Journal of Discrete Algorithms 24 (2014) 48–58

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

Algorithms for path-constrained sequence alignment [☆]

Tamar Pinhas [a,1], Nimrod Milo [a,1], Gregory Kucherov [a,b], Michal Ziv-Ukelson [a,*]

[a] *Department of Computer Science, Ben-Gurion University of the Negev, Be'er Sheva, Israel*
[b] *CNRS/LIGM, Université Marne-la-Vallée, France*

A R T I C L E   I N F O

Article history:
Available online 28 September 2013

Keywords:
Sequence alignment
Regular expressions
Dynamic programming
miRNA

A B S T R A C T

We define a novel variation on the constrained sequence alignment problem in which the constraint is given in the form of a regular expression. Given two sequences, an alphabet Γ describing pairwise sequence alignment operations, and a regular expression R over Γ, the problem is to compute the highest scoring sequence alignment A of the given sequences, such that $A \in \Gamma^* L(R) \Gamma^*$.

Two algorithms are given for solving this problem. The first basic algorithm is general and solves the problem in $O(nmr \log^2 r)$ time and $O(\min\{n, m\}r)$ space, where m and n are the lengths of the two sequences and r is the size of the NFA for R. The second algorithm is restricted to *rigid patterns* and exploits this restriction to reduce the NFA size factor r in the time complexity to a smaller factor corresponding to the length of the rigid pattern. A rigid pattern P is a regular expression of the form $P = P_1 \cup \cdots \cup P_k$, where P_i does not contain the Kleene-closure star or union. $|P|$ is compacted by supporting alignment patterns P that do not contain the Kleene-closure star, and exploits this constraint to reduce the NFA size factor r in the time complexity to a smaller factor $|P|$. $|P|$ is compacted by supporting alignment patterns extended by *meta-characters* including general insertion, deletion and match operations, as well as some cases of substitutions. meta-characters used in P. $\{m, i\}^*$ or $P \in (\Gamma \cup \{m, d\})^*$, the problem can be solved in time $O(nm)$, while for a pattern $P \in (\Gamma \cup \{m, i, d\})^*$, the problem can be solved in time $O(nm \log |P|)$. For a pattern $P \in (\Gamma \cup \{m, s, i, d\})^*$, the problem can be solved in time $O(nm \log |P|)$ in some cases: one case is for scoring functions *Score* for which there exists $\mathit{Score}' : \Sigma \to \mathbb{R}$ such that $\mathit{Score}(\nu, \sigma) = \mathit{Score}'(\nu) + \mathit{Score}'(\sigma)$ for every $\nu \neq \sigma$, and the other is when $occ_s(P) = O(\log(\max\{n, m\}))$. For a rigid pattern $P = P_1 \cup \cdots \cup P_k$, these time bounds range from $O(knm)$ to $O(knm \log(\max\{|P_i|\}))$, depending on the meta-characters used in P.

An additional result obtained along the way is an extension of the algorithm of Fischer and Paterson for String Matching with Wildcards. Our extension allows the input strings to include "negation symbols" (that match all letters but a specific one) while retaining the original time complexity.

We implemented both algorithms and applied them to data-mine new miRNA seeding patterns in *C. elegans* Clip-seq experimental data.

☆ An early version of this paper containing partial results appeared in LATA 2010 [17].
* Corresponding author.
E-mail address: michaluz@cs.bgu.ac.il (M. Ziv-Ukelson).
[1] These authors contributed equally to this work.

1570-8667/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.jda.2013.09.003

1. Introduction

Sequence alignment is one of the most basic and well-studied problems in string processing [8], with numerous applications in computational biology. However, various applications, such as modeling RNA–RNA interaction according to known hybridization patterns (exemplified in Section 5), motivate the application of additional constraints on the sequence alignment for the purpose of improved speed or accuracy. Additional constraints reflect a priori knowledge of the alignment and, therefore, narrow the problem search space or guide the search towards a preferred solution.

The types of constraints imposed on sequence alignment appearing in the literature can be classified into three categories as follows: In the first category, the constraint delineates aligned substrings of each of the aligned strings independently. Variants in this category include string pattern [20], regular expression [1,14] constraints. In the second category, the constraint delineates the alignment path, rather than the aligned substrings. Variants in this category include position anchoring [18], k-difference alignment [8] and spaced seeds [13]. In the third category, a context-sensitive scoring scheme is used to influence and guide the alignment. Examples of this category are position-based scoring [11] and the probabilistic models HMM [5] and CM [6].

In this paper, we address a new problem variant of constrained sequence alignment, denoted *Sequence Alignment with Regular Expression Path Constraint* (SA-REPC). In this sequence alignment variant, a regular expression pattern constrains the form of the alignment. Given two sequences S and T an alphabet Γ describing pairwise sequence alignment operations, and a regular expression R over Γ, the problem is to compute the highest scoring sequence alignment A of S and T, such that $A \in \Gamma^*L(R)\Gamma^*$. Among the above categories, the new problem we introduce in this paper falls within the second category, extending it to include constraints in the form of regular expressions.

We give two algorithms for solving the new problem. The first basic algorithm is general and solves the problem in $O(nmr\log^2 r)$ time and $O(\min\{n,m\}r)$ space, where m and n are the lengths of S and T, respectively, and r is the size of the NFA for R.

The second algorithm is restricted to *rigid patterns* and exploits this restriction to reduce the NFA size factor r in the time complexity to a smaller factor corresponding to the length of the rigid pattern. A rigid pattern P is a regular expression of the form $P = P_1 \cup \cdots \cup P_k$, where P_i does not contain the Kleene-closure star or union. In order to compact $|P|$ and utilize the factor difference advantage, the second algorithm supports alignment patterns extended by *meta-characters* $\{m,s,i,d\}$ and its time complexity depends on the meta-characters used in P. For a rigid pattern $P \in (\Gamma \cup \{m,i\})^*$ or $P \in (\Gamma \cup \{m,d\})^*$, the problem can be solved in time $O(nm)$, while for a rigid pattern $P \in (\Gamma \cup \{m,i,d\})^*$, the problem can be solved in time $O(nm\log|P|)$. For a rigid pattern $P \in (\Gamma \cup \{m,s,i,d\})^*$, the problem can be solved in time $O(nm\log|P|)$ in some cases: one case is for scoring functions *Score* for which there exists $Score' : \Sigma \to \mathbb{R}$ such that $Score(\nu,\sigma) = Score'(\nu) + Score'(\sigma)$ for every $\nu \neq \sigma$, and the other is when $occ_s(P) = O(\log(\max\{n,m\}))$. For a rigid pattern $P = P_1 \cup \cdots \cup P_k$, these time bounds become respectively $O(knm)$ and $O(knm\log(\max\{|P_i|\}))$.

An additional result obtained in this work body is an extension of the algorithm of Fischer and Paterson [7] for Exact String Matching with Wild Characters. Our extension allows the input strings to include "negation symbols" (that match all letters but a specific one) while retaining the original time complexity.

We implemented both algorithms and applied them to data-mine new miRNA seeding patterns in Clip-seq data. Our tool is available via web-interface in http://www.cs.bgu.ac.il/~negevcb/CAlign.

The rest of the paper proceeds as follows. Notations and problem definitions are given in Section 2. The first, general algorithm is described in Section 3. Then, in Section 4, we give the second, more efficient algorithm for constrained patterns. Finally, in Section 5, we give our implementation of the algorithms and exemplify their bioinformatic application. Additional details omitted from the paper can be found in the supplementary materials [19].

2. Notations and basic definitions

We assume a fixed *sequence* alphabet Σ that does not contain letter '$-$' which is used to denote indels (i.e. insert or delete operations). The *alignment alphabet* Γ is the alphabet of pairs of symbols: $\Gamma = \{\binom{\sigma}{\nu} \mid \sigma,\nu \in \Sigma \cup \{-\}\} \setminus \{\binom{-}{-}\}$. An alignment of two strings over Σ is naturally represented by a string over Γ. Given an alignment string $A = a_1 \ldots a_n$ over Γ we define $A^{\uparrow} = \sigma_1 \ldots \sigma_n$ and $A^{\downarrow} = \nu_1 \ldots \nu_n$, where $a_i = \binom{\sigma_i}{\nu_i}$ for all i. For any string S over $\Sigma \cup \{-\}$, we denote by S^- the string resulting from removing from S all the occurrences of letter '$-$'. In these terms, the alignment of two strings is defined as follows.

Definition 1. Given two strings S, T over Σ, an alignment of S and T is a string over Γ such that $(A^{\uparrow})^- = S$ and $(A^{\downarrow})^- = T$.

We use the notation $A = \text{align}(S,T)$ to state that A is an alignment of S and T.

Given a *score* function $Score : \Gamma \to \mathbb{R}$, we extend its definition to accept $A \in \Gamma^*$, as follows. The score of an alignment $A = a_1 \ldots a_n$ over Γ is defined by

$$Score(A) = \sum_{i=1}^{n} Score(a_i). \tag{1}$$