Common Intervals of Two Sequences

Gilles Didier

Laboratoire Genome et Informatique - CNRS UMR 8116
Tour Evry2 - 523, place des Terrasses de l'Agora
91034 EVRY CEDEX
didier@genopole.cnrs.fr

Abstract. Looking for the subsets of genes appearing consecutively in two or more genomes is an useful approach to identify clusters of genes functionally associated. A possible formalization of this problem is to modelize the order in which the genes appear in all the considered genomes as permutations of their order in the first genome and find k-tuples of contiguous subsets of these permutations consisting of the same elements: the common intervals. A drawback of this approach is that it doesn't allow to take into account paralog genes and genomic internal duplications (each element occurs only once in a permutation). To do it we need to modelize the order of genes by sequences which are not necessary permutations.
In this work, we study some properties of common intervals between two general sequences. We bound the maximum number of common intervals between two sequences of length n by n^2 and present an $O(n^2 \log(n))$ time complexity algorithm to enumerate their whole set of common intervals. This complexity does not depend on the size of the alphabets of the sequences.

1 Introduction

The comparison of the sequences of genes of two or more organisms, in particular finding locally conserved subsets of genes in several genomes, can be used to predict protein function or just to determine cluster of genes functionally associated [3,6,7]. A way to find such clusters is to represent these sequences of genes by permutations and to find the consecutive elements conserved in these permutations, which are called common intervals [5]. A limitation of this type of modelization is that an element of a permutation can occur only once within it. Consequently paralog genes or internal duplication in a genome cannot be modelized by permutation. This yields to define and study a notion of common intervals of general sequences.

But firstly, let us recall that the question of finding all common intervals of two permutations of n elements was studied and solved by algorithms having $O(n^2)$ or $O(n + K)$ time complexity, where K is the effective number of common intervals [8]. This number K is less or equal than $\binom{n}{2}$ in the case of two permutations. So in the worst case, any algorithm solving this question will

G. Benson and R. Page (Eds.): WABI 2003, LNBI 2812, pp. 17–24, 2003.
© Springer-Verlag Berlin Heidelberg 2003

run in $O(n^2)$ time. Enumerating the common intervals of a given number (≥ 2) of permutations was solved in [4], and some more theoretical applications and developments of this type of algorithms in [2].

We are interested here in a similar problem over two general sequences. The main difference with permutations is that elements can occur more than a single time. Because of this, techniques used in the permutations case cannot be directly transposed. We first give a bound for the number of common intervals between two sequences and an algorithm to enumerate them. The algorithmic complexity is $O(n^2 \log(n))$ in the worst case, where n is the length of the sequences. In the case of two permutations, this algorithm runs in $O(n^2)$ time.

A recent algorithm, developed by Amir *et al* in [1], solves the same question in $O(n|\Sigma| \log(n) \log(|\Sigma|))$, where $|\Sigma|$ is the size of the alphabet of the sequences, by using an efficient coding of the (sub-)alphabets (called *fingerprints*) of their subsequences. Depending on the size of the alphabet relatively to the length of the sequences, their algorithm or the present one is the fastest.

2 Definitions and Notations

Let s be a finite sequence over a certain set of elements called symbols, we note $|s|$ its length and index it from 1 to $|s|$: $s = s_1 s_2 \dots s_{|s|}$. The alphabet of s, noted $\mathcal{A}_s$, is the set of symbols appearing in this sequence: $\mathcal{A}_s = \{s_l | 1 \leq l \leq |s|\}$. To simplify the statements of definitions and proofs, we add a special symbol δ, never occurring in the considered sequences, at their beginnings and their ends: *i.e.* we fix $s_0 = s_{|s|+1} = \delta$ for each sequence s. Nevertheless, in the following, 0 and $|s| + 1$ won't be considered as positions of s and $\delta \notin \mathcal{A}_s$.

For a pair (i, j) of integers with $1 \leq i \leq j \leq |s|$ we note $[i, j]$ the set $\{i, i+1, \dots, j\}$. The corresponding *subsequence* of s, noted $s_{[i,j]}$ is the sequence $s_i s_{i+1} \dots s_j$. A subsequence $s_{[i,j]}$ is said *alphabetically deliminated* if $s_{i-1} \notin \mathcal{A}_{s_{[i,j]}}$ and $s_{j+1} \notin \mathcal{A}_{s_{[i,j]}}$.

Let be two sequences s and s', a pair $([i, j], [i', j'])$ is a *common interval* of s and s' if it satisfies:

1. $s_{[i,j]}$ and $s'_{[i',j']}$ are alphabetically deliminated ;
2. $\mathcal{A}_{s_{[i,j]}} = \mathcal{A}_{s'_{[i',j']}}$.

We note $\mathcal{C}(s, s')$ the set of common intervals of s and s'. Considering only alphabetically deliminated subsequences involves a sort of maximality property over intervals. More precisely, if $([i, j], [i', j'])$ and $([k, l], [i', j'])$ (resp. and $([i, j], [k', l'])$) belong to $\mathcal{C}(s, s')$ then the intervals $[k, l]$ and $[i, j]$ (resp. $[k', l']$ and $[i', j']$) are disjoint or equal.

For the following sequences:

$$s = \texttt{a b c b d a d b c a}$$
$$s' = \texttt{b a c a d b a b d c}$$

The set of common intervals is:

$\mathcal{C}(s, s') = \{([1,1],[2,2]),([1,1],[4,4]),([1,1],[7,7]),([1,2],[1,2]),([1,2],[6,8]),([1,4],[1,4]),([1,10],[1,10]),...\}$

3 Number of Common Intervals

By definition, the alphabetically deliminated subsequences starting at a position i of s cannot end after the first occurrence of s_{i-1} following i. For each position i of s, let us denote by $\mathbf{o}^i$ the sequence of the set of symbols occurring between i (included) and the first occurrence of s_{i-1} following i (excluded), ordered according to their first occurrences. The main interest of this sequence is that the alphabets of alphabetically deliminated subsequences starting at i are the sets $\{\mathbf{o}^i_1, \ldots, \mathbf{o}^i_m\}$ for all integer $m \in \{1, \ldots, |\mathbf{o}^i|\}$.

We note $\mathbf{p}_i$ the application which associates to each $m \in \{1, \ldots, |\mathbf{o}^i| - 1\}$, the position preceding the first occurrence of $\mathbf{o}^i_{k+1}$ following i and to $|\mathbf{o}^i|$ the position preceding the first occurrence of s_{i-1} following i.

Remark 1 *The set of alphabetically deliminated subsequences of s starting at i is $\{[i, \mathbf{p}_i(m)] \,|m \in \{1, \ldots, |\boldsymbol{o}^i|\}\}$ and we have $\mathcal{A}_{s_{[i,\mathbf{p}_i(m)]}} = \{\boldsymbol{o}^i_1, \ldots, \boldsymbol{o}^i_m\}$ for all integer $m \in \{1, \ldots, |\boldsymbol{o}^i|\}$.*

We associate to each symbol $x \in \mathcal{A}_{s'}$ the integer called *i-rank*, noted $\mathbf{r}_i(x)$ and defined as the position of x in $\mathbf{o}^i$ if x occurs in it and as $+\infty$ if not. We fix always $\mathbf{r}_i(\delta)$ to $+\infty$. The restriction of the application $\mathbf{r}_i$ to the set of symbols with finite images is one to one.

$\mathbf{o}^1$ = a b c d
$\mathbf{o}^2$ = b c d
$\mathbf{o}^3$ = c
$\mathbf{o}^4$ = b d a
...

	1	2	3	4
$\mathbf{p}_1$	1	2	4	10
$\mathbf{p}_2$	2	4	5	–
$\mathbf{p}_3$	3	–	–	–
$\mathbf{p}_4$	4	5	8	–
...	...	...	...	...

	a	b	c	d
$\mathbf{r}_1$	1	2	3	4
$\mathbf{r}_2$	$+\infty$	1	2	3
$\mathbf{r}_3$	$+\infty$	$+\infty$	1	$+\infty$
$\mathbf{r}_4$	3	1	$+\infty$	2
...	...	...	...	...

Fig. 1. Construction of $\mathbf{o}^i$, $\mathbf{p}_i$ and $\mathbf{r}_i$ of the sequence s in the preceding example.

An interval $[a, b]$ is said to be *i-complete* if $\{\mathbf{r}_i(s'_l) | l \in [a, b]\} = \{1, \ldots, m\}$, where $m = \max\{\mathbf{r}_i(s'_l) | l \in [a, b]\}$. If $[a, b]$ is i-complete then $\mathcal{A}_{s'_{[a,b]}} = \mathcal{A}_{s_{[i,\mathbf{p}_i(m)]}}$. So an interval i-complete and alphabetically deliminated of s' is common with an interval of s starting at i.

For each position k of s', we note $\mathbf{I}_i(k)$ the interval $[a, b]$ containing k such that the subsequence $s'_{[a,b]}$ of s' verifies:

1. $\mathbf{r}_i(s'_l) \leq \mathbf{r}_i(s'_k)$ for all $l \in [a, b]$;
2. $\mathbf{r}_i(s'_{a-1}) > \mathbf{r}_i(s'_k)$ and $\mathbf{r}_i(s'_{b+1}) > \mathbf{r}_i(s'_k)$.

The collection of intervals $\mathbf{I}_i$ is hierarchically-nested.

Lemma 1 *If $\mathbf{I}_i(k)$ and $\mathbf{I}_i(k')$ are two intervals of $\mathbf{I}_i$, then $\mathbf{I}_i(k) \cap \mathbf{I}_i(k') \in \{\emptyset, \mathbf{I}_i(k), \mathbf{I}_i(k')\}$. More precisely, if $\mathbf{I}_i(k) \cap \mathbf{I}_i(k') \neq \emptyset$ we have:*

1. $\mathbf{I}_i(k) \subset \mathbf{I}_i(k')$ *if and only if* $\boldsymbol{r}_i(s'_k) \leq \boldsymbol{r}_i(s'_{k'})$;
2. $\mathbf{I}_i(k) \supset \mathbf{I}_i(k')$ *if and only if* $\boldsymbol{r}_i(s'_k) \geq \boldsymbol{r}_i(s'_{k'})$;
3. $\mathbf{I}_i(k) = \mathbf{I}_i(k')$ *if and only if* $\boldsymbol{r}_i(s'_k) = \boldsymbol{r}_i(s'_{k'})$.

Proof. We just prove the assertion 1, the other ones being direct consequences. By definition, the maximum i-rank over $\mathbf{I}_i(k)$ (resp.$\mathbf{I}_i(k')$) is reached in position k (resp. k').If $\mathbf{I}_i(k) \supset \mathbf{I}_i(k')$ then $\mathbf{r}_i(s'_k) \geq \mathbf{r}_i(s'_{k'})$.

Let $\mathbf{I}_i(k) = [a, b]$ and $\mathbf{I}_i(k') = [a', b']$ be two intervals of $\mathbf{I}_i$ such that $\mathbf{I}_i(k) \cap \mathbf{I}_i(k') \neq \emptyset$. Assume that $\max\{s'_l | l \in \mathbf{I}_i(k)\} = \mathbf{r}_i(s'_k) \leq \mathbf{r}_i(s'_{k'}) = \max\{s'_l | l \in \mathbf{I}_i(k')\}$. As $\mathbf{I}_i(k) \cap \mathbf{I}_i(k') \neq \emptyset$, we have either $a \in [a', b']$ or $b \in [a', b']$. If $a \in [a', b']$ then b cannot be strictly greater than b', in the other case, $s'_{b'}$, by definition greater than $\max\{s'_l | l \in \mathbf{I}_i(k')$ would belong to $[a, b]$, which would be in contradiction with the hypothesis. In the same way, if $b \in [a', b']$ then a cannot be strictly smaller than a' ; and the assertion 1 is proved.

The collection $\mathbf{I}_i$ can be represented as a tree of which nodes are associated to an interval $\mathbf{I}_1(k)$, or eventually to several intervals if they are confounded, and where the set of all the leafs having a given node $\mathbf{I}_1(k)$ as ancestor corresponds to the set of positions of s' bounded by it (figure 2).

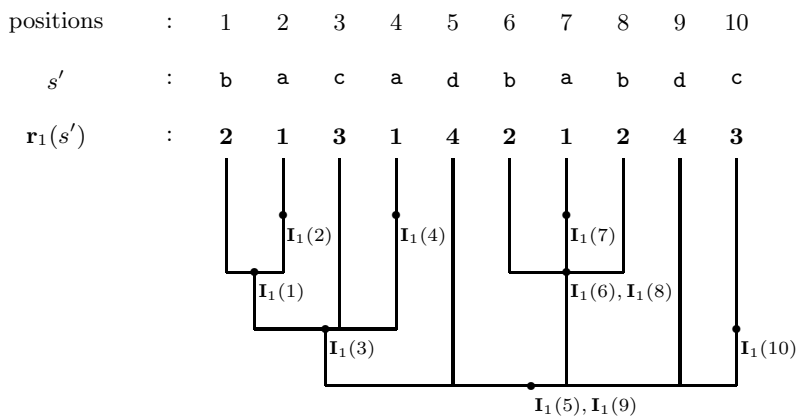

Fig. 2. Tree representation of the hierarchy of the $\mathbf{I}_1(k)$ of the example.

Lemma 2 *Let s and s' be two sequences. If $([i, j], [a, b])$ is a common interval of s and s' then there is a position k of s' such that $[a, b] = \mathbf{I}_i(k)$*

Proof. If $([i, j], [a, b]) \in \mathcal{C}(s, s')$ then the subsequences $s_{[i,j]}$ and $s'_{[a,b]}$ are alphabetically deliminated and verify $\mathcal{A}_{s_{[i,j]}} = \mathcal{A}_{s'_{[a,b]}} = \mathcal{A}$. By definition of $\mathbf{r}_i$, there is no symbol in $\mathcal{A}$ of infinite i-rank and each integer between 1

and $m = \max\{\mathbf{r}_i(x)|x \in \mathcal{A}\}$ appears as an i-rank of a symbol of $\mathcal{A}$: *i.e.* $\{\mathbf{r}_i(x)|x \in \mathcal{A}\} = \{1, \ldots, m\}$ and $[a, b]$ is i-complete. As $s'_{[a,b]}$ is alphabetically deliminated, the symbols s'_{a-1} and s'_{b+1} are not in $\mathcal{A}$. Moreover, each rank in $\{1, \ldots, m\}$ has an unique antecedent in $\mathcal{A}_{s'}$ and we have necessarily $\mathbf{r}_i(s'_{a-1})$ and $\mathbf{r}_i(s'_{b+1})$ strictly greater than m, equal to $\max\{\mathbf{r}_i(s'_l)|l \in [a, b]\}$. Let $k \in [a, b]$ be such that $\mathbf{r}_i(s'_k) = m$, it follows from the preceding and the definition of $\mathbf{I}_i(k)$ that $[a, b] = \mathbf{I}_i(k)$.

As for each position i of s, there are at most $|s'|$ subsequences $\mathbf{I}_i(k)$, we obtain the following bound.

Theorem 1 *Let s and s' be two sequences. The cardinal of $\mathcal{C}(s, s')$ is smaller than $|s| \times |s'|$.*

This bound is certainly not optimal. But as in the particular case of two n-permutations the optimal bound is $\binom{n}{2}$, the optimal bound for two sequences of length n grows as $O(n^2)$.

4 Algorithm

Let k be a position of s' with a finite i-rank $\mathbf{r}_i(s'_k)$, the *left-neighbour* (resp. the *right-neighbour*) of k is the greatest position smaller (resp. the smallest position greater) than k where a symbol of i-rank $\mathbf{r}_i(s'_k) + 1$ occurs if it exists (if there is not such symbol then k won't have a right- and/or a left-neighbour).

If k and k' are positions of s', the *i-distance* between k and k' is the maximum i-rank of symbols occurring in the subsequence $s'_{[k,k']}$ if $k \leq k'$ or in $s'_{[k',k]}$ if $k \geq k'$. The smallest distance of a position k of finite i-rank and a position of i-rank $\mathbf{r}_i(s'_k) + 1$ is reached either between k and its left-neighbour or between k and its right-neighbour.

Let j be a position of finite i-rank and L (resp. R) the i-distance between p and its left-neighbour (resp. its right-neighbour) if it exists and $+\infty$ if not. If R and L are both infinite, j don't have a *successor*. In the other case, the *successor* of j is:

- its left-neighbour if $L \leq R$;
- its right-neighbour if $L > R$.

An *i-path* of order m is a sequence p of m positions of s' verifying:

1. $\mathbf{r}_i(s'_{p_l}) = l$ for all $l \in \{1, \ldots m\}$;
2. for all $l \in \{1, \ldots m - 1\}$, p_{l+1} is the successor of p_l.

Lemma 3 *If p and q are two i-paths such that $p_j = q_j$ for an integer $j \in \{1, \ldots, \min(|p|, |q|)\}$ then $p_l = q_l$ for all $l \in \{j, \ldots, \min(|p|, |q|)\}$.*

Proof. By definition an i-path can be extend in only one way which depends only on its last element.

Theorem 2 *An interval* $\mathbf{I}_i(k)$ *is* i*-complete if and only if it contains an* i*-path of order* $\boldsymbol{r}_i(s'_k)$.

Proof. By definition, if $\mathbf{I}_i(k)$ contains an i-path of order $\mathbf{r}_i(s'_k) = \max\{\mathbf{r}_i(s'_l) | l \in \mathbf{I}_i(k)\}$, then it is complete.

Reciprocally let us assume $\mathbf{I}_i(k)$ to be complete. We will proceed by induction over the length of an i-path of $\mathbf{I}_i(k)$. In particular, there is at least a position $p_1 \in \mathbf{I}_i(k)$ such that $\mathbf{r}_i(p_1)$ and an i-path of order 1 belong to $\mathbf{I}_i(k)$. Assume that the i-path of order $l < \mathbf{r}_i(s'_k)$ is included in $\mathbf{I}_i(k)$. As $\mathbf{I}_i(k)$ is complete, there is at least a position $p \in \mathbf{I}_i(k)$ with a i-rank equal to $l+1$. As the i-distance of p_l and any position not in $\mathbf{I}_i(k)$ is greater than $\mathbf{r}_i(s'_k)$ (by definition of $\mathbf{I}_i(k)$). If p is the position of $\mathbf{I}_i(k)$ of i-rank $l+1$ having the smallest i-distance d from p_l which is the left- or right-neighbour of p_l, then p is the successor of p_l and we have an i-path of order $l+1$.

The lemma 2 and the theorem 2 ensure that $([i,j],[a,b])$ is a common interval of s and s' if and only if there exists a position k of s such that $[a,b] = \mathbf{I}_i(k)$ and $\mathbf{I}_i(k)$ contains an i-path p of order $\mathbf{r}_i(s'_k)$. Without loss of generality we can choose $k = p_{|p|}$ ($\mathbf{r}_i(p_{|p|}) = \mathbf{r}_i(k)$, $\mathbf{r}_i(p_{|p|}) \in \mathbf{I}_i(k)$ and lemma 1).

The main idea of our algorithm is, for each position i of s, to test if the elements of each i-path p of s' are included in $\mathbf{I}_i(p_{|p|})$. From the preceding, if this inclusion is granted then we have a common interval of the form $([i, \mathbf{p}_i(\mathbf{r}_i(s'_{p_{|p|}}))], \mathbf{I}_i(p_{|p|}))$ (see remark 1) and all the common intervals will be find by this way.

To perform this we need first to compute, fixing a position i of s, all the intervals $\mathbf{I}_i$. This can be done in a linear time and additional space using a stack S (see the part Compute the table of intervals $\mathbf{I}_i$ and the table $LeftDistance$ in the algorithm).

Another step is the computing of the successor of each position of s'. To do it, we need to determine the left- and right-neighbours of each position of s', which can be classically done in a linear time. The main difficulty is to compute the i-distances between each position and its left- and right-neighbours. In the algorithm these distances are stored in the tables $LeftDistance$ and $RightDistance$. The table $LeftDistance$ is compute in the same time and using the same stack than the intervals. The table $RightDistance$ is compute in the same way. It need for each position to perform a binary search in the stack. So the complexity of this part of the algorithm is in the worst case $O(|s'| \log(|s'|))$. The table of successors is easy to compute in a linear time using the tables $LeftDistance$ and $RightDistance$.

The last step is straightforward. Starting by all the positions of i-rank 1 of s', we extend iteratively all the i-paths starting from these and test the preceding inclusion at each iteration. Following the lemma 3, we don't need to consider each position more than once in an i-path ($PositionParsed$ is a boolean table maintained to avoid such useless iterations). Complexity of this part becomes linear with the length of s' in time and space.

Algorithm 1 Compute the set of common intervals between two sequences

for all position i of s **do**
  Compute the rank table $\mathbf{r}_i$, $iMax$ the max of finite i-ranks and the corresponding positions $\mathbf{p}_i$
  Compute the chain tables $LeftNeighbour$ and $RightNeighbour$
  Initialize the tables $LeftDistance$ and $RightDistance$ to $+\infty$
  Initialize the stack S
  Compute the table of intervals $\mathbf{I}_i$ and the table $LeftDistance$
  for $k = 0$ **to** $|s'| + 1$ **do**
    if $\mathbf{r}_i(s'_k) < iMax$ and $LeftNeighbour(k)$ is not EMPTY **then**
      $t \leftarrow$ the smallest position in the stack $\geq LeftNeighbour(k)$ (computed by binary search)
      $LeftDistance[k] \leftarrow \mathbf{r}_i(s'_t)$
    end if
    while $\mathbf{r}_i(s'_{\mathbf{top}(S)}) < \mathbf{r}_i(s'_k)$ **do**
      $\mathbf{I}_i(\mathbf{top}(\mathrm{S}))$.end $\leftarrow k - 1$
      pop(S)
    end while
    if $\mathbf{r}_i(s'_{\mathbf{top}(S)}) \neq \mathbf{r}_i(s'_k)$ **then**
      $\mathbf{I}_i(k)$.start $\leftarrow \mathbf{top}(S) + 1$
    else
      $\mathbf{I}_i(k)$.start $\leftarrow \mathbf{I}_i(\mathbf{top}(S))$.$start$
    end if
    push(k, S)
  end for
  Compute the table $RightDistance$
  for $k = |s'| + 1$ **to** 0 **do**
    if $\mathbf{r}_i(s'_k) < iMax$ and $RightNeighbour(k)$ is not EMPTY **then**
      $t \leftarrow$ the greatest position in the stack $\leq RightNeighbour(k)$ (computed by binary search)
      $RightDistance[k] \leftarrow \mathbf{r}_i(s'_t)$
    end if
    while $\mathbf{r}_i(s'_{\mathbf{top}(S)}) < \mathbf{r}_i(s'_k)$ **do**
      pop(S)
    end while
    push(k, S)
  end for
  Compute the table $Successor$
  for all position k of s' **do**
    if $\mathbf{r}_i(s'_k) < iMax$ **and** $\min(LeftDistance[k], RightDistance[k]) < +\infty$ **then**
      if $LeftDistance[k] \leq RightDistance[k]$ **then**
        $Successor(k) \leftarrow LeftNeighbour(k)$
      else
        $Successor(k) \leftarrow RightNeighbour(k)$
      end if
    else
      $Successor(k) \leftarrow$ EMPTY
    end if
  end for
  Parse the i-paths and output the common intervals
  Initialize the table $PositionParsed$ to FALSE
  for all position j of i-rank 1 of s' **do**
    $BoundL \leftarrow j$; $BoundR \leftarrow j$
    repeat
      $BoundL \leftarrow \min(j, BoundL)$; $BoundR \leftarrow \max(j, BoundR)$
      if $[BoundL, BoundR] \subset \mathbf{I}_i(j)$ **then**
        output $([i, \mathbf{p}_i(\mathbf{r}_i(s'_k))], \mathbf{I}_i(j))$ a common interval
      end if
      $PositionParsed(j) \leftarrow$ TRUE ; $j \leftarrow Successor(j)$
    until j is EMPTY **or** $PositionParsed(j)$ is TRUE
  end for
end for

For each position i of s, the greater complexity to compute common intervals with left bound i in their s-component is $O(|s'|\log(|s'|))$. If the length of the sequences s and s' are $O(n)$, the general complexity of the algorithm is $O(n^2\log(n))$.

Acknowledgements

We thank Mathieu Raffinot for introducing the problem, references and discussions and Marie-Odile Delorme for helpful comments and suggestions. We also thank the referees for comments and indicating the reference [1].

References

1. A. Amir, A. Apostolico, G. Landau and G. Satta, Efficient Text Fingerprinting Via Parikh Mapping, *Journal of Discrete Algorithms*, to appear.
2. A. Bergeron, S. Heber, J. Stoye, Common intervals and sorting by reversals: a marriage of necessity, *ECCB 2002*, 54-63.
3. T. Dandekar, B. Snel, M. Huynen and P. Bork, Conservation of gene order: A fingerprint of proteins that physically interact, *Trends Biochem. Sci.* **23** (1998), 324-328.
4. S. Heber, J. Stoye, Finding All Common Intervals of k Permutations, *CPM 2001*, 207-218.
5. S. Heber, J. Stoye, Algorithms for Finding Gene Clusters, *WABI 2001*, 252-263.
6. A. R. Mushegian and E. V. Koonin, Gene order is not conserved in bacterial evolution, *Trends Genet.* **12** (1996), 289-290.
7. J. Tamames, M. Gonzales-Moreno, J.Mingorance, A. Valencia, Conserved clusters of functionally related genes in two bacterial genomes, *J. Mol. Evol.* **44** (1996), 66-73.
8. T. Uno and M. Yagira, Fast algorithms to enumerate all common intervals of two permutations, *Algorithmica*, **26**(2) (2000), 290-309.