Algorithms for Finding Gene Clusters

Steffen Heber[1] and Jens Stoye[2]

[1] Department of Computer Science & Engineering
University of California, San Diego
sheber@ucsd.edu
[2] Max Planck Institute for Molecular Genetics
Berlin, Germany
stoye@molgen.mpg.de

Abstract. Comparing gene orders in completely sequenced genomes is a standard approach to locate clusters of functionally associated genes. Often, gene orders are modeled as permutations. Given k permutations of n elements, a k-tuple of intervals of these permutations consisting of the same set of elements is called a *common interval*. We consider several problems related to common intervals in multiple genomes. We present an algorithm that finds all common intervals in a family of genomes, each of which might consist of several chromosomes. We present another algorithm that finds all common intervals in a family of circular permutations. A third algorithm finds all common intervals in signed permutations. We also investigate how to combine these approaches. All algorithms have optimal worst-case time complexity and use linear space.

1 Introduction

The conservation of gene order has been extensively studied so far [25, 19, 16, 12]. There is strong evidence that genes clustering together in phylogenetically distant genomes frequently encode functionally associated proteins [23, 4, 24] or indicate recent horizontal gene transfer [11, 5]. Due to the increasing amount of completely sequenced genomes, the comparison of gene orders to find conserved gene clusters is becoming a standard approach for protein function prediction [20, 17, 22, 6].

In this paper we describe efficient algorithms for finding gene clusters for various types of genomic data. We represent gene orders by permutations (re-orderings) of integers. Hence gene clusters correspond to intervals (contiguous subsets) in permutations, and the problem of finding conserved gene clusters in different genomes translates to the problem of finding *common intervals* in multiple permutations.

In addition to this bioinformatical application, common intervals also relate to the *consecutive arrangement problem* [2, 7, 8] and to cross-over operators for genetic algorithms solving sequencing problems such as the

traveling salesman problem or the single machine scheduling problem [3, 15, 18].

Recently, Uno and Yagiura [26] presented an optimal $O(n + K)$ time and $O(n)$ space algorithm for finding all $K \leq \binom{n}{2}$ common intervals of two permutations π_1 and π_2 of n elements. We generalized this algorithm to a family $\Pi = (\pi_1, \ldots, \pi_k)$ of $k \geq 2$ permutations in optimal $O(kn+K)$ time and $O(n)$ space [10] by restricting the set of common intervals to a smaller, generating subset. To apply common intervals to the bioinformatical problem of finding conserved clusters of genes in data derived from completely sequenced genomes we further extended the above algorithm to additional types of permutations.

Genomes of higher organisms generally consist of several linear chromosomes while bacterial, archaeal, and mitochondrial DNA is organized in one to several circular pieces. While in the first case the algorithm from [10] might report too many gene clusters if the multiple chromosomes are simply concatenated, in the latter case gene clusters might be missed if the circular pieces are cut at some arbitrary point. We handle this problem by adapting the original algorithm to *multichromosomal permutations* as well as *circular permutations*.

For prokaryotes, it is also known that, in the vast majority of cases, functionally associated genes of a gene cluster lie on the same DNA strand [20, 12]. We take this into account by constructing *signed permutations* where the sign of a gene indicates the strand it lies on. We then determine all common intervals with the additional restriction that within each permutation, the elements of a common interval must have the same sign, while between permutations the sign might vary. This allows us to restrict the set of common intervals to biologically meaningful candidates.

The paper is organized as follows. In Section 2 we formally define common intervals and related terminology. We briefly describe the algorithms of Uno and Yagiura [26] and of Heber and Stoye [10] to find all common intervals of 2 (respectively $k \geq 2$) permutations. Then we present time- and space-optimal algorithms for the problem of finding all common intervals in multichromosomal permutations (Section 3), in signed permutations (Section 4), and in circular permutations (Section 5). In Section 6 we show how the various approaches can be combined without sacrificing the optimal time complexity. Section 7 concludes with few final remarks.

2 Common and Irreducible Intervals

2.1 Basic Definitions

A *permutation* π of (the elements of) the set $N := \{1, 2, \ldots, n\}$ is a re-ordering of the elements of N. We denote by $\pi(i) = j$ that the ith element in this re-ordering is j. For $1 \leq x \leq y \leq n$, we set $[x, y] := \{x, x+1, \ldots, y\}$ and call $\pi([x, y]) := \{\pi(i) \mid i \in [x, y]\}$ an *interval* of π.

Let $\Pi = (\pi_1, \ldots, \pi_k)$ be a family of k permutations of N. Without loss of generality we assume in this section that $\pi_1 = id_n := (1, \ldots, n)$. A subset $c \subseteq N$ is called a *common interval* of Π if and only if there exist $1 \leq l_j < u_j \leq n$ for all $1 \leq j \leq k$ such that

$$c \;=\; \pi_1([l_1, u_1]) \;=\; \pi_2([l_2, u_2]) \;=\; \ldots \;=\; \pi_k([l_k, u_k]).$$

Note that this definition excludes common intervals of size one.

In the following we represent a common interval c either by specifying its elements or by the shorter notation $\pi_j([l_j, u_j])$ for a $j \in \{1, \ldots, n\}$. (For $\pi_j = id_n$ this notation further simplifies to $[l_j, u_j]$.) The set of all common intervals of $\Pi = (\pi_1, \ldots, \pi_k)$ is denoted C_Π.

Example 1. Let $N = \{1, \ldots, 9\}$ and $\Pi = (\pi_1, \pi_2, \pi_3)$ with $\pi_1 = id_9$, $\pi_2 = (3, 2, 1, 9, 7, 8, 6, 5, 4)$, and $\pi_3 = (4, 5, 6, 8, 7, 1, 2, 3, 9)$. With respect to π_1 we have

$$C_\Pi = \{[1, 2], [1, 3], [1, 9], [2, 3], [4, 5], [4, 6], [4, 8], [5, 6], [5, 8], [6, 8], [7, 8]\}.$$

□

In order to keep this paper self-contained, in the remainder of this section we recall the algorithms of Uno and Yagiura [26] and of Heber and Stoye [10] that find all common intervals of 2 (respectively $k \geq 2$) permutations. We will restrict our description to basic ideas and only give details where they are necessary for an understanding of the new algorithms described in Sections 3–6 of this paper.

2.2 Finding All Common Intervals of Two Permutations

Here we consider the problem of finding all common intervals of $k = 2$ permutations $\pi_1 = id_n$ and π_2 of N.

An easy test if an interval $\pi_2([x, y])$, $1 \leq x < y \leq n$, is a common interval of $\Pi = (\pi_1, \pi_2)$ is based on the following functions:

$$\begin{aligned} l(x, y) &:= \min \pi_2([x, y]) \\ u(x, y) &:= \max \pi_2([x, y]) \\ f(x, y) &:= u(x, y) - l(x, y) - (y - x). \end{aligned}$$

Since $f(x,y)$ counts the number of elements in $[l(x,y),u(x,y)] \setminus \pi_2([x,y])$, an interval $\pi_2([x,y])$ is a common interval of Π if and only if $f(x,y)=0$. A simple algorithm to find C_Π is to test for each pair of indices (x,y) with $1 \le x < y \le n$ if $f(x,y)=0$, yielding a naive $O(n^3)$ time or, using running minima and maxima, a slightly more involved $O(n^2)$ time algorithm.

In order to save the time to test $f(x,y)=0$ for some pairs (x,y), Uno and Yagiura [26] introduce the notion of *wasteful* candidates for y.

Definition 1. *For a fixed x, a right interval end $y > x$ is called* wasteful *if it satisfies $f(x',y) > 0$ for all $x' \le x$.*

Based on this notion, Uno and Yagiura give an algorithm called RC (short for *Reduce Candidate*) that has as its essential part a data structure Y consisting of a doubly-linked list *ylist* for the indices of non-wasteful right interval end candidates and, storing intervals of *ylist*, two further doubly-linked lists *llist* and *ulist* that implement the functions l and u in order to compute f efficiently. An outline of Algorithm RC is shown in Algorithm 1 where $L.succ(e)$ denotes the successor of element e in a doubly linked list L.

Algorithm 1 (Reduce Candidate, RC)

Input: A family $\Pi = (\pi_1 = id_n, \pi_2)$ of two permutations of $N = \{1,\dots,n\}$.
Output: The set of all common intervals C_Π.

```
1: initialize Y
2: for x = n − 1, ..., 1 do
3:    update Y    // trim ylist, update llist and ulist
4:    y ← x
5:    while (y ← ylist.succ(y)) defined and f(x, y) = 0 do
6:       output [l(x, y), u(x, y)]
7:    end while
8: end for
```

After initializing the lists of Y, a counter x (corresponding to the currently investigated left interval end) runs from $n-1$ down to 1. In each iteration step, during the update of Y, *ylist* is trimmed such that afterwards the function $f(x,y)$ is monotonically increasing for the elements y remaining in *ylist*. In lines 5–7, this allows us to efficiently find all common intervals with left end x by evaluating $f(x,y)$ running left-to-right through the elements $y > x$ of *ylist* until an index y is encountered with $f(x,y) > 0$ when the reporting procedure stops.

For details of the data structure Y and the update procedure in line 3, see [26, 10]. The analysis shows that the update of data structure Y in line 3 can be performed in amortized $O(1)$ time, such that the complete algorithm takes $O(n+K)$ time to find the K common intervals of π_1 and π_2.

2.3 Irreducible Intervals

Before we show how to generalize Algorithm RC to find all common intervals of $k \geq 2$ permutations, we first present a useful generating subset of the set of common intervals, the set of *irreducible intervals* [10] and report a few of their properties.

We say that two common intervals $c_1, c_2 \in C_\Pi$ have a *non-trivial overlap* if $c_1 \cap c_2 \neq \emptyset$ and neither includes the other. A list $p = (c_1, \ldots, c_{\ell(p)})$ of common intervals $c_1, \ldots, c_{\ell(p)} \in C_\Pi$ is a *chain* (of length $\ell(p)$) if every two successive intervals in p have a non-trivial overlap. A chain of length one is called a *trivial chain*, all other chains are called *non-trivial chains*. A chain that can not be extended to its left or right is a *maximal chain*. It is easy to see that for a chain p of common intervals, the interval $\tau(p) := \bigcup_{c' \in p} c'$ is a common interval as well. We say that p *generates* $\tau(p)$.

Definition 2. *A common interval c is called* reducible *if there is a non-trivial chain that generates c, otherwise it is called* irreducible.

This definition partitions the set of common intervals C_Π into the set of reducible intervals and the set of irreducible intervals, denoted I_Π. Obviously, $1 \leq |I_\Pi| \leq |C_\Pi| \leq \binom{n}{2}$.

Example 1 (cont'd). For $\Pi = (\pi_1, \pi_2, \pi_3)$ as above, the irreducible intervals (with respect to $\pi_1 = id_9$) are

$$I_\Pi \;=\; \{[1,2], [1,9], [2,3], [4,5], [5,6], [6,8], [7,8]\}.$$

The reducible intervals are generated as follows:

$$\begin{aligned} [1,3] &= [1,2] \cup [2,3], \\ [4,6] &= [4,5] \cup [5,6], \\ [4,8] &= [4,5] \cup [5,6] \cup [6,8], \\ [5,8] &= [5,6] \cup [6,8]. \end{aligned}$$

□

We cite the following two results from [10] (without proofs) which indicate the great value of the concept of irreducible intervals.

Lemma 1. *Given a family* $\Pi = (\pi_1, \ldots, \pi_k)$ *of permutations of* $N = \{1, 2, \ldots, n\}$*, the set of irreducible intervals* I_Π *allows us to reconstruct the set of all common intervals* C_Π *in optimal* $O(|C_\Pi|)$ *time.* □

Lemma 2. *Given a family* $\Pi = (\pi_1, \ldots, \pi_k)$ *of permutations of* $N = \{1, 2, \ldots, n\}$*, we have* $1 \leq |I_\Pi| \leq n - 1$. □

2.4 Finding All Common Intervals of k Permutations

Now we can describe the algorithm from [10] that finds all K common intervals of a family of $k \geq 2$ permutations of N in $O(kn + K)$ time.

For $1 \leq i \leq k$, set $\Pi_i := (\pi_1, \ldots, \pi_i)$. Starting with $I_{\Pi_1} = \{[j, j+1] \mid 1 \leq j \leq n-1\}$, the algorithm successively computes I_{Π_i} from $I_{\Pi_{i-1}}$ for $i = 2, \ldots, k$ (see Algorithm 2). The algorithm employs a mapping

$$\varphi_i : I_{\Pi_{i-1}} \to I_{\Pi_i}$$

that maps each element $c \in I_{\Pi_{i-1}}$ to the smallest common interval $c' \in C_{\Pi_i}$ that contains c. It is shown in [10] that this mapping exists and is surjective, i.e., $\varphi_i(I_{\Pi_{i-1}}) := \{\varphi_i(c) \mid c \in I_{\Pi_{i-1}}\} = I_{\Pi_i}$. Furthermore, it is

Algorithm 2 (Finding all Common Intervals of k Permutations)

Input: A family $\Pi = (\pi_1 = id_n, \pi_2, \ldots, \pi_k)$ of k permutations of $N = \{1, \ldots, n\}$.
Output: The set of all common intervals C_Π.

```
1: I_Π1 ← ([1,2],[2,3],...,[n−1,n])
2: for i = 2,...,k do
3:     I_Πi ← {φ_i(c) | c ∈ I_Πi−1}    // (see Algorithm 3)
4: end for
5: generate C_Π from I_Π = I_Πk using Lemma 1
6: output C_Π
```

shown that $\varphi_i(I_{\Pi_{i-1}})$ can be efficiently computed by a modified version of Algorithm RC where the data structure Y is supplemented by a data structure S that is derived from $I_{\Pi_{i-1}}$. S consists of several doubly-linked *clist*s containing intervals of *ylist*, one for each maximal chain of the intervals in $I_{\Pi_{i-1}}$.

Using π_1 and π_i, as in Algorithm RC, the *ylist* of Y allows for a given x to access all non-wasteful right interval end candidates y of $C_{(\pi_1, \pi_i)}$.

The aim of S is to further reduce these candidates to only those indices y for which simultaneously $[x, y] \in C_{\Pi_{i-1}}$ (ensuring $[x, y] \in C_{\Pi_i}$) and $[x, y]$ contains an interval $c \in I_{\Pi_{i-1}}$ that is not contained in any smaller interval from C_{Π_i}. Together this ensures that exactly the irreducible intervals $[x, y] \in \varphi_i(I_{\Pi_{i-1}})$ are reported.

An outline of the modified version of Algorithm RC is shown in Algorithm 3. Essentially, S keeps a list of *active* intervals, i.e., intervals from

Algorithm 3 (Extended Algorithm RC)

Input: A family $\Pi = (\pi_1 = id_n, \pi_i)$ of two permutations of $N = \{1, \ldots, n\}$; a set of irreducible intervals $I_{\Pi_{i-1}}$.
Output: The set of irreducible intervals I_{Π_i}.

```
1: initialize Y and S
2: for x = n − 1, ..., 1 do
3:    update Y and S    // trim ylist, update l/ulist; activate elements of the clists
4:    while ([x', y] ← S.first_active_interval(x)) defined and f(x, y) = 0 do
5:       output [l(x, y), u(x, y)]
6:       deactivate [x', y]
7:    end while
8: end for
```

$I_{\Pi_{i-1}}$ for which the image under mapping φ_i has not yet been determined. In the reporting loop of lines 4–7, rather than testing if $f(x, y) = 0$ running from left to right through all indices $y > x$ of *ylist*, only right ends of active intervals are tested. Therefore, function $S.\mathit{first\_active\_interval}(x)$ returns the active intervals in left-to-right order with respect to their right end y. If an active interval $[x', y]$ gives rise to a common interval, i.e., if $f(x, y) = 0$, then an element of $\varphi_i(I_{\Pi_{i-1}})$ is encountered and the active interval is deactivated. Similar to Algorithm RC, reporting stops whenever the first active interval with right end y is encountered such that $f(x, y) > 0$.

Again, for details of the data structure and the update procedure in line 3 we refer to the original description [10]. There it is also shown that updating the data structure S takes amortized $O(1)$ time. Hence, due to the reduced output size (see Lemma 2), the Extended Algorithm RC takes only $O(n)$ time. Together with Lemma 1 this implies the overall time complexity $O(kn + K)$ for Algorithm 2. The additional space usage is $O(n)$.

3 Common Intervals of Multichromosomal Permutations

In view of biological reality, in the following we consider variants of the common intervals problem that have to be addressed when dealing with real genomic data. Our first variant that we consider is the scenario where the genome consists of multiple chromosomes.

As above, let $N := \{1, 2, \ldots, n\}$ represent a set of n genes. A *chromosome* c of N is defined as a linearly ordered subset of N and will be represented as a linear list. A *multichromosomal permutation* π of N is defined as a set of chromosomes, containing each element of N exactly once, i.e.

$$\pi = \{c_1, \ldots, c_l\} \quad \text{with} \quad N = \dot{\bigcup}_{1 \le i \le l} c_i.$$

Given a family $\Pi = (\pi_1, \ldots, \pi_k)$ of k multichromosomal permutations of N, a subset $s \subseteq N$ is called a *common interval* of Π if and only if for each multichromosomal permutation π_i, $i = 1, \ldots, k$, there exists a chromosome with s as an interval.

Example 2. Let $N = \{1, \ldots, 6\}$ and $\Pi = (\pi_1, \pi_2, \pi_3)$ with $\pi_1 = \{(1, 2, 3), (4, 5, 6)\}$, $\pi_2 = \{(1, 5, 6, 4), (3, 2)\}$, and $\pi_3 = \{(1, 6, 4, 5), (3), (2)\}$. Here chromosome ends are indicated by parentheses. The only common interval is $\{4, 5, 6\}$. □

A modification of Algorithm 2 can be used for finding all common intervals of k multichromosomal permutations. We start by arranging the chromosomes of each multichromosomal permutation in arbitrary order. This way we obtain a family $\Pi' = (\pi'_1, \pi'_2, \ldots, \pi'_k)$ of k (standard) permutations π'_i, $i = 1, \ldots, k$. Without loss of generality we assume that $\pi'_1 = id_n$. Now, as above, set $\Pi'_i := (\pi'_1, \pi'_2, \ldots, \pi'_i)$. Then, starting with

$$I_{\Pi'_1} := \{[j, j+1] \mid j, j+1 \text{ on the same chromosome in } \pi_1 \text{ and } 1 \le j < n\},$$

the algorithm successively computes $I_{\Pi'_i}$ from $I_{\Pi'_{i-1}}$ for $i = 2, \ldots, k$ using a modification of Algorithm 2, where in the extended Algorithm RC (Algorithm 3) the reporting procedure is not only stopped whenever $f(x, y) > 0$, but also as soon as the genes at indices x and y belong to different chromosomes of π_i.

By the definition of $I_{\Pi'_1}$, this algorithm will never place two genes from different chromosomes in π_1 together in a common interval. Moreover, by the modification of Algorithm 3, two genes from different chromosomes of the other genomes $\pi_2, \ldots, \pi_k$ will never be placed together in a common interval. Nevertheless, the location of common intervals that lie on the

same chromosome in all genomes is not affected by the modification of the algorithm. Since the additional test if x and y belong to the same chromosome is a constant time operation, and the output can only be smaller than that of the original Algorithm 3, the new algorithm also takes $O(n)$ time to generate $I_{\Pi'_i}$ from $I_{\Pi'_{i-1}}$. The outer loop (Algorithm 2) and the final generation of the common intervals from the irreducible intervals (Lemma 1) are unchanged, so that we have the following

Theorem 1. *Given k multichromosomal permutations of $N = \{1, \ldots, n\}$, all K common intervals can be found in optimal $O(kn + K)$ time using $O(n)$ additional space.* □

4 Common Intervals of Signed Permutations

In this section we consider the problem of finding all common intervals in a family of signed permutations. It is common practice when considering genome rearrangement problems, to denote the direction of a gene in the genome by a plus (+) or minus (−) sign depending on the DNA strand it is located on [21]. In the context of sorting signed permutations by reversals [1, 9, 13, 14], the sign of a gene tells the direction of the gene in the final (sorted) permutation and changes with each reversal. In our context, it has been observed that for prokaryotes, functionally coupled genes, e.g. in operons, virtually always lie on the same DNA strand [20, 12]. Hence, when given signed permutations, we require that the sign does not change within an interval. Between the different permutations, the sign of the intervals might vary, though. This restricts the (original) set of all common intervals to the biologically more meaningful candidates.

Example 3. Let $N = \{1, \ldots, 6\}$ and $\Pi = (\pi_1, \pi_2, \pi_3)$ with $\pi_1 = (+1, +2, +3, +4, +5, +6)$, $\pi_2 = (-3, -1, -2, +5, +4, +6)$, and $\pi_3 = (-4, +5, +6, -2, -3, -1)$. With respect to π_1 the interval $[1, 3]$ is a common interval, but $[4, 5]$ and $[4, 6]$ are not. □

Obviously, the number of common intervals in signed permutations can be considerably smaller than the number of common intervals in unsigned permutations. Hence, applying Algorithm 2 followed by a filtering step will not yield our desired time-optimal result.

However, the problem can be solved easily by applying the algorithm for multichromosomal permutations from the previous section. Since a common interval in signed permutations can never contain two genes with different sign, we break the signed permutations into pieces ("chromosomes") wherever the sign changes. This is clearly possible in linear

time. Then we apply the algorithm from the previous section to the obtained family of multichromosomal permutations, the result being exactly the common intervals of the original signed permutations. Hence, we have the following

Theorem 2. *Given k signed permutations of $N = \{1, \ldots, n\}$, all K common intervals can be found in optimal $O(kn + K)$ time using $O(n)$ additional space.* □

5 Common Intervals of Circular Permutations

As discussed in the Introduction, much of the DNA in nature is circular. Consequently, by representing genomes as (possibly multichromosomal) *linear* permutations of genes and then looking for common gene clusters, one might miss clusters that span across the (mostly arbitrary) dissection point where the circular genome is linearized.

In this section we consider an arrangement of the set of genes $N = \{1, 2, \ldots, n\}$ along a circle and call this a *circular permutation*. Given a family $\Pi = (\pi_1, \ldots, \pi_k)$ of k circular permutations of N, a (sub)set $c \subseteq N$ of genes is called a *common interval* if and only if the elements of c occur uninterruptedly in each circular permutation.

Example 4. Let $N = \{1, \ldots, 6\}$ and $\Pi = (\pi_1, \pi_2, \pi_3)$ with $\pi_1 = (1, 2, 3, 4, 5, 6)$, $\pi_2 = (2, 4, 5, 6, 1, 3)$, and $\pi_3 = (6, 4, 1, 3, 2, 5)$. Apart from the trivial intervals (N, the singletons, and N minus each singleton), the common intervals of Π are $\{1, 2, 3\}$, $\{1, 2, 3, 4\}$, $\{1, 4, 5, 6\}$, $\{2, 3\}$, $\{4, 5, 6\}$, $\{5, 6\}$. □

In the following we will show how to find all K common intervals in a family of circular permutations in optimal $O(kn + K)$ time. Again, this can be done by an easy modification of the original algorithm from Section 2, in combination with the following observation.

Lemma 3. *Let c be a common interval of a family Π of circular permutations of N. Then its complement $\bar{c} := N \setminus c$ is also a common interval of Π.*

Proof. This follows immediately from the definition of common intervals of circular permutations. □

Note that Lemma 3 does not hold for irreducible intervals.

Algorithm 4 (Finding all Common Intervals of k Circular Permutations)

Input: A family $\Pi = (\pi_1 = id_n, \pi_2, \ldots, \pi_k)$ of k circular permutations of $N = \{1, \ldots, n\}$.
Output: The set of all common intervals C_Π.
1: $I^*_{\Pi_1} \leftarrow (\{1,2\}, \{2,3\}, \ldots, \{n-1, n\}, \{n, 1\})$
2: **for** $i = 2, \ldots, k$ **do**
3: $\quad I^*_{\Pi_i} \leftarrow \{\varphi^*_i(c) \mid c \in I^*_{\Pi_{i-1}}\}$ // (see text)
4: **end for**
5: generate C^*_Π from $I^*_\Pi = I^*_{\Pi_k}$ using Lemma 1
6: $\overline{C}^*_\Pi \leftarrow \{\bar{c} \mid c \in C^*_\Pi\}$
7: output $C^*_\Pi \cup \overline{C}^*_\Pi$

The general idea is now to first find only the common intervals of size $\leq \lfloor \frac{n}{2} \rfloor$, and then find the remaining common intervals by complementing these. The procedure is outlined in Algorithm 4. The main difference to Algorithm 2 is that function φ_i is replaced by a variant, denoted φ^*_i, that works on circular permutations and only generates irreducible intervals of size $\leq \lfloor \frac{n}{2} \rfloor$. This function is implemented by multiple calls to the original function φ_i. The two circular permutations π_1 and π_i are therefore linearized in two different ways each, namely by once cutting them between positions n and 1, and once cutting between positions $\lfloor \frac{n}{2} \rfloor$ and $\lfloor \frac{n}{2} \rfloor + 1$. Then φ_i is applied to each of the four resulting pairs of linearized permutations. For convenience, the output of common intervals of length $> \lfloor \frac{n}{2} \rfloor$ is suppressed. Finally, the resulting intervals of the four runs of φ_i are merged, sorted according their start and end positions using counting sort, and duplicates are removed. Clearly, φ^*_i generates all irreducible intervals of size $\leq \lfloor \frac{n}{2} \rfloor$ in $O(n)$ time. Hence, we have the following

Theorem 3. *Given k circular permutations of $N = \{1, \ldots, n\}$, all K common intervals can be found in optimal $O(kn + K)$ time using $O(n)$ additional space.* □

6 Combination of the Algorithms

In this section we show how to handle arbitrary combinations of multichromosomal, signed, and circular permutations.

Combining multichromosomal and signed permutations is straightforward, but it is not obvious how to handle combinations which involve circular chromosomes without loosing the optimal running time. Circular chromosomes of different genomes might now have incompatible gene contents and Lemma 3 no longer holds as the following example shows.

Example 5. Let $N = \{1, \ldots, 8\}$ and $\Pi = (\pi_1, \pi_2)$ with $\pi_1 = \{(1, 2, 3, 4), (5, 6, 7, 8)\}$ and $\pi_2 = \{(1, 3, 5, 6, 7), (2, 4, 8)\}$ where all chromosomes are circular. While $c = \{5, 6\}$ is a common interval, its complement $N \setminus c = \{1, 2, 3, 4, 7, 8\}$ is not. □

We overcome these problems by a preprocessing step where we include artificial *breakpoints* into the genomes. The breakpoints do not affect common intervals but refine the permutations so that they can be handled by our algorithms. The preprocessing is performed as follows.

We compare permutation π_1 successively to each of the other permutations π_i, $2 \leq i \leq k$ and test for each pair of neighboring genes in π_1 (i.e. for each chromosome $c = (\pi_1(l), \pi_1(l+1), \ldots, \pi_1(r))$ the pairs $\{\pi_1(j), \pi_1(j+1)\}$ for $l \leq j \leq r-1$, plus the pair $\{\pi_1(l), \pi_1(r)\}$ for circular c) if they lie on the same chromosome in π_i or not. If not, they can not be elements of the same common interval and we introduce a new artificial breakpoint between the two genes in π_1. Then we do the same comparison in the opposite direction, i.e. we introduce breakpoints between neighboring genes of π_i, $2 \leq i \leq k$ whenever they do not lie on the same chromosome of π_1. At the first time a breakpoint is inserted in a circular chromosome, the chromosome is linearized by cutting at the breakpoint and replacing it in the genome by the appropriately circularly shifted linear chromosome. Breakpoints in a linear chromosome dissect the chromosome. This preprocessing can be performed in $O(kn)$ time.

After the preprocessing, the genes that do not occur in any circular chromosome can be handled by the algorithm for multichromosomal permutations (Section 3) in a straightforward way. The genes that occur in at least one circular chromosome are partitioned into sets of genes which correspond to a single circular chromosome. This partition is well defined, since the set of genes of each remaining circular chromosome corresponds, in the other genomes, either to one circular or to one or several linear chromosomes. Each element of this partition is now treated separately. We start by restricting all genomes to the selected gene set. If each of the restricted genomes is circular we can apply the algorithm for circular permutations (Section 5) directly. Otherwise we choose a restricted genome that consists of one or several linear chromosomes and arrange the chromosomes in an arbitrary order. Denote l (r) the first (last) gene in this order. We proceed as in the multichromosomal case (Section 3) except we encounter a circular genome π_c. If l and r are neighboring genes in π_c we linearize π_c by cutting between them and proceed as for a linear genome. Otherwise, similar to the case of circular permutations (Section 5), we copy π_c four times and linearize the copies by cutting one copy on the left

of l, one copy on the right of l, one copy on the left of r, and one copy on the right of r. For each of these genomes we compute all irreducible intervals. The resulting intervals are merged, sorted according their start and end positions using counting sort, and duplicates are removed. This procedure guarantees that we determine all irreducible intervals except for those, which contain l and r simultaneously. But due to our choice of l and r there is at most one such interval, the trivial one, which contains all genes. We test this interval separately.

Since the above described preprocessing and the modifications of the algorithms for multichromosomal and circular permutations do not affect the optimal asymptotic running time, we have

Theorem 4. *Given k multichromosomal, signed, circular or linear (or mixed) permutations of $N = \{1, \ldots, n\}$, all K common intervals can be found in optimal $O(kn + K)$ time using $O(n)$ additional space.* □

7 Conclusion

In this paper we have presented time and space optimal algorithms for variants of the common intervals problem for k permutations. The variants we considered, multichromosomal permutations, signed permutations, circular permutations, and their combinations, were motivated by the requirements imposed by real data we were confronted with in our experiments. While in preliminary testing we have applied our algorithms to bacterial genomes, it is obvious that in a realistic setting, one should further relax the problem definition. In particular, one should allow for missing or additional genes in a common interval while imposing a penalty whenever this occurs. Such relaxations seem to make the problem much harder, though.

Acknowledgments

We thank Richard Desper and Zufar Mulyukov for carefully reading this manuscript as well as Pavel Pevzner for a helpful discussion on the ideas contained in the manuscript.

References

1. V. Bafna and P. Pevzner. Genome rearrangements and sorting by reversals. *SIAM J. Computing*, 25(2):272–289, 1996.
2. K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs and graph planarity using *PQ*-tree algorithms. *J. Comput. Syst. Sci.*, 13(3):335–379, 1976.
3. R. M. Brady. Optimization strategies gleaned from biological evolution. *Nature*, 317:804–806, 1985.
4. T. Dandekar, B. Snel, M. Huynen, and P. Bork. Conservation of gene order: A fingerprint of proteins that physically interact. *Trends Biochem. Sci.*, 23(9):324–328, 1998.
5. J. A. Eisen. Horizontal gene transfer among microbial genomes: new insights from complete genome analysis. *Curr. Opin. Genet. Dev.*, 10(6):606–611, 2000.
6. W. Fujibuchi, H. Ogata, H. Matsuda, and M. Kanehisa. Automatic detection of conserved gene clusters in multiple genomes by graph comparison and P-quasi grouping. *Nucleic Acids Res.*, 28(20):4029–4036, 2000.
7. D. Fulkerson and O. Gross. Incidence matrices with the consecutive 1s property. *Bull. Am. Math. Soc.*, 70:681–684, 1964.
8. M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.
9. S. Hannenhalli and P. A. Pevzner. Transforming cabbage into turnip: Polynomial algorithm for sorting signed permutations by reversals. *J. ACM*, 46(1):1–27, 1999.
10. S. Heber and J. Stoye. Finding all common intervals of k permutations. In *Proceedings of the 12th Annual Symposium on Combinatorial Pattern Matching, CPM 2001*, volume 2089 of *Lecture Notes in Computer Science*, pages 207–219. Springer Verlag, 2001. To appear.
11. M. A. Huynen and P. Bork. Measuring genome evolution. *Proc. Natl. Acad. Sci. USA*, 95(11):5849–5856, 1998.
12. M. A. Huynen, B. Snel, and P. Bork. Inversions and the dynamics of eukaryotic gene order. *Trends Genet.*, 17(6):304–306, 2001.
13. H. Kaplan, R. Shamir, and R. E. Tarjan. A faster and simpler algorithm for sorting signed permutations by reversals. *SIAM J. Computing*, 29(3):880–892, 1999.
14. J. D. Kececioglu and D. Sankoff. Efficient bounds for oriented chromosome inversion distance. In M. Crochemore and D. Gusfield, editors, *Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching, CPM 94*, volume 807 of *Lecture Notes in Computer Science*, pages 307–325. Springer Verlag, 1994.
15. S. Kobayashi, I. Ono, and M. Yamamura. An efficient genetic algorithm for job shop scheduling problems. In *Proc. of the 6th International Conference on Genetic Algorithms*, pages 506–511. Morgan Kaufmann, 1995.
16. W. C. I. Lathe, B. Snel, and P. Bork. Gene context conservation of a higher order than operons. *Trends Biochem. Sci.*, 25(10):474–479, 2000.
17. E. M. Marcotte, M. Pellegrini, H. L. Ng, D. W. Rice, T. O. Yeates, and D. Eisenberg. Detecting protein function and protein-protein interactions from genome sequences. *Science*, 285:751–753, 1999.
18. H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer. Evolution algorithms in combinatorial optimization. *Parallel Comput.*, 7:65–85, 1988.
19. A. R. Mushegian and E. V. Koonin. Gene order is not conserved in bacterial evolution. *Trends Genet.*, 12(8):289–290, 1996.

20. R. Overbeek, M. Fonstein, M. D'Souza, G. D. Pusch, and N. Maltsev. The use of gene clusters to infer functional coupling. *Proc. Natl. Acad. Sci. USA*, 96(6):2896–2901, 1999.
21. P. A. Pevzner. *Computational Molecular Biology: An Algorithmic Approach*. MIT Press, Cambridge, MA, 2000.
22. B. Snel, G. Lehmann, P. Bork, and M. A. Huynen. STRING: A web-server to retrieve and display the repeatedly occurring neigbourhood of a gene. *Nucleic Acids Res.*, 28(18):3443–3444, 2000.
23. J. Tamames, G. Casari, C. Ouzounis, and A. Valencia. Conserved clusters of functionally related genes in two bacterial genomes. *J. Mol. Evol.*, 44(1):66–73, 1997.
24. J. Tamames, M. Gonzalez-Moreno, J. Mingorance, A. Valencia, and M. Vicente. Bringing gene order into bacterial shape. *Trends Genet.*, 17(3):124–126, 2001.
25. R. L. Tatusov, A. R. Mushegian, P. Bork, N. Brown, W. S. Hayes, M. Borodovsky, K. E. Rudd, and E. V. Koonin. Metabolism and evolution of *Haemophilus influenzae* deduced from a whole-genome comparison with *Escherichia coli*. *Curr. Biol.*, 6:279–291, 1996.
26. T. Uno and M. Yagiura. Fast algorithms to enumerate all common intervals of two permutations. *Algorithmica*, 26(2):290–309, 2000.