K S A

T - - - G F P S V G K T K D D A
|                           |
T F S V A K D D D G K S - - - A
        * * * * * * * *
```

In all examples given in this section, a match of identical symbols is scored 1 and all other cases are scored 0. The upper alignment shown above is an optimal alignment without constraint, while the lower one is an optimal constrained alignment. The constraint R is $(\mathtt{G}+\mathtt{A})\Sigma\Sigma\Sigma\Sigma\mathtt{GK}(\mathtt{S}+\mathtt{T})$, the P-loop motif. The starred columns "support" the satisfaction of constraint R: both GFPSVGKT and AKDDDGKS match R. For more descriptions about biological applications of RECSA, the reader is referred to Arslan's paper [2].

There are well-established algorithms to convert R into an ε-free NFA A (see, e.g., [6]). In [2], R is converted into A first. Then a weighted finite automaton M is constructed from A to accept all alignments satisfying R. Automaton M is essentially a product machine of A. Each state in M corresponds to a pair of states in A. The alphabet of M corresponds to edit operations; an edit operation is represented as a pair of symbols in $\Sigma_- = \Sigma \cup \{-\}$, excluding $(-,-)$. The transitions in M and their relationship with those in A are illustrated in Fig. 1. Given a sequence of edit operations, or equivalently an alignment, as input to M, some states in M can be reached from the initial state (q_0, q_0). Consider the example in Fig. 1: given $\left[\begin{smallmatrix}\mathtt{t}-\\ \mathtt{ta}\end{smallmatrix}\right]$, states (q_0, q_1) as well as (q_0, q_0) are reached, while given $\left[\begin{smallmatrix}\mathtt{t}-\mathtt{c}\\ \mathtt{tat}\end{smallmatrix}\right]$, no state other than (q_0, q_0) is reached. In particular, given a feasible constrained alignment satisfying R, for example, $\left[\begin{smallmatrix}\mathtt{ta}-\\ \mathtt{tac}\end{smallmatrix}\right]$, state (q_1, q_1), the final state of M in the example, is reached; the self loop on the final state is added for this purpose. As in a standard dynamic programming alignment algorithm, Arslan's algorithm iterates over pairs (i_1, i_2) of indices of the input strings.

On each (i_1, i_2), a corresponding M_{i_1,i_2}, an automaton with the same transitions as those in M, is constructed. Each state (p, q) of M_{i_1,i_2} contains the score of an optimal alignment of $S_1[1..i_1]$ and $S_2[1..i_2]$ such that (p, q) can be reached if the alignment is given to M_{i_1,i_2} as input; such (p, q) is called *active* in [2]. If no such alignment exists for a state (p, q) of M_{i_1,i_2}, its corresponding score is set to $-\infty$ and it is not active. Hence the optimal score of aligning S_1 and S_2 can be found by taking the maximum of the scores stored in all the final states in $M_{|S_1|,|S_2|}$. Given an additional input symbol $(a, b) \in \Sigma_-^2 \setminus \{(-,-)\}$, the active states in M_{i_1,i_2} may change, and the scores are updated by adding the score of the current input symbol (edit operation). The resulting weighted automaton is denoted by $M_{i_1,i_2}^{(a,b)}$. The score updating rule in Arslan's algorithm is

$$M_{i_1,i_2} = \max\bigl\{M_{i_1-1,i_2}^{(S_1[i_1],-)}, M_{i_1,i_2-1}^{(-,S_2[i_2])}, M_{i_1-1,i_2-1}^{(S_1[i_1],S_2[i_2])}\bigr\},$$

where the maximum operation applied on the three weighted automata yields a weighted automaton with each state scored by the maximum score from the corresponding states in the three automata.

To update the active states and the scores, all transitions in M are examined by the algorithm in [2]. The time complexity stated in [2] is $O(rn^2)$, where r is the size of the transition function of M. Let V be the set of states in A. There can be up to $|\Sigma|$ transitions from a state of A to another. Hence the number of transitions in A can be up to $O(|\Sigma||V|^2)$, and that in M can be up to $O(|\Sigma|^2|V|^4)$ since M is made by a product of A. The time complexity of the algorithm in [2] can then be $O(|\Sigma|^2|V|^4n^2)$, if implemented naïvely. In addition, $O(n)$ copies of M are maintained throughout in [2], and the space complexity stated in [2] is $O(rn)$, which can be up to $O(|\Sigma|^2|V|^4n)$, if implemented naïvely. In [2] it is not mentioned how to reconstruct the optimal alignment. The stated space complexity is for the computation of the optimal score. Here we make the dependence on the alphabet size explicit since in protein applications for which RECSA is originally formulated, $|\Sigma|^2$ is 400 which would significantly affect the efficiency.

The critical information in M is the scores on the states; storing the entire M is unnecessary. In this paper, we directly use a matrix to store scores. The matrix is implemented using a simple array and can be easily manipulated. Time and space complexity of our algorithm are, respectively, bounded by $O(|V|^3n^2)$ and $O(|V|^2n)$ in the worst case; we also mention how the optimal alignment can be reconstructed within this space bound. When $|V| = O(\log n)$, which is not remote since in general R is much shorter than the input sequences [10], we propose another algorithm to take advantage of operations on $O(\log n)$-length data, which takes constant time in the standard unit-cost RAM model. The resulting time complexity is $O(|V|^2 \log |V| n^2)$ in this case. Although M is not used in our algorithm, it serves as a conceptual framework facilitating our presentation.

2. Preliminaries

For convenience throughout this paper we assume that $|S_1| = |S_2| = n$. An edit operation can be defined to be a symbol in $\Sigma_- \times \Sigma_- \setminus \{(-,-)\}$, where $\Sigma_- = \Sigma \cup \{-\}$. An edit operation is written as either a pair (a, b) or a column vector $\left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right]$ in this paper. Define $\rho : \Sigma_-^* \to \Sigma^*$ to be the "removing spaces" operator, e.g., $\rho(\texttt{a-tc-g}) = \texttt{atcg}$. A sequence $\mathcal{A} = \left[\begin{smallmatrix} a_1 & \cdots & a_m \\ b_1 & & b_m \end{smallmatrix}\right]$ of edit operations is called an alignment of S_1 and S_2 if $\rho(a_1 \cdots a_m) = S_1$ and $\rho(b_1 \cdots b_m) = S_2$. Let γ be a real-valued scoring function defined on sequences of edit operations satisfying

$$\gamma\left(\left[\begin{smallmatrix} a_1 & \cdots & a_m \\ b_1 & & b_m \end{smallmatrix}\right]\right) = \begin{cases} 0 & \text{if } m = 0, \text{ i.e., if it is} \\ & \quad \text{an empty alignment;} \\ \gamma\left(\left[\begin{smallmatrix} a_1 & \cdots & a_{m-1} \\ b_1 & & b_{m-1} \end{smallmatrix}\right]\right) + \gamma\left(\left[\begin{smallmatrix} a_m \\ b_m \end{smallmatrix}\right]\right) & \\ \quad \text{otherwise.} & \end{cases}$$

Let R be a regular expression over alphabet Σ. The goal of RECSA is to find a maximum-scored alignment $\mathcal{A} = \left[\begin{smallmatrix} a_1 & \cdots & a_m \\ b_1 & & b_m \end{smallmatrix}\right]$ of S_1 and S_2 such that there exist ℓ_1 and ℓ_2 with both $\rho(a_{\ell_1} \cdots a_{\ell_2})$ and $\rho(b_{\ell_1} \cdots b_{\ell_2})$ matching R.

Let $A = (Q, \Sigma, \delta, q_0, F)$ be an ε-free NFA equivalent to R, which can be constructed manually or by any established algorithm. Then $\delta(p, \varepsilon) = \delta(p, -) = \{p\}$ for all $p \in Q$. We also define $\delta(Q', a)$ to be $\bigcup_{p \in Q'} \delta(p, a)$, where $Q' \subseteq Q$ and $a \in \Sigma$. In this paper we use Q and V interchangeably. Following the notations in [2], $M = (Q^M, W^M, \Sigma^M, \delta^M, q_0^M, F^M)$, where $Q^M = Q \times Q$, $W^M : Q^M \to \mathbb{R}$, $\Sigma^M = \Sigma_- \times \Sigma_- \setminus \{(-,-)\}$, $q_0^M = (q_0, q_0)$ and $F^M = F \times F$. Transition function δ^M is defined as

$$\delta^M\bigl((p, q), (a, b)\bigr) = \begin{cases} \delta(p, a) \times \delta(q, b) & \\ \quad \text{if } (p, q) \notin F^M \cup \{(q_0, q_0)\}; & \\ \delta(p, a) \times \delta(q, b) \cup \{(p, q)\} & \\ \quad \text{otherwise.} & \end{cases}$$

Function δ^M can be naturally extended to be defined on the Cartesian product of Q^M and the set of all possible alignments. A state (p, q) of M_{i_1,i_2} is active iff there exists some alignment $\mathcal{A}$ of $S_1[1..i_1]$ and $S_2[1..i_2]$ such that $(p, q) \in \delta^M(q_0^M, \mathcal{A})$. Each state (p, q) in M_{i_1,i_2} has a score $W_{i_1,i_2}(p, q)$ assigned to it. If (p, q) is active, then $W_{i_1,i_2}(p, q)$ is the score of an optimal alignment $\mathcal{A}$ of $S_1[1..i_1]$ and $S_2[1..i_2]$ such that $(p, q) \in$

$\delta^M(q_0^M, \mathcal{A})$. Otherwise no such alignment exists and $W_{i_1,i_2}(p,q) = -\infty$.

3. The algorithms

In this section we present two algorithms, the first for the general case and the second for the case that $|V| = O(\log n)$.

3.1. The general case

Recall that automaton M_{i_1,i_2} accepts alignments of $S_1[1..i_1]$ and $S_2[1..i_2]$ satisfying the regular expression constraint. Also recall that once we have $W_{n,n}(p,q)$ for all $p,q \in Q$, the optimal score of aligning S_1 and S_2 with the constraint satisfied can be easily found by $\max_{(p,q)\in F\times F} W_{n,n}(p,q)$. Hence if we can compute all W_{i_1,i_2} then the problem is solved. There is a simple relationship among W_{i_1,i_2}, W_{i_1-1,i_2}, W_{i_1,i_2-1} and W_{i_1-1,i_2-1} which allows one to compute W_{i_1,i_2} based on the other three. Define $W_{i_1,i_2}^{(a,b)}(p,q) = \max_{(p',q'):\ (p,q)\in\delta^M((p',q'),(a,b))} W_{i_1,i_2}(p',q') + \gamma(a,b)$. Then for $i_1, i_2 \geqslant 1$, the relationship can be stated as follows:

$$W_{i_1,i_2}(p,q) = \max\bigl\{W_{i_1-1,i_2}^{(S_1[i_1],-)}(p,q),\ W_{i_1,i_2-1}^{(-,S_2[i_2])}(p,q),\ W_{i_1-1,i_2-1}^{(S_1[i_1],S_2[i_2])}(p,q)\bigr\}. \tag{1}$$

The algorithm in [2] is based on (1). On each (i_1,i_2), weighted automaton M_{i_1,i_2}, whose transitions are the same as those in M, is constructed such that state (p,q) in M_{i_1,i_2} carries the score $W_{i_1,i_2}(p,q)$. Given M_{i_1-1,i_2}, $M_{i_1-1,i_2}^{(S_1[i_1],-)}$ can be computed, which is also a weighted automaton where state (p,q) carries the score $W_{i_1-1,i_2}^{(S_1[i_1],-)}(p,q)$; $M_{i_1,i_2-1}^{(-,S_2[i_2])}$ and $M_{i_1-1,i_2-1}^{(S_1[i_1],S_2[i_2])}$ are similarly defined. Then M_{i_1,i_2} is computed as $\max\{M_{i_1-1,i_2}^{(S_1[i_1],-)}, M_{i_1,i_2-1}^{(-,S_2[i_2])}, M_{i_1-1,i_2-1}^{(S_1[i_1],S_2[i_2])}\}$, which is a weighted automaton where state (p,q) has score as defined in the right-hand side of (1). To compute $M_{i_1-1,i_2}^{(S_1[i_1],-)}$ from M_{i_1-1,i_2}, in [2] each transition of M_{i_1-1,i_2} is examined a constant number of times so that for each (p,q) the score $W_{i_1-1,i_2}^{(S_1[i_1],-)}(p,q)$ can be computed. Automata $M_{i_1,i_2-1}^{(-,S_2[i_2])}$ and $M_{i_1-1,i_2-1}^{(S_1[i_1],S_2[i_2])}$ are computed in the same way, and M_{i_1,i_2} can then be computed easily.

Let E be the set of "label-free" arcs in A, i.e., $(q',q) \in E$ iff $q \in \delta(q',a)$ for some $a \in \Sigma$. In particular, there can be at most one "label-free" arc from state q' to q. Clearly, there can be up to $O(|\Sigma||E|)$ transitions in A and $O(|\Sigma|^2|E|^2)$ transitions in M.

It is not necessary to maintain the whole machine structure of M on each (i_1,i_2). The relevant information is the scores $W_{i_1,i_2}(p,q)$. On each (i_1,i_2) we use table L_{i_1,i_2} to store W_{i_1,i_2}. In addition, to handle the transitions we need not use δ^M directly; the use of δ is sufficient. We construct a table T to represent δ: $T[q',a;q] = 1$ if $q \in \delta(q',a)$, and $T[q',a;q] = 0$ otherwise. This construction is easy, taking $O(|\Sigma||V|^2)$ time and space if automaton A is originally represented as an adjacency list. The computation of $L_{i_1,i_2}[p,q] = W_{i_1,i_2}(p,q)$ can then be improved, and is detailed as follows.

For (i_1,i_2) fixed, first consider the computation of $W_{i_1-1,i_2-1}^{(S_1[i_1],S_2[i_2])}(p,q)$. In terms of δ instead of δ^M, $W_{i_1-1,i_2-1}^{(S_1[i_1],S_2[i_2])}(p,q)$ can be written as given in formula (2) below. Hence the use of δ suffices.

Assume that $L_{i_1-1,i_2-1}[p,q]$ has been correctly computed for all $p,q \in V$. Then to compute $L_{i_1-1,i_2-1}^{(S_1[i_1],S_2[i_2])}[p,q]$ for all $p,q \in V$, expected to be equal to $W_{i_1-1,i_2-1}^{(S_1[i_1],S_2[i_2])}(p,q)$ and initialized to be $-\infty$, we proceed as follows. Let L be a temporary table with all entries $L[p,q]$ initialized to be $-\infty$.

```
1: for p', p ∈ V with T[p', S1[i1]; p] = 1 do
2:     for q' ∈ V do
3:         L[p, q']
               ← max{L[p, q'], L_{i1-1,i2-1}[p', q'] + γ(S1[i1],
                     S2[i2])};
4: for q', q ∈ V with T[q', S2[i2]; q] = 1 do
5:     for p ∈ V do
6:         L_{i1-1,i2-1}^{(S1[i1],S2[i2])}[p, q]
               ← max{L_{i1-1,i2-1}^{(S1[i1],S2[i2])}[p, q], L[p, q']};
7: for p, q ∈ V with (p, q) ∈ (F × F) ∪ {(q0, q0)} do
8:     L_{i1-1,i2-1}^{(S1[i1],S2[i2])}[p, q]
           ← max{L_{i1-1,i2-1}^{(S1[i1],S2[i2])}[p, q],
                 L_{i1-1,i2-1}[p, q] + γ(S1[i1], S2[i2])};
```

In the first part (lines 1–3) $L[p,q']$ is computed to be $\max_{p':\ p\in\delta(p',S_1[i_1])} L_{i_1-1,i_2-1}[p',q'] + \gamma(S_1[i_1],S_2[i_2])$. In the second part (lines 4–6) $L_{i_1-1,i_2-1}^{(S_1[i_1],S_2[i_2])}[p,q]$ is

$$\begin{cases} \max_{p',q':\ p\in\delta(p',S_1[i_1]),q\in\delta(q',S_2[i_2])} W_{i_1-1,i_2-1}(p',q') + \gamma(S_1[i_1],S_2[i_2]), \\ \quad \text{if } (p,q) \notin F^M \cup \{(q_0,q_0)\}; \\ \max_{p',q':\ p\in\delta(p',S_1[i_1]),q\in\delta(q',S_2[i_2]) \text{ or } (p',q')=(p,q)} W_{i_1-1,i_2-1}(p',q') + \gamma(S_1[i_1],S_2[i_2]), \\ \quad \text{otherwise.} \end{cases} \tag{2}$$

computed to be $\max_{q':\, q\in\delta(q',S_2[i_2])} L[p,q']$, which in turn is $\max_{p':\, p\in\delta(p',S_1[i_1]),q':\, q\in\delta(q',S_2[i_2])} L_{i_1-1,i_2-1}[p', q'] + \gamma(S_1[i_1], S_2[i_2])$, as desired. The last two lines deal with the self loops on the final and initial states in M.

For $p \in V$, let $\deg(p)$ be the in degree of p in the graph (V, E), where E is the set of label-free arcs obtained from A. Then the time taken by the above code segment is $O(\sum_{p\in V} \deg(p) \times |V|)$, which is $O(|E||V|)$. Basically, the two-step approach makes the algorithm presented here faster than the one in [2].

The computation for each of $L^{(S_1[i_1],-)}_{i_1-1,i_2}$ and $L^{(-,S_2[i_2])}_{i_1,i_2-1}$ is easier (e.g., to compute $L^{(S_1[i_1],-)}_{i_1-1,i_2}$, we can remove lines 4–6 in the above code segment, replace $\gamma(S_1[i_1], S_2[i_2])$ with $\gamma(S_1[i_1], -)$, etc.), which also takes $O(|E||V|)$ time.

Now we consider the boundary cases when $i_1 = 0$ or $i_2 = 0$. It is clear that $W_{0,0}(q_0, q_0) = 0$ and that $W_{0,0}(p, q) = -\infty$ for $(p, q) \neq q_0^M$ since the only alignment of $S_1[1..0] = \varepsilon$ and $S_2[1..0] = \varepsilon$ is an empty alignment, with a score of 0 by the definition of γ, and from q_0^M we can only reach q_0^M given an empty alignment as input to M. In addition, it can be seen that

$$W_{i_1,i_2}(p,q) = \begin{cases} W^{(S_1[i_1],-)}_{i_1-1,0}(p,q) \\ \quad \text{if } i_1 > 0 \text{ and } i_2 = 0, \\ W^{(-,S_2[i_2])}_{0,i_2-1}(p,q) \\ \quad \text{if } i_2 > 0 \text{ and } i_1 = 0. \end{cases}$$

In [2], the weights for all the states $(p, q) \neq (q_0, q_0)$ in $M_{i_1,0}$ for $i_1 > 0$ or in M_{0,i_2} for $i_2 > 0$ are all set to $-\infty$, which, without appropriate assumptions on γ (e.g., triangle inequality; no such assumption is made in [2]), would lead to a failure of considering possible optimal constrained alignments like $\left[\begin{smallmatrix} \mathtt{c} & - \\ - & \mathtt{t} \end{smallmatrix}\right]$ where $R = \mathtt{c} + \mathtt{t}$, $S_1 = \mathtt{c}$ and $S_2 = \mathtt{t}$. Certainly, such a problem may be fixed by initializing weights in the manner stated above.

Now that $L^{(S_1[i_1],S_2[i_2])}_{i_1-1,i_2-1}$, $L^{(S_1[i_1],-)}_{i_1-1,i_2}$ and $L^{(-,S_2[i_2])}_{i_1,i_2-1}$ can be computed, by (1) it is easy to compute L_{i_1,i_2} in $O(|V|^2)$ time for a fixed (i_1, i_2). Graph (V, E) is connected if considered undirected, and hence $|V| = O(|E|)$. Therefore the total time of our algorithm is $O(|V||E|n^2)$.

To reconstruct the optimal constrained alignment, first recall that in an optimal constrained alignment $\mathcal{A} = \left[\begin{smallmatrix} a_1 & \cdots & a_m \\ b_1 & & b_m \end{smallmatrix}\right]$ of S_1 and S_2, there exist ℓ_1 and ℓ_2 such that $\delta(q_0, \rho(a_{\ell_1} \cdots a_{\ell_2})) \cap F \neq \emptyset$ and $\delta(q_0, \rho(b_{\ell_1} \cdots b_{\ell_2})) \cap F \neq \emptyset$ (recall that $\rho(\cdot)$ is the "removing spaces" operation). We may regard $\mathcal{A}$ as composed of three parts: an optimal unconstrained alignment of $\rho(a_1 \cdots a_{\ell_1-1})$ and $\rho(b_1 \cdots b_{\ell_1-1})$, that of $\rho(a_{\ell_1} \cdots a_{\ell_2})$ and $\rho(b_{\ell_1} \cdots b_{\ell_2})$ and that of $\rho(a_{\ell_2+1} \cdots a_m)$ and $\rho(b_{\ell_2+1} \cdots b_m)$. Note that, an optimal unconstrained alignment of $\rho(a_{\ell_1} \cdots a_{\ell_2})$ and $\rho(b_{\ell_1} \cdots b_{\ell_2})$ automatically satisfies the constraint, since both strings match the regular expression. Let $j_1 = |\rho(a_1 \cdots a_{\ell_1-1})|$, $j'_1 = |\rho(a_1 \cdots a_{\ell_2})|$, $j_2 = |\rho(b_1 \cdots b_{\ell_1-1})|$ and $j'_2 = |\rho(b_1 \cdots b_{\ell_2})|$. Clearly if we know j_1, j'_1, j_2 and j'_2 then the constrained alignment $\mathcal{A}$ can be constructed in $O(n)$ space by using Hirschberg's celebrated divide-and-conquer algorithm [5] to align $S_1[1..j_1]$ and $S_2[1..j_2]$, align $S_1[j_1 + 1..j'_1]$ and $S_2[j_2 + 1..j'_2]$, and align $S_1[j'_1 + 1..n]$ and $S_2[j'_2 + 1..n]$. For this purpose we compute matrices $\eta_1^{i_1,i_2}$ and $\eta_2^{i_1,i_2}$ for each (i_1, i_2). Let $\mathcal{A} = \left[\begin{smallmatrix} a_1 & \cdots & a_m \\ b_1 & & b_m \end{smallmatrix}\right]$ be the alignment implied by $L_{i_1,i_2}[p, q]$. If $(p, q) \in F \times F$, then $\eta_1^{i_1,i_2}[p, q]$ stores (j_1, j_2) and $\eta_2^{i_1,i_2}[p, q]$ stores (j'_1, j'_2) such that $p \in \delta(q_0, S_1[j_1 + 1..j'_1])$ and $q \in \delta(q_0, S_2[j_2 + 1..j'_2])$, where $S_1[j_1 + 1..j'_1] = \rho(a_{\ell_1} \cdots a_{\ell_2})$ and $S_2[j_2 + 1..j'_2] = \rho(b_{\ell_1} \cdots b_{\ell_2})$ for some ℓ_1 and ℓ_2. If $(p, q) \notin F \times F$, then $\eta_1^{i_1,i_2}[p, q]$ stores (j_1, j_2) such that $p \in \delta(q_0, S_1[j_1 + 1..i_1])$ and $q \in \delta(q_0, S_2[j_1 + 1..i_2])$, where $S_1[j_1 + 1..i_1] = \rho(a_{\ell_1} \cdots a_m)$ and $S_2[j_2 + 1..i_2] = \rho(b_{\ell_1} \cdots b_m)$ for some ℓ_1; $\eta_2^{i_1,i_2}[p, q]$ simply stores (i_1, i_2). Note that when computing L_{i_1,i_2} we can determine the value of (a_m, b_m); for example, $(a_m, b_m) = (S_1[i_1], -)$ if $L_{i_1,i_2}[p, q] = L^{(S_1[i_1],-)}_{i_1-1,i_2}[p, q]$.

To see how to compute these values we assume that $(a_m, b_m) = (S_1[i_1], -)$; the other two cases are similar. Let $(p', q) \in \delta(q_0^M, \left[\begin{smallmatrix} a_1 & \cdots & a_{m-1} \\ b_1 & & b_{m-1} \end{smallmatrix}\right])$ be such that $L^{(S_1[i_1],-)}_{i_1-1,i_2}[p, q] = L_{i_1-1,i_2}[p', q] + \gamma(S_1[i_1], -)$ and that $(p, q) \in \delta^M((p', q), (S_1[i_1], -))$. Such (p', q) can be determined during the computation of $L^{(S_1[i_1],-)}_{i_1-1,i_2}[p, q]$. Consider first that $(p, q) \notin F \times F$. Then $\eta_2^{i_1,i_2}[p, q] = (i_1, i_2)$. If $(p', q) = (q_0, q_0)$, then clearly we can set $\eta_1^{i_1,i_2}[p, q] = (i_1 - 1, i_2)$. If $(p', q) \neq (q_0, q_0)$ then we can set $\eta_1^{i_1,i_2}[p, q] = \eta_1^{i_1-1,i_2}[p', q]$ since if $(j_1, j_2) = \eta_1^{i_1-1,i_2}[p', q]$ then $p' \in \delta(q_0, S_1[j_1 + 1..i_1 - 1])$ and $q \in \delta(q_0, S_2[j_2 + 1..i_2])$ where $S_1[j_1 + 1..i_1 - 1] = \rho(a_{\ell_1} \cdots a_{m-1})$ and $S_2[j_2 + 1..i_2] = \rho(b_{\ell_1} \cdots b_{m-1})$ for some ℓ_1. Since $(p, q) \notin F \times F$, and either $p \neq q_0$ or $p \neq p'$, we have $(p, q) \in \delta^M((p', q), (S_1[i_1], -))$ iff $p \in \delta(p', S_1[i_1])$. Hence $p \in \delta(q_0, S_1[j_1 + 1..i_1])$ and $q \in \delta(q_0, S_2[j_2 + 1..i_2])$ with $S_1[j_1 + 1..i_1] = \rho(a_{\ell_1} \cdots a_m)$ and $S_2[j_2 + 1..i_2] = \rho(b_{\ell_1} \cdots b_m)$. Now consider $(p, q) \in F \times F$. When $(p, q) = (p', q)$, then we simply set $\eta_1^{i_1,i_2}[p, q] = \eta_1^{i_1-1,i_2}[p', q]$ and $\eta_2^{i_1,i_2}[p, q] = \eta_2^{i_1-1,i_2}[p', q]$. When $(p, q) \neq (p', q)$, then $\eta_1^{i_1,i_2}[p, q]$ and $\eta_2^{i_1,i_2}[p, q]$ are set as in the case where $(p, q) \notin F \times F$.

1: **for** $q \in V$ **do**
2: Compute array X such that $L_{i_1-1,i_2}[X[j], q] \geqslant L_{i_1-1,i_2}[X[j+1], q]$; /* by sorting */
3: Initialize bit vector Y of length $|V|$ to be all 0;
4: **for** $j = 1$ to $|V|$ **do**
5: $T' \leftarrow T[X[j], S_1[i_1]] \otimes \bar{Y}$;
/* T' is a temporary bit vector, $\otimes$ is the bitwise AND operator and $\bar{Y}$ is the bitwise complement of Y. */
6: $Y \leftarrow Y \oplus T'$; /* $\oplus$ is the bitwise OR operator. */
7: $t \leftarrow \|T'\|_1$; /* The number of bits with value 1 in T'. */
8: **for** $c \leftarrow 1$ to t **do**
9: $L^{(S_1[i_1],-)}_{i_1-1,i_2}[\lambda[T'], q] \leftarrow L_{i_1-1,i_2}[X[j], q] + \gamma(S_1[i_1], -)$;
/* $\lambda[T']$ is the position of the leftmost bit with value 1 in T'. */
10: Set bit $\lambda[T']$ of T' to 0;
11: **for** $p, q \in V$ with $(p, q) \in (F \times F) \cup \{(q_0, q_0)\}$ **do**
12: $L^{(S_1[i_1],-)}_{i_1-1,i_2}[p, q] \leftarrow \max\{L^{(S_1[i_1],-)}_{i_1-1,i_2}[p, q], L_{i_1-1,i_2}[p, q] + \gamma(S_1[i_1], -)\}$;

Algorithm 1.

Therefore $\eta_1^{i_1,i_2}$ and $\eta_2^{i_1,i_2}$ can be set without difficulty. For the boundary case, all entries in $\eta_1^{0,0}$ and $\eta_2^{0,0}$ are simply set to $(0, 0)$. Let

$$(p^*, q^*) = \arg \max_{(p',q') \in F \times F} \{L_{n,n}[p', q']\}.$$

With the knowledge of $\eta_1^{n,n}[p^*, q^*]$ and $\eta_2^{n,n}[p^*, q^*]$, an optimal constrained alignment of S_1 and S_2 can then be constructed in linear space.

When computing L_{i_1,i_2} all those $L_{i'_1,i'_2}$ with $i'_1 \leqslant i_1 - 2$ are not necessary. Similar arguments are also applicable to $\eta_1^{i_1,i_2}$ and $\eta_2^{i_1,i_2}$. Each of these matrices has $O(|V|^2)$ entries. Given $\eta_1^{n,n}$ and $\eta_2^{n,n}$, the reconstruction of an optimal constrained alignment takes $O(n)$ space (and $O(n^2)$ time so the time complexity stated before is not affected). Therefore the space complexity of our algorithm is $O(|V|^2 n)$. On the other hand, the space complexity stated in [2] is $O(rn)$, where r is the number of transitions in M, without reconstruction of the optimal alignment. It is clear that $r = \Omega(|V|^2)$.

3.2. An algorithm for short regular expression constraints

In general R is much shorter than S_1 and S_2 [10]. The number of states in the automaton A equivalent to R may hence also be small relative to n. Here we present another algorithm which can be combined with the algorithm presented in the last subsection to yield a more efficient algorithm when $|V| = O(\log n)$. Under this assumption, we represent each $T[q, a]$ (regarded as a vector containing entries $T[q, a; p]$ for all $p \in V$) by a bit vector of length $|V|$; such a bit vector now takes $O(1)$ space to store. For brevity we only show the computation of $L^{(S_1[i_1],-)}_{i_1-1,i_2}$. The other cases can be handled similarly.

To compute $L^{(S_1[i_1],-)}_{i_1-1,i_2}[p, q]$, we need the maximum of $L_{i_1-1,i_2}[p', q] + \gamma(S_1[i_1], -)$ over all $p' \in V$ such that $p \in \delta(p', S_1[i_1])$. If computed directly, this maximum takes time proportional to the number of such p'. However, if we already know which p' is responsible for the maximum, then no more computation is needed. Consider the computation of $L^{(S_1[i_1],-)}_{i_1-1,i_2}[p, q]$ for all $p \in V$ for a fixed $q \in V$. Suppose that $L_{i_1-1,i_2}[X[j], q] \geqslant L_{i_1-1,i_2}[X[j+1], q]$, where $j = 1, \ldots, |V| - 1$ and $\{X[i]\colon\ 1 \leqslant i \leqslant |V|\} = V$. Clearly, for all those p with $p \in \delta(X[1], S_1[i_1])$, $L^{(S_1[i_1],-)}_{i_1-1,i_2}[p, q]$ can be simply set to $L_{i_1-1,i_2}[X[1], q] + \gamma(S_1[i_1], -)$. As to those p with $p \in \delta(X[2], S_1[i_1])$, if $p \notin \delta(X[1], S_1[i_1])$ also, then $L^{(S_1[i_1],-)}_{i_1-1,i_2}[p, q]$ can be simply set to $L_{i_1-1,i_2}[X[2], q] + \gamma(S_1[i_1], -)$. Continue with this process, all $L^{(S_1[i_1],-)}_{i_1-1,i_2}[p, q]$ can be computed. Constant time operations on data with $O(\log n)$ bits make the determination of whether $p \in \delta(X[j], S_1[i_1]) \setminus \bigcup_{k=1}^{j-1} \delta(X[k], S_1[i_1])$ very efficient. Indeed, if we have a bit vector Y such that $Y[p] = 1$ iff $p \in \bigcup_{k=1}^{j-1} \delta(X[k], S_1[i_1])$, then *all* those $p \in \delta(X[j], S_1[i_1]) \setminus \bigcup_{k=1}^{j-1} \delta(X[k], S_1[i_1])$ can be found in constant time by taking $T[X[j], S_1[i_1]] \otimes \bar{Y}$, where $\otimes$ is the bitwise AND operation and $\bar{Y}$ is the bitwise complement of Y. This is the fundamental reason why the algorithm here can achieve the desired efficiency; note that comparable efficiency cannot be achieved for this operation without the assumption of $|V| = O(\log n)$. The procedure is formally stated in Algorithm 1.

The correctness should have been clear from the above discussion. Here simply note that lines 7–10 implement the statement "For each $p \in T'$ set $L^{(S_1[i_1],-)}_{i_1-1,i_2}[p, q] \leftarrow L_{i_1-1,i_2}[X[j], q] + \gamma(S_1[i_1], -)$."

We now analyze the time complexity of the above code segment. Line 2 takes $O(|V|^2 \log |V|)$ total time. Line 3 takes $O(|V|)$ total time. Lines 5–7 take $O(1)$ time

for each iteration, hence take $O(|V|^2)$ time in total. For a fixed q, line 8 is executed $O(|V|)$ times since the T' for a specific value of j is disjoint from all those corresponding to other values of j. Hence the total time taken by lines 8–10 is $O(|V|^2)$. Lines 11–12 also take $O(|V|^2)$ total time. As we need to iterate over all index pairs (i_1, i_2), the overall time complexity for solving RECSA is $O(|V|^2 \log |V| n^2)$. Therefore by combining the algorithm presented in the last subsection, we can solve RECSA in time $O(\min\{|V| \log |V|, |E|\}|V| n^2)$.

The above results are summarized as follows.

Theorem 1. *RECSA can be solved in* $O(|V||E|n^2)$ *time and* $O(|V|^2 n)$ *space. If* $|V| = O(\log n)$*, then it can be solved in* $O(\min\{|E|, |V| \log |V|\}|V| n^2)$ *time and* $O(|V|^2 n)$ *space.*

4. Future works

In practice it is important to have an algorithm to align multiple sequences with the given regular expression constraint satisfied. In [2,1] it is suggested to extend the structure of weighted finite automata to support multiple sequences. However the size of such automata grows exponentially with the number of sequences, becoming an exponential multiplicative factor to the exponential time complexity of a multiple sequence alignment procedure. Hence a reasonable progressive algorithm of RECSA for multiple sequences is still in demand.

References

[1] A.N. Arslan, Multiple sequence alignment containing a sequence of regular expressions, in: Proc. IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology, 2005, pp. 1–7.

[2] A.N. Arslan, Regular expression constrained sequence alignment, in: A. Apostolico, M. Crochemore, K. Park (Eds.), CPM 2005, in: Lecture Notes in Computer Science, vol. 3537, Springer, 2005, pp. 322–333.

[3] F.Y.L. Chin, N.L. Ho, T.W. Lam, P.W.H. Wong, Efficient constrained multiple sequence alignment with performance guarantee, Journal of Bioinformatics and Computational Biology 3 (2005) 1–18.

[4] F.Y.L. Chin, A.D. Santis, A.L. Ferrara, N.L. Ho, S.K. Kim, A simple algorithm for the constrained sequence problems, Information Processing Letters 90 (2004) 175–179.

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

[6] J.E. Hopcroft, R. Motwani, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, second ed., Addison-Wesley, 2001.

[7] N. Hulo, C.J.A. Sigrist, V. Le Saux, P.S. Langendijk-Genevaux, L. Bordoli, A. Gattiker, E. De Castro, P. Bucher, A. Bairoch, Recent improvements to the PROSITE database, Nucleic Acids Research 32 (2004) 134–137.

[8] T. Jiang, Y. Xu, M.Q. Zhang (Eds.), Current Topics in Computational Molecular Biology, MIT Press, 2002.

[9] C.L. Lu, Y.P. Huang, A memory-efficient algorithm for multiple sequence alignment with constraints, Bioinformatics 21 (2005) 20–30.

[10] C.J. Sigrist, L. Cerutti, N. Hulo, A. Gattiker, L. Falquet, M. Pagni, A. Bairoch, P. Bucher, PROSITE: A documented database using patterns and profiles as motif descriptors, Briefings in Bioinformatics 3 (2002) 265–274.

[11] 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, W.-I. Chou, Constrained sequence alignment tool development and its application to RNase family alignment, Journal of Bioinformatics and Computational Biology 1 (2003) 267–287.

[12] Y.-T. Tsai, The constrained common sequence problem, Information Processing Letters 88 (2003) 173–176.

[13] Y.-T. Tsai, Y.P. Huang, C.T. Yu, C.L. Lu, MuSiC: A tool for multiple sequence alignment with constraints, Bioinformatics 20 (2004) 2309–2311.