Output-Sensitive Algorithms for Finding the Nested Common Intervals of Two General Sequences

Biing-Feng Wang, Jhih-Hong Ye, and Song-Hsuan Lin

Department of Computer Science

National Tsing Hua University

Hsinchu, Taiwan 30013, Republic of China

bfwang@cs.nthu.edu.tw, shlin@cs.nthu.edu.tw, jhong@cs.nthu.edu.tw

Correspondence Address:

Prof. Biing-Feng Wang

Department of Computer Science

National Tsing Hua University

Hsinchu, Taiwan 30013, TAIWAN

Phone: 886-3-5742805

Fax: 886-3-5723694

E-mail: bfwang@cs.nthu.edu.tw

Output-Sensitive Algorithms for Finding the Nested Common Intervals of Two General Sequences [*]

Biing-Feng Wang, Jhih-Hong Ye, and Song-Hsuan Lin

Department of Computer Science, National Tsing Hua University

Hsinchu, Taiwan 30013, Republic of China

bfwang@cs.nthu.edu.tw, shlin@cs.nthu.edu.tw, jhong@cs.nthu.edu.tw

Abstract

The focus of this paper is the problem of finding all nested common intervals of two general sequences. Blin, Faye, and Stoye introduced three models to define nested common intervals of two sequences: the uniqueness, the free-inclusion, and the bijection models. We consider all the three models. For the uniqueness and the bijection models, we give $O(n + N_{\text{out}})$-time algorithms, where N_{out} denotes the size of the output. For the free-inclusion model, we give an $O(n^{1+\varepsilon} + N_{\text{out}})$-time algorithm, where $\varepsilon > 0$ is an arbitrarily small constant. We also present an upper bound on the size of the output for each model. For the uniqueness and the free-inclusion models, we show that $N_{\text{out}} = O(n^2)$. Consequently, our algorithms for the uniqueness and the free-inclusion models improve the previous $O(n^3)$-time algorithms by Blin, Faye, and Stoye. Let $C = \sum_{\alpha\in\Sigma} o_1(\alpha)o_2(\alpha)$, where Σ is the set of distinct genes, and $o_1(\alpha)$ and $o_2(\alpha)$ are, respectively, the numbers of copies of α in the two given sequences. For the bijection model, we show that $N_{\text{out}} = O(Cn)$. As compared with Blin, Faye, and Stoye's $O(n^3)$-time algorithm, our new algorithm is more practical, as C is likely to be much smaller than n^2 in practice.

Key words: algorithms, data structures, common intervals, comparative genomics, conserved gene clusters

[*] This research is supported by the National Science Council of the Republic of China under grant NSC-99-2221-E-007-046-MY3.

1. Introduction

Identifying conserved gene clusters of two or more genomes is a fundamental problem in computational comparative genomics [8, 11, 17, 18, 20, 24]. It is a delicate task to define gene cluster properties that are biologically meaningful. In recent years, several formal models have been developed to capture the essential biological features of a conserved gene cluster [1, 4, 6, 10, 12, 19, 22, 25, 26]. Please refer to [3] for an excellent survey. One important example is the *common interval model* introduced by Uno and Yagiura [25], in which a genome sequence is considered as a permutation of distinct genes and a *common interval* is defined to be a set of genes that appear consecutively, possibly in different orders, in two given permutations. Uno and Yagiura had an algorithm that finds all common intervals of two permutations of n genes in $O(n + N_{\text{out}})$ time, using $O(n)$ space, where N_{out} denotes the size of the output. Heber and Stoye [13] extended this work to find all common intervals of k permutations in $O(kn + N_{\text{out}})$ time, using $O(kn)$ space. In addition, Didier [9] extended this model to include paralogs by considering a sequence definition more general than a strict permutation, and gave an algorithm that finds all common intervals of two sequences in $O(n^2\log n)$ time, using $O(n)$ space, on the extended model. Later, Schmidt and Stoye [23] improved this result to $O(n^2)$ time. Schmidt and Stoye's algorithm can be extended to find all common intervals of k sequences in $O(kn^2)$ time, using $O(n^2)$ additional space.

For genome comparison, one cluster property that is generally not considered explicitly, but may be assumed implicitly, is *nestedness* [14]. A gene cluster of size s is *nested* if either $s = 2$ or it contains a nested cluster of size $s - 1$. Kurzik-Dumke and Zengerle [16] observed this property early in 1996. In this paper, we focus on the *nested common interval model*, in which a cluster is defined to be a common interval with the nestedness property. Hoberman and Duran [14] proposed this model and presented an $O(n^2)$-time simple algorithm for finding all nested common intervals of two permutations. Recently, Blin, Faye, and Stoye [5] studied the problem of finding nested common intervals comprehensively. For permutations, they gave an improved algorithm that requires $O(n + N_{\text{out}})$ time. A nested common interval of size s is *maximal* if it is not contained in a nested common interval of size $s + 1$. Blin, Faye, and Stoye showed that the number of maximal nested common intervals of two permutations is $O(n)$ and gave a linear time algorithm to find all maximal nested common intervals. For sequences, the definition of nestedness is subtle. Depending on the treatment one wants to apply to duplicate genes, Blin, Faye, and Stoye proposed three models to define nested common intervals of two sequences. The models are called, respectively, the *uniqueness*, the *free-inclusion*, and the *bijection models* in this paper. Formal definitions of these models are given in Section 2. On each of the models, Blin, Faye, and Stoye gave an $O(n^3)$-time algorithm for finding all nested common intervals of two sequences. An *approximate nested common interval* is a nested common interval with at most δ gaps. Blin, Faye, and Stoye also studied the problem of finding all maximal approximate nested common intervals. For permutations, they had an $O(n3^\delta)$-time algorithm. For sequences, they had $O(n^3 15^\delta)$-time algorithms for the uniqueness and the free-inclusion models.

In this paper, we study the problem of finding all nested common intervals of two sequences. Efficient algorithms are presented for all the three models defined in [5]. More specifically, for the uniqueness and the bijection models, we give $O(n + N_{\text{out}})$-time algorithms; and for the free-inclusion model, we give an $O(n^{1+\varepsilon} + N_{\text{out}})$-time algorithm, where $\varepsilon > 0$ is an arbitrarily small constant. We also present an upper bound on the size of the output for each model. For the uniqueness and the free-inclusion models, we show that $N_{\text{out}} = O(n^2)$. Consequently, our algorithms for the uniqueness and the free-inclusion models improve the previous $O(n^3)$-time algorithms in [5]. Let $C = \sum_{\alpha \in \Sigma} o_1(\alpha)o_2(\alpha)$, where Σ is the set of distinct genes, and $o_1(\alpha)$ and $o_2(\alpha)$ are, respectively, the numbers of copies of α in the two given sequences. For the bijection model, we show that $N_{\text{out}} = O(Cn)$. In the worst case, $C = n^2$ and thus the worst-case time complexity of our algorithm on the bijection model is $O(n^3)$. Since the best-case time complexity of the $O(n^3)$-time algorithm in [5] is the same as the worst case, Blin, Faye, and Stoye suggested developing an more efficient algorithm for the bijection model as a future work. As compared with their algorithm, our new algorithm is more practical, as C is likely to be much smaller than n^2 in practice [27]. Moreover, our algorithm is output-sensitive. In many practical cases, the number of (nested) common intervals is small [5, 25].

The remainder of this paper is organized as follows. Section 2 introduces notation and definitions that are used throughout this paper. Section 3 presents algorithms for the problem of finding all nested common intervals of two sequences on the uniqueness, the free-inclusion, and the bijection models. Finally, Section 4 concludes this paper.

2. Notation and definitions

A genome is model as a string over an alphabet Σ of homology families. A string is a *permutation* if each element in Σ appears exactly once, otherwise it is a *sequence*. Let S be a string with length $|S|$. We use $\Sigma(S)$ to denote the set of homology families which appear in S. The i^{th} element of S is denoted by $S[i]$, where $1 \le i \le |S|$. For any indices i and j such that $1 \le i \le j \le |S|$, the substring of S consisting of $S[i]$, $S[i + 1]$, …, and $S[j]$ is denoted by $S[i, j]$. For each $\alpha \in \Sigma$, let $occ(\alpha, S)$ be the number of copies of α in S. For example, let $S = (d_1, e_2, d_3, f_4, d_5, a_6, e_7, g_8, e_9)$, where the letters represent homology families and the numbers in the subscript denote indices. In this example, we have $S[2, 6] = (e_2, d_3, f_4, d_5, a_6)$, $\Sigma(S) = \{a, d, e, f, g\}$, and $occ(e, S) = 3$.

For any integers i and j, where $i \le j$, we use interval $[i, j]$ to denote the set of all integers in the range i to j. The common intervals and nested common intervals of two permutations S_1 and S_2 are defined as follows. A pair of intervals $([i_1, j_1], [i_2, j_2])$, where $1 \le i_1 < j_1 \le n$ and $1 \le i_2 < j_2 \le n$, is a *common interval* of S_1 and S_2 if $\Sigma(S_1[i_1, j_1]) = \Sigma(S_2[i_2, j_2])$. Note that this definition excludes common intervals of size one, since they are not considered in the definition of a nested common interval. A common interval $([i_1, j_1], [i_2, j_2])$ of S_1 and S_2 is called a *nested common interval* if either (1) $|[i_1, j_1]| = |[i_2, j_2]| = 2$, or (2) $[i_1, j_1]$ contains a sub-interval X_1 and $[i_2, j_2]$ contains a sub-interval X_2 such that (X_1, X_2) is a nested common interval of size $|X_1| = |[i_1, j_1]| - 1$. Clearly, in this definition, X_1 is either $[i_1 + 1, j_1]$ or $[i_1, j_1 - 1]$, and X_2 is either $[i_2 + 1, j_2]$ or $[i_2, j_2 - 1]$. For example, consider $S_1 = (a_1, b_2, c_3, d_4, e_5, f_6, g_7, h_8)$ and $S_2 = (e_1, g_2, f_3, h_4, b_5, d_6, a_7, c_8)$. In this example, $([6, 7], [2, 3])$, $([5, 7], [1, 3])$, $([5, 8], [1, 4])$ are, respectively, nested common intervals of sizes 2, 3, and 4; and $([1, 4], [5, 8])$ is a common interval, but is not a nested common interval.

It is easy to extend the definition of a common interval to sequences. However, since several copies of a gene may appear in a substring, the definition of nestedness in sequences is subtle. A nested common interval of size s could be extended to produce a nested common interval of size $s + 1$. Depending on the treatment one wants to apply when, during the extension of an interval, an element that already inside the interval is met once again, Blin, Faye, and Stoye [5] proposed the following three models to define a nested common interval of two sequences S_1 and S_2.

(1) It is forbidden to include a second copy of a gene in a nested common interval, which is called the *uniqueness* model in this paper.

(2) We can extend a nested common interval to include a gene that is already inside it "for free", caring about only the "innermost occurrence". We call this model the *free-inclusion* model. More precisely, in this model, a common interval $([i_1, j_1], [i_2, j_2])$ is a *nested common interval* if either (1) $|\Sigma(S_1[i_1, j_1])| = |\Sigma(S_2[i_2, j_2])| = 2$, or (2) $[i_1, j_1]$ contains a sub-interval $[i'_1, j'_1]$ and $[i_2, j_2]$ contains a sub-interval $[i'_2, j'_2]$ such that $([i'_1, j'_1], [i'_2, j'_2])$ is a nested common interval of size $|\Sigma(S_1[i'_1, j'_1])| = |\Sigma(S_1[i_1, j_1])| - 1$.

(3) If $([i_1, j_1], [i_2, j_2])$ is a nested common interval, each gene in $S_1[i_1, j_1]$ must match a unique gene of the same family in $S_2[i_2, j_2]$. We call this model the *bijection* model. More formally, in this model, a common interval $([i_1, j_1], [i_2, j_2])$ is a *nested common interval* if either (1) $|[i_1, j_1]| = |[i_2, j_2]| = 2$, or (2) there exist $x \in \{i_1, j_1\}$ and $y \in \{i_2, j_2\}$ such that $S_1[x] = S_2[y]$ and $([i_1, j_1] \setminus \{x\}, [i_2, j_2] \setminus \{y\})$ is a nested common interval.

For example, consider two sequences $S_1 = (a_1, b_2, a_3, e_4, a_5, a_6, d_7, c_8)$ and $S_2 = (d_1, a_2, c_3, b_4, a_5, a_6, b_7, e_8)$. The common interval ([2, 5], [5, 8]) is nested on the bijection model, but is not on the uniqueness model, since the family "a" occurs more than once. The common interval ([1, 6], [4, 8]) is nested on the free-inclusion model, but is not on the bijection model, since we can not find a bijection, from the genes in $S_1[1, 6]$ to the genes in $S_2[4, 8]$, to satisfy the nestedness property.

For the first two models, we define the *size* of a nested common interval $([i_1, j_1], [i_2, j_2])$ to be $|\Sigma(S_1[i_1, j_1])|$ (i.e., the number of distinct families in $S_1[i_1, j_1]$); while for the last model, we define the *size* to be the length $|[i_1, j_1]|$.

Consider the free-inclusion model. Since an interval can *freely* include genes that are already inside, several nested common intervals may indeed refer to the same nested common interval. We say that a nested common interval $([i_1, j_1], [i_2, j_2])$ is *closed* if none of $S_1[i_1 - 1]$, $S_1[j_1 + 1]$, $S_2[i_2 - 1]$, $S_2[j_2 + 1]$ belongs to $\Sigma(S_1[i_1, j_1])$. For example, if $S_1 = (a_1, c_2, a_3, a_4, b_5, b_6, d_7, b_8)$ and $S_2 = (a_1, b_2, g_3, a_4, b_5, b_6, c_7, d_8)$, then ([3, 6], [4, 6]) is a closed nested common interval, but ([4, 6], [4, 5]) is not, since it can be freely extended to include $S_1[3]$ and $S_2[6]$. By definition, if $([i_1, j_1], [i_2, j_2])$ is a closed nested common interval, it can not be extended to any of $S_1[i_1 - 1]$, $S_1[j_1 + 1]$, $S_2[i_2 - 1]$, $S_2[j_2 + 1]$ to obtain a longer nested common interval of the same size. In the bijection and the uniqueness models, we are interested in finding all nested common intervals. However, in the free-inclusion model, we are only interested in finding all closed nested common intervals.

3. Finding the nested common intervals of two sequences

Let S_1 and S_2 be two sequences of length n. Section 3.1 presents an $O(n + N_{\text{out}})$-time algorithm for finding all nested common intervals of S_1 and S_2 on the bijection model. Sections 3.2 and 3.3 extend the algorithm in Section 3.1 to the uniqueness and the free-inclusion models, respectively.

3.1 The bijection model

Blin, Faye, and Stoye's [5] had an $O(n^3)$-time algorithm for finding all nested common intervals of S_1 and S_2 on the bijection model. Their algorithm is described in Section 3.1.1. Then, our new algorithm is presented in Section 3.1.2.

3.1.1 Blin, Faye, and Stoye's algorithm

First, a directed acyclic graph G is constructed. (Figure 1 gives an illustration, which comes from [5].) The vertex set of G represents all possible candidates of nested common intervals. More specifically, each pair of intervals $([i_1, j_1], [i_2, j_2])$ is represented by a vertex (i_1, j_1, i_2, j_2), where $1 \leq i_1 < j_1 \leq n$, $1 \leq i_2 < j_2 \leq n$, and $S_1[i_1, j_1]$ and $S_2[i_2, j_2]$ contain the same multiset of families. In the bijection model, two intervals $[i_1, j_1]$ and $[i_2, j_2]$ can form a nested common interval only if $\|[i_1, j_1]\| = \|[i_2, j_2]\|$. Therefore, the number of vertices in G is $O(n^3)$.

In G, a directed edge is drawn from a vertex $v = (i_1, j_1, i_2, j_2)$ to a vertex $v' = (i'_1, j'_1, i'_2, j'_2)$ if and only if there exist $x \in \{i_1 - 1, j_1 + 1\}$ and $y \in \{i_2 - 1, j_2 + 1\}$ such that $S_1[x] = S_2[y]$ and $([i'_1, j'_1], [i'_2, j'_2]) = ([i_1, j_1] \cup \{x\}, [i_2, j_2] \cup \{y\})$. An edge from a vertex v to a vertex v' indicates that if v is a nested common interval, then v' is a nested common interval. Each vertex has an out-degree at most four. Thus, the number of edges in G is $O(n^3)$.

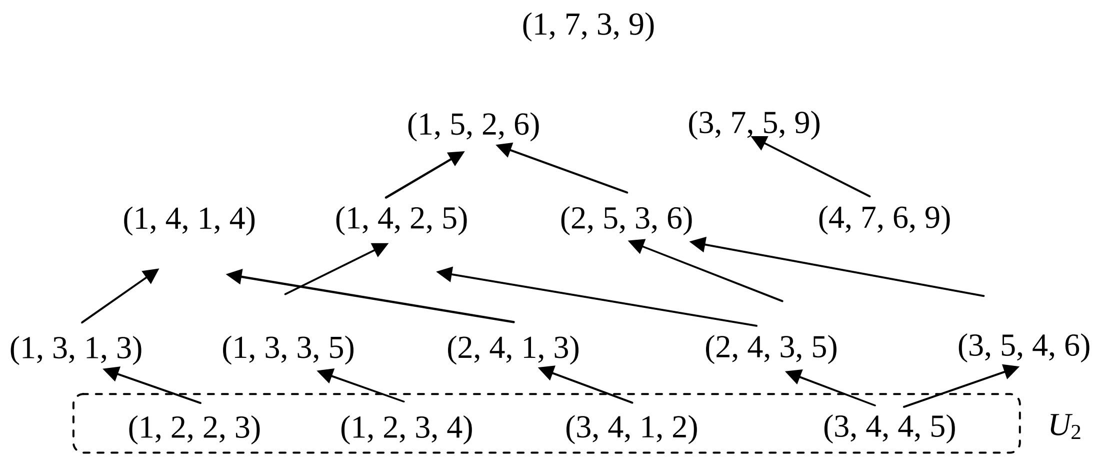


Figure 1. Graph G for $S_1 = (c_1, a_2, b_3, c_4, d_5, e_6, f_7)$ and $S_2 = (b_1, c_2, a_3, c_4, b_5, d_6, f_7, c_8, e_9)$.

Let U_2 be the set of vertices in G corresponding to multisets of size two. By definition, the vertices in U_2 represent nested common intervals of size two. Each vertex in G is a nested common interval if and only if it can be reached from a vertex in U_2. Therefore, all nested common intervals can be found by performing a simple graph-searching algorithm on G, starting from the vertices in U_2.

Blin, Faye, and Stoye showed that G can be constructed in $O(n^3)$ time. Consequently, the time complexity of the above algorithm is $O(n^3)$.

3.1.2 An $O(n + N_{out})$-time algorithm

Our new algorithm is a modified version of Blin, Faye, and Stoye's. It also solves the problem by finding all vertices in G that can be reached from the vertices in U_2. If the construction of the whole graph G is necessary, since the number of vertices and edges is $O(n^3)$, Blin, Faye, and Stoye's algorithm is optimal. Our idea is to do the finding without constructing the whole graph.

We classify the vertices of G into two classes: essential vertices and redundant vertices. A vertex is *essential* if it is a nested common interval, otherwise it is *redundant*. For $s \geq 2$, let U_s be the set of essential vertices corresponding multisets of size s. Our problem is to compute all nonempty sets U_s. The set U_2 is the set of vertices in G corresponding to multisets of size two. For $s > 2$, according to the nestedness property, each vertex in U_s is reachable from a vertex in U_{s-1}.

Therefore, we can generate the vertices in U_s by using the vertices in U_{s-1}. Our algorithm works as follow. First, we find the set U_2. Then, iteratively, we find the set U_s by using the vertices in U_{s-1} for each $s > 2$. The computation of U_2 is done by the following procedure.

```
Procedure SIZETWOINTERVAL ($S_1$, $S_2$)
begin
1.  $I_1 \leftarrow \{[i_1, i_1 + 1] \mid 1 \le i_1 < n\}$; $I_2 \leftarrow \{[i_2, i_2 + 1] \mid 1 \le i_2 < n\}$
2.  $\Delta \leftarrow \{\{S_1[i_1], S_1[i_1 + 1]\} \mid 1 \le i_1 < n\} \cup \{\{S_2[i_2], S_2[i_2 + 1]\} \mid 1 \le i_2 < n\}$
3.  for each set $M \in \Delta$ do
4.  begin
5.        $L_1(M) \leftarrow$ a list containing the intervals $[i_1, i_1 + 1]$ in $I_1$ with $\{S_1[i_1], S_1[i_1 + 1]\} = M$
6.        $L_2(M) \leftarrow$ a list containing the intervals $[i_2, i_2 + 1]$ in $I_2$ with $\{S_2[i_2], S_2[i_2 + 1]\} = M$
7.  end
8.  $U_2 \leftarrow \{(i_1, i_1 + 1, i_2, i_2 + 1) \mid [i_1, i_1 + 1] \in L_1(M), [i_2, i_2 + 1] \in L_2(M), M \in \Delta\}$
end
```

Using radix sort, the intervals in I_1 and I_2 can be sorted to obtain Δ and all $L_1(M)$ and $L_2(M)$ in $O(n)$ time. Therefore, lines 1-7 requires linear time. Consequently, SIZETWOINTERVAL takes $O(n + |U_2|)$ time.

We proceed to discuss the computation of U_s for a fixed $s > 2$. For ease of discussion, for each vertex (i_1, j_1, i_2, j_2), we define its LL-, LR-, RL-, and RR-extensions, respectively, to be the vertices $(i_1 - 1, j_1, i_2 - 1, j_2)$, $(i_1 - 1, j_1, i_2, j_2 + 1)$, $(i_1, j_1 + 1, i_2 - 1, j_2)$, and $(i_1, j_1 + 1, i_2, j_2 + 1)$. The LL-extension of (i_1, j_1, i_2, j_2) is *feasible* if $S_1[i_1 - 1] = S_2[i_2 - 1]$, and is *infeasible* otherwise. The feasibilities of LR-, RL-, and RR-extensions are defined similarly. For each vertex $v = (i_1, j_1, i_2, j_2)$ in U_{s-1}, the vertices in U_s that are reachable from v correspond to the feasible extensions of (i_1, j_1, i_2, j_2). Therefore, U_s is constructed as follows. First, we generate all feasible extensions of the vertices in U_{s-1} and store them in an array E. Since each vertex has at most four feasible extensions, $|E| \le 4|U_{s-1}|$ and thus this step takes $O(|U_{s-1}|)$ time. Each vertex in U_s is contained in E. However, since a vertex may appear more than once in E, we need to remove duplicates. All vertices in U_s are nested common intervals of the same size s. Therefore, two elements (i_1, j_1, i_2, j_2) and (i'_1, j'_1, i'_2, j'_2) in E are the same vertex if and only if $(i_1, i_2) = (i'_1, i'_2)$. To remove duplicates, we sort the elements $(i_1, j_1,$

i_2, j_2) in E according to the lexicographic order of (i_1, i_2) and then delete duplicates by a simple scan on the sorted sequence. After the removal, E stores the set of vertices of U_s.

The bottleneck of the above construction is to sort the elements in E. We implement this step by radix sort. More specifically, we sort the elements two times with a stable sort: first on i_2 and next on i_1. Since all indices i_1 and i_2 are integers in the range 1 to n, by counting sort, this step can be done in $O(n + |U_{s-1}|)$ time. In the following, we show that the time can be reduced to $O(|U_{s-1}|)$, so that the overall time complexity is $O(n + N_{\text{out}})$. We start with the following simple result.

Lemma 1. Let K be a set of integers in the range 1 to n, where $|K| < n$, and let K^* be the sorted sequence of the integers in K. Let A be a sequence of integers drawn from K. Given K^*, we can stably sort the integers in A in $O(|K| + |A|)$ time.

Proof: The sorting is done by counting sort, with modifications, as follows. Two auxiliary arrays are used: the array $B[1..|A|]$ is used to hold the sorted output, and the array $C[1..n]$ is used to count the occurrences of numbers. First, we set $C[K^*[i]] = 0$ for each i, $1 \le i \le |K^*|$. Next, we increase $C[A[i]]$ by one for each i, $1 \le i \le |A|$. Then, we compute $C[K^*[i]] = C[K^*[i]] + C[K^*[i-1]]$ for $i = 2, 3, ..., |A|$. Finally, for each number in A, taken in reverse order, we do the following: place $A[i]$ into $B[C[A[i]]]$ and decrease $C[A[i]]$ by one. The above computation takes $O(|K| + |A|)$ time. Thus, the lemma holds. $\square$

For brevity, we only describe how to sort the elements in E according to i_1. For $s \ge 2$, an additional array $I_1(s)$ is maintained to store the distinct i_1 indices of the vertices in U_s, in increasing order. Clearly, $I_1(2)$ can be computed in $O(n)$ time. Using $I_1(s-1)$, the sorting according to i_1 is done as follows. Let $I_1^-(s-1)$ be the sequence obtained from $I_1(s-1)$ by decreasing each number by one. Since each element in E is an extension of a vertex in U_{s-1}, all i_1 indices of the elements in E are integers in $I_1(s-1)$ or $I_1^-(s-1)$. Let I^* be an array that stores the set of distinct integers in $I_1(s-1)$ and $I_1^-(s-1)$ in increasing order. By Lemma 1, given I^*, we can do the sorting for i_1 in $O(|I_1(s)| + |E|) = O(|U_{s-1}|)$ time. The array I^* can be built in $O(|I_1(s-1)|)$ time by merging $I_1(s-1)$ and $I_1^-(s-1)$ and then removing duplicates. Therefore, given U_{s-1} and $I_1(s-1)$, U_s can be computed in $O(|U_{s-1}|)$

time. After sorting the elements in E according to i_1, it is easy to obtain the sorted array $I_1(s)$.

In summary, U_2 can be computed in $O(n + |U_2|)$ time, and for $s > 2$ each set U_s can computed in $O(|U_{s-1}|)$ time. Therefore, we obtain the following.

Theorem 1. All nested common intervals of two sequences of length n on the bijection model can be found in $O(n + N_{\text{out}})$ time.

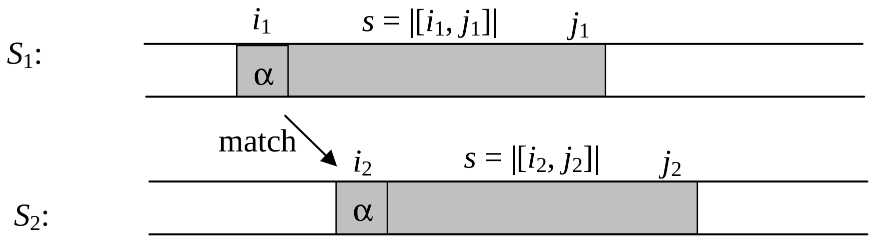


Figure 2. A feasible LL-extension in the bijection model.

In the remainder of this section, we show that the number of output, N_{out}, is $O(Cs_{\max})$, where $C = \sum_{\alpha\in\Sigma} occ(\alpha, S_1)occ(\alpha, S_2)$ and $s_{\max} \le n$ is the size of the largest nested common interval. Clearly, each pair $(S_1[i_1], S_2[i_2])$ with $S_1[i_1] = S_2[i_2]$ contributes at most four vertices to U_2. Thus, the number of essential vertices in U_2 is $O(C)$. All other essential vertices are feasible LL-, LR-, RL-, and RR-extensions of vertices. Let N_{LL}, N_{LR}, N_{RL}, and N_{RR} be, respectively, the numbers of essential vertices that are feasible LL-, LR-, RL-, and RR-extensions of vertices. In the following, we show that $N_{\text{LL}} \le Cs_{\max}$. Let $v = (i_1, j_1, i_2, j_2)$ be an essential vertex that is the feasible LL-extension of a vertex. (See Figure 2.) By definition, $S_1[i_1] = S_2[i_2]$. Since $|[i_1, j_1]| = |[i_2, j_2]|$, we can write (i_1, j_1, i_2, j_2) as $(i_1, i_1 + s - 1, i_2, i_2 + s - 1)$, where $s = |[i_1, j_1]|$. Thus, the feasible LL-extension of a vertex can be uniquely specified by a 3-tuple (i_1, i_2, s), where $S_1[i_1] = S_2[i_2]$ and $s \le s_{\max}$. The number of all such 3-tuples is at most $Cs_{\max}$, from which we conclude that $N_{\text{LL}} \le Cs_{\max}$. Similarly, N_{LR}, N_{RL}, and N_{RR} are all bounded by $Cs_{\max}$. Consequently, we obtain the following.

Theorem 2. In the bijection model, the number of nested common intervals of two sequences of length n is $O(Cs_{\max})$.

3.2 The uniqueness model

In the uniqueness model, it is forbidden to include a second copy of a gene in a nested common interval. We find all nested common intervals in this model by the algorithm in Section 3.1.2 with the following modifications.

1. We add the following constraint on each vertex (i_1, j_1, i_2, j_2) in U_s: $|\Sigma(S_1[i_1, j_1])| = |[i_1, j_1]| = |\Sigma(S_2[i_2, j_2])| = |[i_2, j_2]|$.
2. To check whether the LL-extension of a vertex (i_1, j_1, i_2, j_2) is feasible, we add the following constraint: $S_1[i_1 - 1] \notin \Sigma(S_1[i_1, j_1])$. The feasibilities of LR-, RL-, and RR-extensions are redefined similarly.

By definition, $U_2 = \{(i_1, i_1 + 1, i_2, i_2 + 1) \mid S_1[i_1] \neq S_1[i_1 + 1], S_2[i_2] \neq S_2[i_2 + 1], \{S_1[i_1], S_1[i_1 + 1]\} = \{S_2[i_2], S_2[i_2 + 1]\}, 1 \leq i_1, i_2 < n\}$. Clearly, using radix sort, U_2 can be computed in $O(n + |U_2|)$ time. To generate the feasible LL-, LR-, RL-, and RR-extensions of a vertex (i_1, j_1, i_2, j_2) in U_{s-1}, we need to test whether $S_1[i_1 - 1]$ (or $S_1[j_1 + 1]$) occurs in $S_1[i_1, j_1]$. As indicated in [15], this test can be done efficiently as follows. We pre-compute two arrays $Prev[1..n]$ and $Next[1..n]$, where $Prev[i]$ and $Next[i]$ store, respectively, the previous and next occurrences of $S_1[i]$ in S_1, $1 \leq i \leq n$. It is easy to compute these two arrays in $O(n)$ time. Then, the test for $S_1[i_1 - 1]$ (or $S_1[j_1 + 1]$) is done in $O(1)$ time by checking whether $Next[i_1 - 1] \leq j_1$ (or $Prev[j_1 + 1] \geq i_1$).

With the above modifications, for $s > 2$ each set U_s can still be computed in $O(|U_{s-1}|)$ time. Therefore, we obtain the following.

Theorem 3. All nested common intervals of two sequences of length n on the uniqueness model can be found in $O(n + N_{\text{out}})$ time.

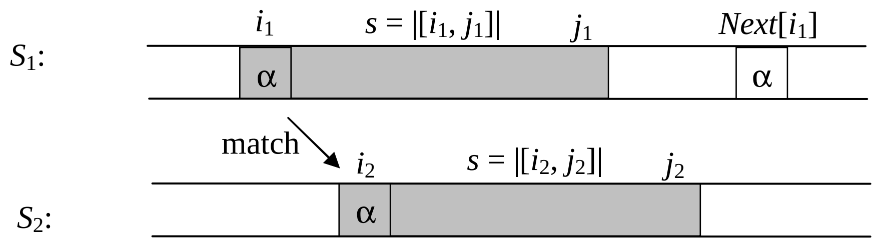


Figure 3. A feasible LL-extension in the uniqueness model.

Let N_{LL}, N_{LR}, N_{RL}, and N_{RR} be defined the same as in Section 3.1.2. In the following, we show that $N_{LL} \leq n^2$. Similar to the bijection model, the feasible LL-extension of a vertex can be uniquely specified by a 3-tuple (i_1, i_2, s) with $S_1[i_1] = S_2[i_2]$ and $s \leq s_{\max}$. (See Figure 3.) Moreover, since it is forbidden to include a second copy of a gene, s should be less than $Next(i_1) - i_1$; otherwise, $j_1 \geq Next[i_1]$ and $S_1[i_1]$ occurs twice in $S_1[i_1, j_1]$. For convenience, we call a 3-tuple (i_1, i_2, s) *feasible* if $S_1[i_1] = S_2[i_2]$ and $s < Next(i_1) - i_1$. Then, our problem is to count the number of all feasible 3-tuples. Consider a fixed index $i_2 = a$. Let $(x_1, x_2, ..., x_k)$ be the positions in S_1 at which $S_2[a]$ occurs. Note that by definition, $Next(x_i) = x_{i+1}$ for $1 \leq i < k$. The set of all feasible 3-tuples (i_1, i_2, s) with $i_2 = a$ is $\{(x_1, a, s) \mid 1 \leq s < x_2 - x_1\} \cup \{(x_2, a, s) \mid 1 \leq s < x_3 - x_2\} \cup ... \cup \{(x_k, a, s) \mid 1 \leq s < (n + 1) - x_k\}$. Therefore, the number of all feasible 3-tuples (i_1, i_2, s) with $i_2 = a$ is less than $(x_2 - x_1) + (x_3 - x_2) + ... + ((n + 1) - x_k) = n + 1 - x_1 \leq n$. Consequently, the number of all feasible 3-tuples is less than n^2. Similarly, N_{LR}, N_{RL}, and N_{RR} are all smaller than n^2. Therefore, we obtain the following.

Theorem 4. In the uniqueness model, the number of nested common intervals of two sequences of length n is $O(\min\{n^2, Cs_{\max}\})$.

3.3 The free-inclusion model

Recall that in the free-inclusion model, the size of a nested common interval $([i_1, j_1], [i_2, j_2])$ is defined to be the number of distinct families in $S_1[i_1, j_1]$, instead of the length $|[i_1, j_1]|$. Also recall that a nested common interval $([i_1, j_1], [i_2, j_2])$ is closed, if we can not include any of $S_1[i_1 - 1]$, $S_1[j_1 + 1]$, $S_2[i_2 - 1]$, $S_2[j_2 + 1]$ to obtain longer nested common intervals of the same size. For the problem of finding all closed nested common intervals of two sequences, Blin, Faye, and Stoye [5] had an $O(n^3)$-time algorithm. In this section, we gives an improved algorithm that requires $O(n^{1+\varepsilon} + N_{\text{out}})$ time.

Section 3.3.1 gives a data structure that is useful in the extension of a nested common interval. Section 3.3.2 describes the improved algorithm.

3.3.1. A data structure

To simplify notation, we assume that a string S is extended on both ends by a terminal character $0 \notin \Sigma$ (i.e., $S[0] = S[|S| + 1] = 0$). Let S be a string of length n and $[i, j]$ be an interval. We define $\alpha(S, i, j)$ to be the largest index $k < i$ with $S[k] \notin \Sigma(S[i, j])$, and define $\beta(S, i, j)$ to be the smallest index $k > j$ with $S[k] \notin \Sigma(S[i, j])$. For example, if $S = (a_1, e_2, d_3, f_4, d_5, d_6, a_7, a_8, f_9, g_{10}, e_{11})$, we have $\alpha(S, 4, 7) = 2$ and $\beta(S, 4, 7) = 10$. In this section, we show that with an $O(n^{1+\varepsilon})$-time preprocessing, $\alpha(S, i, j)$ and $\beta(S, i, j)$ can be found in $O(1)$ time for any interval $[i, j]$. By symmetry, only the preprocessing for computing $\beta(S, i, j)$ is described.

For an array A and an integer pair (i, j), define $\pi(A, i, j)$ to be the smallest index $k \geq j$ such that $A(k) < i$. Let *Prev* be an array in which $Prev[k]$ stores the previous occurrence of $S[k]$ in S, $1 \leq k \leq n$. If $Prev[k]$ does not exist, let $Prev[k] = -\infty$. Then, it is easy to see that the first element in $S[j + 1, n]$ that does not occur in $S[i, j]$ is also the first element in $S[j + 1, n]$ with $Prev[k] < i$. Thus, $\beta(S, i, j)$ can be computed by finding $\pi(Prev, i, j + 1)$. Let A be an array containing n integers in the range 1 to n. In the following, we discuss how to preprocess A so that a query $\pi(A, i, j)$ with $(i, j) \in [1, n] \times [1, n]$ can be answered in $O(1)$ time.

A naïve approach is to use a table to store $\pi(A, i, j)$ for each $(i, j) \in [1, n] \times [1, n]$. For a fixed $i \in [1, n]$, all $\pi(A, i, j)$ can be computed in $O(n)$ time as below. Initially, set a variable $k = \infty$. Then, from left to right, we check each $A[j]$ and do the following: if $A[j] < i$, we update k to j; and then we set $\pi(A, i, j) = k$. We obtain the following.

Lemma 2. We can construct a table of size $O(n^2)$ in $O(n^2)$ time for A so that each query $\pi(A, i, j)$ with $(i, j) \in [1, n] \times [1, n]$ can be answered in $O(1)$ time.

Using a standard multi-level scheme [2, 21], the preprocessing time and space in Lemma 3 can be reduced to $O(n^{1+\varepsilon})$ for any constant $\varepsilon > 0$. For brevity, we only show how to reduce the time and space to $O(n^{1.5})$. We first establish the following simple result.

Lemma 3. Let X be an array containing q integers in the range 1 to n, where $q < n$. Using $O(n)$ space and $O(n)$ preprocessing time, a query $\pi(X, i, j)$ with $(i, j) \in [1, n] \times [1, q]$ can be reduced in $O(1)$ time to a query with $(i, j) \in [1, q] \times [1, q]$.

Proof: Define the *rank* of a number a in X to be the number of elements of X that are less than or equal to a. We construct an array $X'[1..q]$ in which $X'[k]$ stores the rank of $X[k]$ in X; and we construct a mapping table $M[1..n]$ in which $M[i]$ stores the rank of i in X. Then, a query $\pi(X, i, j)$ with $(i, j) \in [1, n] \times [1, q]$ is reduced to $\pi(X', M[i], j)$. Using counting sort, the above computation takes $O(n)$ time. Thus, the lemma holds. □

Let $q = n^{0.5}$. First, partition A into q blocks $A_1, A_2, \ldots, A_q$ of size q. Next, for each A_k, we perform range reduction and construct a table of size (q^2) based on Lemmas 2 and 3, so that each query $\pi(A_k, i, j)$ with $(i, j) \in [1, n] \times [1, q]$ can be answered in $O(1)$ time. Let $B[1..q]$ be an array in which $B[k]$ stores the minimum in A_k. We also perform range reduction and apply the naïve approach to preprocess B. It is easy to check that the preprocessing time and the space requirement for B and all blocks A_k are $O(n^{1.5})$.

Given a pair $(i, j) \in [1, n] \times [1, n]$, $\pi(A, i, j)$ is reported as follows. Assume that $A[j]$ is the x^{th} element of block A_y. First, we find $k = \pi(A_y, i, x)$. If $k \neq \infty$, the answer is the position of $A_y[k]$ in A. Otherwise, we find $z = \pi(B, i, y + 1)$. If $z = \infty$, no element in $A[j + 1..n]$ is less than i. Otherwise, the answer is in A_z, which can be found by querying $\pi(A_z, i, 1)$. In summary, we have the following.

Lemma 4. Let S be a string of length n. With an $O(n^{1+\varepsilon})$-time preprocessing, $\alpha(S, i, j)$ and $\beta(S, i, j)$ can be found in $O(1)$ time for any interval $[i, j]$.

3.3.2 An improved algorithm

To describe our algorithm, we need a few more definitions. Let S be a string and $[i, j]$ be an interval. We call $[i, j]$ a *closed interval* of S if neither of $S[i - 1]$ and $S[j + 1]$ occurs in $S[i, j]$. The *size* of a closed interval $[i, j]$ of S is $|\Sigma(S[i, j])|$. We define $c(S, i, j)$ to be the longest interval $[i', j'] \supseteq [i, j]$ with $\Sigma(S[i', j']) = \Sigma(S[i, j])$. Note that by definition, $c(S, i, j) = [\alpha(S, i, j) + 1, \beta(S, i, j) - 1]$ and $c(S, i, j)$ is closed.

Lemma 5. Let $[a, b]$ be a closed interval of a string S. Let $c(S, a - 1, b) = [a', b']$ and $c(S, a, b + 1) = [a'', b'']$. Then, a'' is either a or a'.

Proof: If $S[a - 1] \neq S[b + 1]$, we have $S[a - 1] \notin \Sigma(S[a, b + 1])$ and thus $a'' = a$. If $S[a - 1] = S[b + 1]$, we have $c(S, a - 1, b) = c(S, a, b + 1)$ and thus $a'' = a'$. Therefore, the lemma holds. $\square$

Lemma 6. Let $[a, b]$ and $[c, d]$ be two closed intervals of a string S that are of the same size. Let $c(S, a - 1, b) = [a', b']$ and $c(S, c - 1, d) = [c', d']$. If $a \leq c$, we have $a' \leq c'$.

Proof: Let s be the size of $[a, b]$ and $[c, d]$. Then, $[a', b']$ and $[c', d']$ are of size $s + 1$. Since $S[a - 1] \notin \Sigma(S[a, b])$ and $S[a' - 1] \notin \Sigma(S[a', b])$, we have $|\Sigma(S[a' - 1, b])| = s + 2$. The two intervals $[a, b]$ and $[c, d]$ are closed intervals of the same size. Thus, from $a \leq c$, it is easy to conclude that $b \leq d$. And therefore, $|\Sigma(S[a' - 1, d])| \geq |\Sigma(S[a' - 1, b])| \geq s + 2$. If $a' > c'$, $|\Sigma(S[c', d'])| \geq |\Sigma(S[c', d])| \geq |\Sigma(S[a' - 1, d])| \geq s + 2$, which contradicts to $|\Sigma(S[c', d'])| = s + 1$. Therefore, $a' \leq c'$ and thus the lemma holds. $\square$

We proceed to present an algorithm for computing all closed nested common intervals. Assume that S_1 and S_2 are preprocessed, so that $\alpha(S_1, i, j)$, $\beta(S_1, i, j)$, $\alpha(S_2, i, j)$, $\beta(S_2, i, j)$ can be computed in $O(1)$ time for any interval $[i, j]$. Similar to the algorithm in Section 3.1.2, each pair of intervals $([i_1, j_1], [i_2, j_2])$ is represented by a vertex (i_1, j_1, i_2, j_2). For $s \geq 2$, let U_s be the set of vertices that are closed nested common interval of size s. The set U_2 is computed as follows. First, we compute an array A that stores the set $\{(i_1, i_1 + 1, i_2, i_2 + 1) \mid S_1[i_1] \neq S_1[i_1 + 1], S_2[i_2] \neq S_2[i_2 + 1], \{S_1[i_1], S_1[i_1 + 1]\} = \{S_2[i_2], S_2[i_2 + 1]\}, 1 \leq i_1, i_2 < n\}$. Note that the set stored in A is just the set U_2 in the uniqueness model. Next, we replace each element $(i_1, i_1 + 1, i_2, i_2 + 1)$ in A by $(c(S_1, i_1, i_1 + 1), c(S_2, i_2, i_2 + 1))$. (For convenience, we write $(c(S_1, i_1, j_1), c(S_2, i_2, j_2))$ for (i'_1, j'_1, i'_2, j_2'), where $[i'_1, j'_1] = c(S_1, i_1, j_1)$ and $[i'_2, j'_2] = c(S_2, i_2, j_2)$.) Finally, we obtain U_2 from A by removing duplicates. The above computation of U_2 takes $O(n + |U_2|)$ time. Note that $|U_2|$ is bounded by the number of elements in A, which is $O(C)$.

Next, we discuss the computation of U_s for $s > 2$. Consider a vertex $v = (i_1, j_1, i_2, j_2)$ in U_{s-1}. By definition, v is a closed nested common interval and thus none of $S_1[i_1 - 1]$, $S_1[j_1 + 1]$, $S_2[i_2 - 1]$, $S_2[j_2 + 1]$ occurs in $S_1[i_1, j_1]$. Let the LL-, LR-, RL-, and RR-extensions of v and their feasibilities be defined the same as in Section 3.1.2. If the LL-extension $(i_1 - 1, j_1, i_2 - 1, j_2)$ is feasible, we further define the *closed LL-extension* of v to be $(c(S_1, i_1 - 1, j_1), c(S_2, i_2 - 1, j_2))$. The *closed LR-*, *RL-*, and *RR-extensions* of v are defined similarly.

For $s > 2$, the set U_s is constructed as follows. First, we generate all closed LL-, LR-, RL-, and RR-extensions of the vertices in U_{s-1} and store them in an array E. This step takes $O(|U_{s-1}|)$ time. Next, we obtain the set U_s by removing duplicates in E. Since all elements in E are closed nested common intervals of the same size s, it is easy to conclude that two elements (i_1, j_1, i_2, j_2) and (i'_1, j'_1, i'_2, j'_2) in E are the same if and only if $(i_1, i_2) = (i'_1, i'_2)$. Therefore, similar to the bijection model, we can remove duplicates by simply sorting the elements (i_1, j_1, i_2, j_2) in E according to (i_1, i_2).

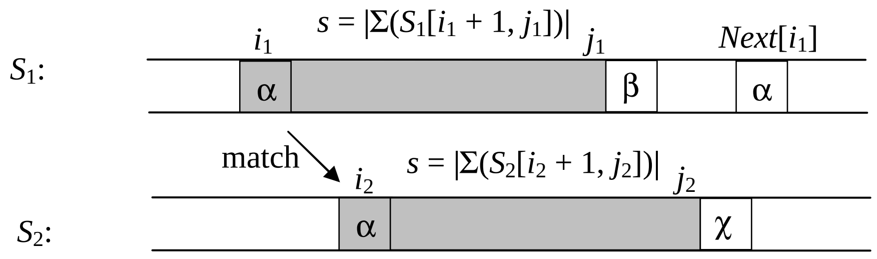
For brevity, we only describe how to do the sorting according to i_1 in $O(|U_{s-1}|)$ time. The array E stores the closed LL-, LR-, RL-, and RR-extensions of the vertices in U_{s-1}. For convenience, for each element v of E, we call the vertex in U_{s-1} for which v is generated the *seed* of v. Let $v = (i_1, j_1, i_2, j_2)$ be an element of E and (a_1, b_1, a_2, b_2) be its seed. By Lemma 5, i_1 is either a_1 or $\alpha(S_1, a_1 - 1, b_1) + 1$. We classify all elements (i_1, j_1, i_2, j_2) of E into two types: an element is of type (1) if its i_1 index is the same as its seed, and is of type (2) otherwise.

The sorting according to i_1 is done as follows. In Step 1, we sort the elements of type (1). In Step 2, we sort the elements of type (2). Finally, in Step 3, we merge the outputs of Steps 1 and 2 to obtain the whole sorted sequence. Let $I_1(s - 1)$ be defined the same as in Section 3.1.2. All i_1 indices of the elements of type (1) are integers in $I_1(s - 1)$. Thus, by Lemma 1, Step 1 can be done in $O(|U_{s-1}|)$ time by using $I_1(s - 1)$. Consider two elements $v = (i_1, j_1, i_2, j_2)$ and $v' = (i'_1, j'_1, i'_2, j'_2)$ of type (2). Let (a_1, b_1, a_2, b_2) and (c_1, d_1, c_2, d_2) be, respectively, the seeds of v and v'. By Lemma 6, i_1

$\leq i'_1$ if and only if $a_1 \leq c_1$. Thus, Step 2 can be done by sorting the elements of type (2) according to the i_1 indices of their seeds. And therefore, Step 2 can also be done in $O(|U_{s-1}|)$ time by using $I_1(s-1)$.

In summary, with an $O(n^{1+\varepsilon})$-time preprocessing, for $s > 2$ each set U_s can be constructed in $O(|U_{s-1}|)$ time. Therefore, we obtain the following.

Theorem 5. All closed nested common intervals of two sequences of length n on the free-inclusion model can be found in $O(n^{1+\varepsilon} + N_{\text{out}})$ time.



i_1
$s = |\Sigma(S_1[i_1 + 1, j_1])|$
j_1
$Next[i_1]$
S_1:
α
β
α
match
i_2
$s = |\Sigma(S_2[i_2 + 1, j_2])|$
j_2
S_2:
α
χ

Figure 4. A feasible LL-extension in the free-inclusion model, where $\alpha, \beta, \chi \notin \Sigma(S_1[i_1 + 1, j_1])$.

In this section, a vertex is *essential* if it represents a closed nested common interval of S_1 and S_2. As mentioned, the number of essential vertices in U_2 is $O(C)$. All other essential vertices are closed LL-, LR-, RL-, and LL-extensions of vertices. Let N_{LL} be the number of essential vertices that are closed LL-extensions of vertices. In the following, we show that $N_{\text{LL}} \leq n^2$. By definition, a closed LL-extension is obtained by extending the feasible LL-extension of an essential vertex. Therefore, N_{LL} is bounded by the number of all feasible LL-extensions of essential vertices. Let (i_1, j_1, i_2, j_2) be the feasible LL-extension of an essential vertex v. (See Figure 4.) By definition, $S_1[i_1] = S_2[i_2]$ and $v = (i_1 + 1, j_1, i_2 + 1, j_2)$ is a closed nested common interval. Let s be the size of v. Then, since $[i_1 + 1, j_1]$ is an s-sized closed interval of S_1, j_1 is the largest index $j \geq i_1 + 1$ with $|\Sigma(S_1[i_1 + 1, j])| = s$. Similarly, j_2 is the largest index $j > i_2 + 1$ with $|\Sigma(S_2[i_2 + 1, j])| = s$. Therefore, the feasible LL-extension (i_1, j_1, i_2, j_2) of each essential vertex can be uniquely specified by a 3-tuple (i_1, i_2, s), where $S_1[i_1] = S_2[i_2]$ and $s \leq s_{\max}$. Since $[i_1 + 1, j_1]$ is a closed interval of S_1, $S_1[i_1]$ does not occur in

$S_1[i_1 + 1, j_1]$. Thus, $j_1 < Next(i_1)$. Moreover, $s = |\Sigma(S_1[i_1 + 1, \quad j_1])| \leq j_1 - i_1$. Consequently, similar to the uniqueness model, we obtain $s < Next(i_1) - i_1$. Thus, N_{LL} is less than n^2. Also, the numbers of essential vertices that are closed LR-, RL-, and RR-extensions of vertices are all less than n^2. Therefore, we obtain the following.

Theorem 6. In the free-inclusion model, the number of closed nested common intervals of two sequences of length n is $O(\min\{n^2, Cs_{\max}\})$.

Remark 1. All nested common intervals found by the algorithm in this section are closed. If non-closed nested common intervals are also of interest, it is not difficult to modify our algorithm for the bijection model to find all nested common intervals in $O(n + N_{\mathrm{out}})$ time. However, the number of all nested common intervals may be as large as $O(n^4)$.

4. Concluding remarks

Our algorithm for finding all nested common intervals on the free-inclusion model requires an $O(n^{1+\varepsilon})$-time preprocessing. One direction for further study is to reduce the preprocessing time. Our algorithms in Section 3 find all nested common intervals. However, one may be interested in finding only maximal nested common intervals. Blin, Faye, and Stoye [5] had an optimal algorithm that finds all maximal nested common intervals of two permutations in linear time. Another direction for further study is to develop output-sensitive algorithms for finding all maximal nested common intervals of two sequences.

References

[1] M. P. Béal, A. Bergeron, S. Corteel, and M. Raffinot, "An algorithmic view of gene teams," *Theoretical Computer Science*, vol. 320, no. 2-3, pp. 395-418, 2004.

[2] J. L. Bentley and H. A. Maurer, "Efficient worst-case data structures for range searching," *Acta Informatica*, vol. 13, pp. 155-168, 1980.

[3] A. Bergeron, Y. Gingras, and C. Chauve, "Formal models of gene clusters," in I. Mandoiu and A. Zelikovskym, editors, *Bioinformatics Algorithms: Techniques and Applications*, chapter 8,

pp. 177-202, 2008, Wiley, New York.

[4] A. Bergeron and J. Stoye, "On the similarity of sets of permutations and its applications to genome comparison," *Journal of Computational Biology*, vol. 13, pp. 1340-1354, 2006.

[5] G. Blin, D. Faye, and J. Stoye, "Finding nested common intervals efficiently," *Journal of Computational Biology*, vol. 17, no. 9, pp. 1183-1194, 2010. (A preliminary version of this paper was presented at the 2009 RECOMB Comparative Genomics.)

[6] S. Böcker, K. Jahn, J. Mixtacki, and J. Stoye, "Computation of median gene clusters," *Journal of Computational Biology*, vol. 16, pp. 1085-1099, 2009.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, McGraw-Hill, 2nd ed., 2001.

[8] T. Dandekar, B. Snel, M. Huynen, and P. Bork. "Conservation of gene order: a fingerprint for proteins that physically interact," *Trends in Biochemical Sciences*, vol. 23, pp. 324-328, 1998.

[9] G. Didier, "Common intervals of two sequences," *Lecture Notes in Computer Science*, vol. 2812, pp. 17-24, 2003.

[10] G. Didier, T. Schmidt, J. Stoye, and D. Tsur, "Character sets of strings," *Journal of Discrete Algorithms*, vol. 5, pp. 330-340, 2007.

[11] M. D. Ermolaeva, O. White, and S. L. Salzberg, "Prediction of operons in microbial genomes," *Nucleic Acids Research*, vol. 29, no. 5, pp. 1216-1221, 2001.

[12] X. He and M. H. Goldwasser, "Identifying conserved gene clusters in the presence of homology families," *Journal of Computational Biology*, vol. 12, no. 6, pp. 638-656, 2005.

[13] S. Heber and J. Stoye, "Finding all common intervals of *k* permutations," *Lecture Notes in Computer Science*, vol. 2089, pp. 207-218, 2001.

[14] R. Hoberman and D. Durand,"The incompatible desiderata of gene cluster properties," *Lecture Notes in Bioinformatics*, vol. 3678, pp. 73-87, 2005.

[15] K. Jahn "Efficient computation of approximate gene clusters based on reference occurrences," in *Proceedings of the 8th RECOMB Comparative Genomics Satellite Workshop*, 2010, pp. 264-277.

[16] U. Kurzik-Dumke and A. Zengerle, "Identification of a novel Drosophila melanogaster gene, angel, a member of a nested gene cluster at locus 59F4,5, " *Biochimica et Biophysica Acta*, vol.

1308, pp. 177-181, 1996.

[17] W. C. Lathe III, B. Snel, and P. Bork, "Gene context conservation of a higher order than operons," *Trends in Biochemical Sciences*, vol. 25, pp. 474-479, 2000.

[18] J. Lawrence, "Selfish operons: the evolutionary impact of gene clustering in prokaryotes and eukaryotes," *Current Opinion in Genetics* & *Development*, vol. 9, no. 6, pp. 642-648, 1999.

[19] N. Luc, J. -L. Risler, A. Bergeron, and M. Raffinot, "Gene teams: a new formalization of gene clusters for comparative genomics," *Computational Biology and Chemistry*, vol. 27, no. 1, pp. 59-67, 2003.

[20] R. Overbeek, M. Fonstein, M. D'Souza, G. D. Pusch, and N. Maltsev, "The use of gene clusters to infer functional coupling," in *Proceedings of the National Academy of Sciences of the United States of America*, vol. 96, no. 6, pp. 2896-2901, 1999.

[21] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*. Springer, Heidelberg, 1985.

[22] S. Rahmann and G.W. Klau, "Integer linear programs for discovering approximate gene clusters," *Lecture Notes in Bioinformatics*, vol. 4175, pp. 298-309, 2006.

[23] T. Schmidt and J. Stoye, "Quadratic time algorithms for finding common intervals in two and more sequences," *Lecture Notes in Computer Science*, vol. 3109, pp. 347-359, 2004.

[24] B. Snel, P. Bork, and M. A. Huynen, "The identification of functional modules from the genomic association of genes," in *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99, no. 9, pp. 5890-5895, 2002.

[25] T. Uno and M. Yagiura, "Fast algorithms to enumerate all common intervals of two permutations," *Algorithmica*, vol. 26, no. 2, pp. 290-309, 2000.

[26] B.-F. Wang and C.-H. Lin, "Improved algorithms for finding gene teams and constructing gene team trees," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, accepted.

[27] B.-F. Wang, C.-C. Kuo, S.-J. Liu, and C.-H. Lin, "A new efficient algorithm for the gene team problem on general sequences," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, under second review.