Journal of Discrete Algorithms 29 (2014) 27–46

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

Extending common intervals searching from permutations to sequences

Irena Rusu *

L.I.N.A., UMR 6241, Université de Nantes, 2 rue de la Houssiniére, BP 92208, 44322 Nantes, France

ARTICLE INFO

Article history:
Received 16 October 2013
Received in revised form 17 June 2014
Accepted 7 October 2014
Available online 18 October 2014

Keywords:
Sequence
Permutation
Genome
Algorithm
Common intervals

ABSTRACT

Common intervals have been defined as a modelization of gene clusters in genomes represented either as permutations or as sequences. Whereas optimal algorithms for finding common intervals in permutations exist even for an arbitrary number of permutations, in sequences no optimal algorithm has been proposed yet even for only two sequences. Surprisingly enough, when sequences are reduced to permutations, the existing algorithms perform far from the optimum, showing that their performances are not dependent, as they should be, on the structural complexity of the input sequences.

In this paper, we propose to characterize the structure of a sequence by the number q of different dominating orders composing it (called the *domination number*), and to use a recent algorithm for permutations in order to devise a new algorithm for two sequences. Its running time is in $O(q_1q_2p + q_1n_1 + q_2n_2 + N)$, where n_1, n_2 are the sizes of the two sequences, q_1, q_2 are their respective domination numbers, p is the alphabet size and N is the number of solutions to output. This algorithm performs better as q_1 and/or q_2 reduce, and when the two sequences are reduced to permutations (i.e. when $q_1 = q_2 = 1$) it has the same running time as the best algorithms for permutations. It is also the first algorithm for sequences whose running time involves the parameter size of the solution. As a counterpart, when q_1 and q_2 are of $O(n_1)$ and $O(n_2)$ respectively, the algorithm is less efficient than other approaches.

1. Introduction

One of the main assumptions in comparative genomics is that a set of genes occurring in neighboring locations within several genomes represent functionally related genes [8,13,17]. Such clusters of genes are then characterized by a highly conserved gene content, but a possibly different order of genes within different genomes. Common intervals have been defined to model clusters [20], and have been used since to detect clusters of functionally related genes [14,18], to compute similarity measures between genomes [4,2] and to predict protein functions [10,21].

Depending on the representation of genomes in such applications, allowing or not the presence of duplicated genes, comparative genomics requires for finding common intervals either in sequences or in permutations over a given alphabet. Whereas the most general – and thus useful in practice – case is the one involving sequences, the easiest to solve is the one involving permutations. This is why, in some approaches [1,2], sequences are reduced to permutations by renumbering the copies of the same gene according to evolutionary based hypothesis. Another way to exploit the performances of algorithms

* Tel.: +33 2 51 12 58 16; fax: +33 2 51 12 58 12.
E-mail address: Irena.Rusu@univ-nantes.fr.

http://dx.doi.org/10.1016/j.jda.2014.10.004
1570-8667/

Table 1
Running time for the existing algorithms when (1) the input sequences are as general as possible (lengths n_1 and n_2), when (2) one of them is a permutation (lengths $n_1 = p = |\Sigma|$ and $n_2 \geq p$), and when (3) both are permutations (lengths $n_1 = n_2 = p = |\Sigma|$). The running time of Didier's algorithm is updated according to Schmidt and Stoye's remark. On the last line, we add the memory space needed by each algorithm ($n_1 = p$ and $n_1 = n_2 = p$ respectively hold for the second and third case). Parameters l_1, l_2, q_1, q_2 are described in the text.

Sequence type	Didier [7]	Schmidt and Stoye [16]	Kolpakov and Raffinot [12]	Our algorithm
Seq. vs. Seq.	$O(n_1 n_2)$	$O(n_1 n_2)$	$O(n_1 + n_2 + l_1 \log p + l_2 \log p + N)$	$O(q_1 q_2 p + q_1 n_1 + q_2 n_2 + N)$
Perm. vs. Seq.	$O(p n_2)$	$O(p n_2)$	$O(n_2 + p^2 \log p + l_2 \log p + N)$	$O(q_2 n_2 + N)$
Perm. vs. Perm.	$O(p^2)$	$O(p^2)$	$O(p^2 \log p)$	$O(p + N)$
Memory space	$O(n_1 + n_2)$	$O(n_1 n_2)$	$O(n_1 + n_2 + l_1 \log p + l_2 \log p)$	$O(n_1 + n_2)$

for permutations in dealing with sequences is to see each sequence as a combination of several permutations, and to deal with these permutations rather than with the sequences. This is the approach we use here.

In permutations on p elements, finding common intervals may be done in $O(Kp + N)$ time where K is the number of permutations and N the number of solutions, using several algorithms proposed in the literature [20,5,9,15]. In sequences (see Table 1), even when only two sequences T and S of respective sizes n_1 and n_2 are considered, the best solutions take quadratic time. In a chronological order, the first algorithm is due to Didier [7] and performs in $O(n_1 n_2 \log n_2)$ time and $O(n_1 + n_2)$ space. Shortly later, Schmidt and Stoye [16] propose an $O(n_1 n_2)$ algorithm which needs $O(n_1 n_2)$ space, and note that Didier's algorithm may benefit from an existing result to achieve $O(n_1 n_2)$ running time whereas keeping the linear space. Both these algorithms use T to define, starting with a given element of it, growing intervals of T with fixed leftpoint and variable rightpoint, that are searched for into S. Alternative approaches attempt to avoid multiple searches of the same interval of T, due to multiple locations, by efficiently computing all intervals in T and all intervals in S before comparing them. The best running time reached by such an algorithm is in $O(n_1 + n_2 + l_1 \log p + l_2 \log p + N)$, obtained by merging the fingerprint trees proposed in [12], where l_1 (respectively l_2) is the number of maximal locations of the intervals in T (respectively S), p is the size of the alphabet and N is the number of solutions to output. The value l_1 (and similarly for l_2) is in $\Omega(p^2)$ and does not exceed $n_1 p$.

The running times of all the existing algorithms have at least two main drawbacks: first, they do not involve at all the number N of output solutions; second, they insufficiently exploit the particularities of the two sequences and, in the particular case where the sequences are reduced to permutations, need quadratic time instead of the optimal $O(p + N)$ time for two permutations on p elements. That means that their performances insufficiently depend both on the inherent complexity of the input sequences, and on the amount of results to output. Unlike the algorithms dealing with permutations, the algorithms for sequences lack of criteria allowing them to decide when the progressive generation of a candidate must be stopped, since it is useless. This is the reason why their running time is independent of the number of output solutions. This is also the reason why when sequences are reduced to permutations the running time is very unsatisfactory.

The most recent optimal algorithm for permutations [15] proposes a general framework for efficiently searching for common intervals and all of their known subclasses in K permutations, and has a twofold advantage, not proposed by other algorithms. First, it permits an easy and efficient selection of the common intervals to output based on two types of parameters. Second, assuming one permutation has been renumbered to be the identity permutation, it outputs all common intervals with the same minimum value together and in increasing order of their maximum value. We use here these properties to propose a new algorithm for finding common intervals in two sequences. Our algorithm strongly takes into account the structure of the input sequences, expressed by the number q of different dominating orders (which are permutations) composing the sequence ($q = 1$ for permutations). Consequently, it has a complexity depending both on this structure and on the number of output solutions. It runs in optimal $O(p + N)$ time for two permutations on p elements, is better than the other algorithms for sequences composed of few dominating orders and, as a counterpart, it performs less well as the number of composing dominating orders grows.

The structure of the paper is as follows. In Section 2 we define the main notions, including that of a dominating order, and give the results allowing us a first simplification of the problem. In Section 3 we propose our approach for finding common intervals in two sequences based on this simplification, for which we describe the general lines. In Sections 4, 5 and 6 we develop each of these general lines and prove correctness and complexity results. As the running time of the algorithm strongly relies on a data structure of type Union–Find, defined in A. Itai's manuscript [11], we describe this data structure in Section 7. Section 8 is the conclusion.

2. Preliminaries

Let T be a sequence of length n over an alphabet $\Sigma := \{1, 2, \ldots, p\}$. We denote the length of T by $\|T\|$, the set of elements in T by $Set(T)$, the element of T at position i, $1 \leq i \leq n$, by t_i and the subsequence of T delimited by positions i, j (included), with $1 \leq i \leq j \leq n$, by $T[i..j]$. An *interval* of T is any set I of integers from Σ such that there exist i, j with $1 \leq i \leq j \leq n$ and $I = Set(T[i..j])$. Then $[i, j]$ is called a *location* of I on T. A *maximal location* of I on T is any location $[i, j]$ such that neither $[i - 1, j]$ nor $[i, j + 1]$ is a location of I.

When T is the identity permutation $Id_p := (12 \ldots p)$, we denote $(i..j) := \{i, i + 1, \ldots, j\}$, which is also $Set(Id_p[i..j])$. Note that all intervals of Id_p are of this form, and that each interval has a unique location on Id_n. When T is an arbitrary