BIOINFORMATICS
Vol. 1 no. 1 2002
Pages 1–10

# Common Intervals and Sorting by Reversals: A Marriage of Necessity

A. Bergeron [1,2], S. Heber [3] and J. Stoye [4]

[1]LaCIM, Université du Québec à Montréal, Canada, [2]Institut Gaspard-Monge, Université Marne-la-Vallée, France, [3]Department of Computer Science and Engineering, University of California, San Diego, USA and [4]Technische Fakultät, Universität Bielefeld, Germany

**ABSTRACT**

This paper revisits the problem of sorting by reversals with tools developed in the context of detecting common intervals. Mixing the two approaches yields new definitions and algorithms for the reversal distance computations, that apply directly on the original permutation.

Traditional constructions such as recasting the signed permutation as a positive permutation, or traversing the overlap graph to analyze its connected components, are replaced by elementary definitions in terms of intervals of the permutation. This yields simple linear time algorithms that identify the essential features in a single pass over the permutation and use only simple data structures like arrays and stacks.

**Contact:** Jens Stoye, AG Genominformatik, Technische Fakultät, Universität Bielefeld, Postfach 10 01 31, 33501 Bielefeld, Germany. E-mail: stoye@TechFak.Uni-Bielefeld.DE

## INTRODUCTION

Let $\pi$ be a permutation of the integers between 1 and $n$ which are provided with a plus or minus sign,

$$\pi = (\pi_1\ \pi_2 \ldots \pi_n).$$

A *reversal* $\rho(i, j)$ is an operation that reverses the block of consecutive elements from $i$ to $j$ in $\pi$, while changing their signs. The *reversal distance* $d(\pi)$ is the minimum number of reversals that transform $\pi$ into the identity permutation,

$$(1\ 2\ \ldots n).$$

A *safe* reversal $\rho$ for the permutation $\pi$ is a reversal such that $d(\rho\pi) = d(\pi) - 1$. Finding a sequence of $d(\pi)$ safe reversals is called the *sorting by reversal problem*. Its solutions are far from unique (Siepel, 2002; Bergeron *et al.*, 2002), but elementary methods to approach this problem are still scarce.

Permutations, and the reversal operation, are useful tools in the comparative study of genomes (Sankoff, 1992). The genome of a species can be thought of as a set of ordered sequences of genes – the ordering devices being the chromosomes –, each gene having an orientation given by its location on the DNA double strand. Different species often share similar genes that were inherited from common ancestors. However, these genes have been shuffled by mutations that modified the content of chromosomes, the order of genes within a particular chromosome, and/or the orientation of a gene. Comparing two sets of similar genes appearing along a chromosome in two different species yields a (signed) permutation. It is widely accepted that the reversal distance of this permutation provides a good estimate of the evolutionary distance between the two species.

Computing the reversal distance, or deciding whether a reversal is safe, traditionally requires to recast the signed permutation as an unsigned permutation of $2n$ elements, and to construct a graph, called the *overlap graph* (Bafna and Pevzner, 1996; Hannenhalli and Pevzner, 1999). Connected components and cycles of this graph play a crucial role in the sorting by reversal problem. In this paper, we show that it is possible to bypass entirely the construction of the overlap graph. We characterize its essential features in terms of subsets of elements of the signed permutation, using tools borrowed from the algebra of common intervals (Uno and Yagiura, 2000; Heber and Stoye, 2001).

With this approach, we were able to derive elementary linear time algorithms to solve problems involving reversal distance computations: the detection of unoriented connected components and the reversal distance computation. All these algorithms make a single "pass" on the elements of the permutation, using stacks to record important features. Analyzing, for example, the potential effects of a reversal $\rho(i, j)$ on the structure of a permutation can be done directly on the original permutation $\pi$, without actually performing the reversal.

These algorithms can be particularly useful in applications such as the reversal median problem (Siepel, 2002), where huge numbers of optimal sequences of reversals satisfying secondary constraints are tested. In such applica-

tions, the detection of safe reversals turns out to be the bottleneck of the procedures, even when using linear time algorithms (Bader *et al.*, 2001). Indeed, any algorithm that operates on the overlap graph must first perform the candidate reversal, update the graph accordingly, and then undo its work in order to test another candidate.

Finally, let's mention that the relations between the sorting problem and common intervals made a first appearance – without explicit mention of the structure – by Kaplan *et al.* (1999), and then by Bergeron (2001), as alternate definitions of hurdles. The present paper constitutes, to the best of our knowledge, the first complete treatment of all connected components of the overlap graph in terms of common intervals.

## BACKGROUND

As usual, we will consider signed permutations framed by 0 and $n$:

$$\pi = (\ 0\ \ \pi_1\ \pi_2 \ldots \pi_{n-1}\ \ n\ ).$$

An *element* $i$ of the permutation is an unsigned integer between 0 and $n$, and each element has a sign, $+$ or $-$. The positive sign $+$ may be omitted. Note that signs are only used to partition the elements in two subsets, and do not carry their usual additive properties. By convention, 0 and $n$ are always positive.

An *interval* $(\ \ \pi_k \ldots \pi_j\ \ )$ of $\pi$ is an ordered set of consecutive elements of $\pi$. When the set of (unsigned) elements $\{\pi_k, \ldots, \pi_j\}$ is a set of consecutive integers, the pair of indices $[k, j]$ is said to be a *common interval* (with the identity permutation). For example, in the permutation:

$$\pi = (\ 0\ \ -2\ \ -4\ \ 3\ \ 1\ \ 5\ ),$$

the unsigned elements of the interval $(\ -2\ \ -4\ \ 3\ )$ can be reordered as $\{2, 3, 4\}$, thus $[1, 3]$ is a common interval of $\pi$.

When elements $i$ and $i+1$ have opposite signs, it is always possible to create a consecutive pair

$$i\ \ i+1 \quad \text{or} \quad -(i+1)\ \ -i$$

in the permutation with a single reversal. These reversals are the basic operations in the sorting problem.

Unfortunately, some of these reversals are not safe. For example, the reversal of elements $(\ -2\ \ -4\ )$ in the permutation $\pi = (\ 0\ \ -2\ \ -4\ \ 3\ \ 1\ \ 5\ )$ creates a positive permutation and no subsequent reversal can form a consecutive pair.

A characterization of safe reversals is given by the following theorem:

THEOREM 1. (Hannenhalli and Pevzner, 1995) *A reversal that creates a consecutive pair is safe if it does not create new unoriented connected components in the overlap graph.*

The purpose of the next sections is to give an elementary definition of unoriented connected components of the overlap graph, enabling us to detect and analyze these components directly on the permutation, without constructing the overlap graph, or traversing its vertices. But, first, we present a detailed example of the process of sorting a signed permutation.

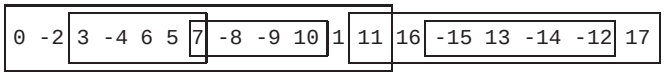
## AN INFORMAL APPROACH TO SORTING

Most permuations, especially those coming from biological data, can be optimally sorted with simple tools. As an illustration of this fact, we solve in this section a nontrivial sorting problem. We also use the opportunity to present in an informal way the structures that are formalized in the following sections. The running example has respectable size for good reason: small permutations are often bad indicators of the behavior of large ones with respect to sorting.

Consider the permutation:

```
0 -2 3 -4 6 5 7 -8 -9 10 1 11 16 -15 13 -14 -12 17
```

It is not clear, at first sight, how to optimally sort it. However, a basic observation is that most problems, like this one, can be decomposed into independent subproblems. The following diagram outlines five blocks in the permutation.




Each block contains a set of consecutive (unsigned) elements, the smallest and greatest being at the ends of the blocks. Both values are positive if the smallest is at the left, and both are negative otherwise.

If two blocks are consecutive, as in

```
0 -2 3 -4 6 5 7 -8 -9 10 1 11 16 -15 13 -14 -12 17
```

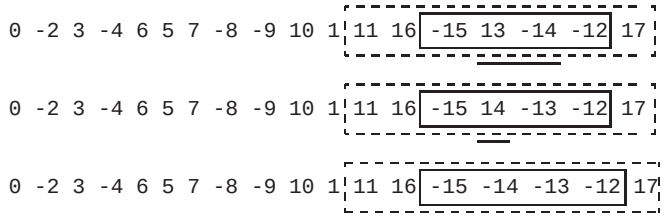
then it is easy to see that sorting the left block can be done independently of the right one. If one block is contained in the other, as in

```
0 -2 3 -4 6 5 7 -8 -9 10 1 11 16 -15 13 -14 -12 17
```

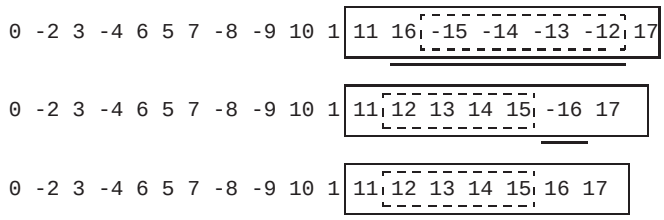
the same remark applies, in the sense that the inner block can be treated as a unit, here the integers from $-15$ to $-12$, from the point of view of the outer block.

In the context of the sorting by reversal problem, each block corresponds to a connected component of the permutation. An *easy-to-sort* block is a block that contains both positive and negative elements, taking each inner block as a unit, and assigning to it the sign of its endpoints. For example, the following sequence of two reversals, indicated by bold underlines, sorts the block between $-15$

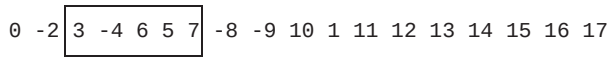
and $-12$ in decreasing order:



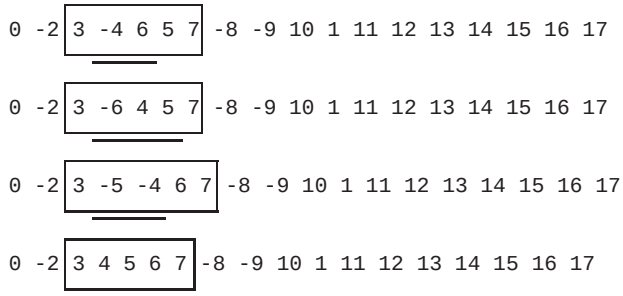

Once the inner block is sorted, one can sort the outer block, this time in ascending order. Note that reversals applied to the outer block treat the inner block as a single unit: there is no need to break it. Moreover, remark that the inner block could have been sorted after the outer block.



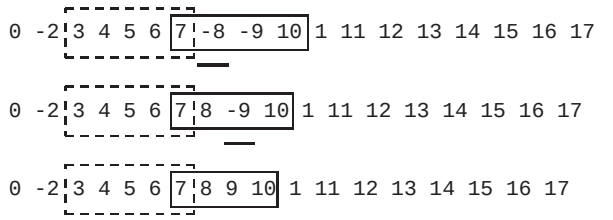

We now turn to the first part of the permutation, and consider the block:



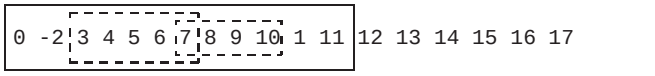

It is tempting to reverse the element $-4$, since it would create the adjacency $(3\ \ 4)$, but it would also create an unsorted block with only positive elements, that is, a *hard-to-sort* block. It is fortunately possible to avoid this situation by considering an alternative strategy:




The next block is sorted with two simple reversals:




The two preceding blocks were inner blocks of a larger one:




which can be sorted with three reversals (left as an exercise). The original permutation can thus be sorted in 12 reversals, and, since each of the reversals created at least one adjacency, the theory guarantees that this number is optimal.

## CONNECTED COMPONENTS OF A SIGNED PERMUTATION

We now formally define, among others, the terms *block* (*framed common interval*), *easy-to-sort* (*oriented*), and *hard-to-sort* (*unoriented*) from the previous section.

### Elementary Reversals

Our first definition generalizes the notion of reversals that create consecutive pairs to any pair of elements $i$ and $i+1$ of the permutation.

DEFINITION 1. *Let $\pi = (0\ \ \pi_1\ \ \pi_2 \ldots \pi_{n-1}\ \ n)$ be a signed permutation. To each element $i$, $0 \leq i < n$, we associate the* elementary reversal $r_i$ *consisting of the interval from $i$ to $i+1$, excluding:*

*i* *if it is positive and precedes $i+1$,*
*or negative and succeeds $i+1$, and*
*i+1* *if it is negative and precedes $i$,*
*or positive and succeeds $i$.*

Note that an excluded endpoint happens to be "well ordered" with respect to the other. For example, if

$$\pi = (\ 0\ \ -6\ \ 3\ \ -4\ \ 5\ \ 2\ \ -1\ \ 7\ \ 9\ \ 8\ \ 10\ )$$

then

$$r_0\ \ =\ \ (\ -6\ \ 3\ \ -4\ \ 5\ \ 2\ \ -1\ )$$

in which 0 has been removed, since the endpoint 0 precedes 1, and 0 is positive. In the same way,

$$r_1\ \ =\ \ (\ 2\ )$$
$$r_2\ \ =\ \ (\ 3\ \ -4\ \ 5\ \ 2\ )$$
etc.

When elements $i$ and $i+1$ have opposite signs, the reversal $r_i$ is said to be *oriented*, otherwise it is *unoriented*. An elementary reversal can also be empty, when the corresponding pair is consecutive.

The following definition is crucial, and will provide the link with the Hannenhalli-Pevzner theory.

DEFINITION 2. *An elementary reversal $r_i$* overlaps *another elementary reversal $r_j$ if $r_i$ contains either $j$ or $j+1$, but not both.*

For example, in

$$\pi = (\ 0\ -6\ \ 3\ -4\ \ 5\ \ 2\ -1\ \ 7\ \ 9\ \ 8\ \ 10\ ),$$

we have

$$\begin{array}{rcl} r_1 & = & (\ 2\ ) \\ r_2 & = & (\ 3\ -4\ \ 5\ \ 2\ ). \end{array}$$

The reversal $r_1$ overlaps $r_2$, since it contains the element 2 but not 3, and $r_2$ overlaps $r_1$, since it contains the element 2, but not 1.

An easy consequence of this definition is that, if $r_i$ overlaps $r_j$, then performing $r_i$ will change the orientation of $r_j$, since only one endpoint of $r_j$ will change sign. In the above example, $r_2$ is unoriented, but after performing $r_1$, it is oriented: $(\ 3\ \ -4\ \ 5\ \ -2\ )$.

The overlap relation also turns out – conveniently – to be symmetric:

PROPOSITION 1. *$r_i$ overlaps $r_j$ if and only if $r_j$ overlaps $r_i$.*

*Proof:* Suppose that $r_i$ contains one endpoint, say $j$, of $r_j$. If $j$ is different from $i$ and $i+1$, then the interval $r_j$ will certainly contain one endpoint of $r_i$.

The interesting cases occur when either $j = i+1$ or $i = j+1$. Consider the first case; the other one can be proved in a similar way. Suppose that we have three elements, $i$, $i+1$, and $i+2$. If in the permutation $i+1$ lies between $i$ and $i+2$, then, from the assumption that $r_i$ contains $i+1$, it follows that also $r_{i+1}$ contains $i+1$, thus $r_{i+1}$ contains one endpoint of $r_i$.

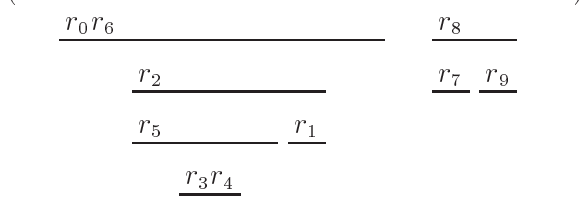
If in the permutation both $i$ and $i+2$ lie on the same side of $i+1$, then $r_i$ contains $i+1$ if and only if $r_{i+1}$ does not contain $i+1$. If $r_i$ contains $i+1$ and not $i+2$, the element $i+2$ must be farther from $i+1$ than $i$ is, thus $r_{i+1}$ contains $i$. ■

DEFINITION 3. *A* connected component *of a permutation is a connected component of the graph of the overlap relation.*

The overlap relation of Definition 2 is similar to the arc overlap relation of the usual Hannenhalli-Pevzner theory (see Kaplan *et al.* (1999) for example), but defined on the original intervals of the permutation and not on the corresponding unsigned permutation on $2n$ points. The two relations yield isomorphic graphs: indeed, in the arc overlap graph each vertex corresponds to a reversal, and two reversals are connected if and only if performing one modifies the orientation of the other.

The advantages of defining the overlap relation on the intervals of the original permutation will reside in the possibility to detect and analyze the connected components by direct inspection of the permutation.

EXAMPLE 1. *In the permutation*




*we underlined the elementary reversals. There are four connected components:*

$$\{r_0, r_6\}, \{r_1, r_2, r_5\}, \{r_3, r_4\}, \{r_7, r_8, r_9\}.$$

*How one finds them, apart working from scratch from the definition, is not immediately clear. But it should be trivial by the end of the next section.*

## Spans of Connected Components

The *span* of a connected component $C$ is the minimal interval containing the elementary reversals of $C$. In Example 1, the span of $\{r_1, r_2, r_5\}$ is $(\ 3\ \ -4\ \ 5\ \ 2\ )$. Note here that the span of this component is a common interval, in the permutation framed by its extremal elements 1 and 6. We will show that this feature characterizes the connected components.

DEFINITION 4. *A* framed *common interval $F$ in a signed permutation is an interval of the form:*

$$\begin{array}{cccc} & (\ m & \pi_k \ldots \pi_j & M\ ) \\ \textit{or} & (\ -M & \pi_k \ldots \pi_j & -m\ ), \end{array}$$

*such that:*

*1. The values $m$ and $M$ are respectively the minimum and maximum of the elements of $F$;*

*2. $[k-1, j+1]$ is a common interval;*

*3. $F$ is not the union of shorter intervals with the above two properties.*

The interval $(\ \pi_k \ldots \pi_j\ )$ is called the *interior* of the framed common interval. We have the following theorem:

THEOREM 2. *An interval is a framed common interval if and only if its interior is the span of a connected component.*

*Proof:* We first remark that any elementary reversal included in a framed common interval $F$ cannot contain the endpoints of the interval. Indeed, both $r_m$ and $r_{M-1}$ are included in $F$ and do not contain $m$ or $M$, and since the interior of $F$ is a common interval, for any other element $i$, $m < i < M-1$, $i+1$ is in the interior of $F$. Thus, any connected component lies completely within any framed common interval.

Suppose that $I = (\ \pi_k \dots \pi_j\ )$ is the span of a connected component $C$. If the span is empty, the result holds trivially. Otherwise, by definition, there is at least one reversal $r$ in $C$ that contains $\pi_k$, and one reversal $r'$ that contains $\pi_j$. Moreover, no elementary reversal can contain one element in $(\ \pi_k \dots \pi_j\ )$, and one outside, since it would overlap either $r$ or $r'$.

If $i$ is an element of $I$, then $r_i$ and $r_{i-1}$ are also included in this interval. Otherwise, $r_i$, for example, would be disjoint from $I$, implying that $i = \pi_k$ or $i = \pi_j$, and either $r$ or $r'$ overlaps $r_i$.

Thus, both $i-1$ and $i+1$ are in the interval $I' = (\ \pi_{k-1} \dots \pi_{j+1}\ )$, implying that $\pi_{k-1}$ and $\pi_{j+1}$ are the extremal elements of the interval, and that $[k-1, j+1]$ is a common interval. Since $\pi_{k-1}$ and $\pi_{j+1}$ are not in the span of $C$, they are respectively equal to either $m$ and $M$, or to $-M$ and $-m$. Finally, $I'$ cannot be the union of shorter framed common intervals since a connected component must lie within any framed common interval.

On the other hand, if $F = (m \dots M)$ is a framed common interval, then the span of the component of $r_m$ is framed by $(m \dots M')$, with $M' \leq M$. But if $M' < M$, then the interval $(M \dots M')$ is a common interval that begins with its minimal value and ends with its maximal value, and $F = (m \dots M') \cup (M' \dots M)$.

The case $(-M \dots -m)$ is treated similarly. ■

EXAMPLE 1 (CONT'D). *Consider again the permutation*

$$\pi = (\ 0\ \ -6\ \ 3\ \ -4\ \ 5\ \ 2\ \ -1\ \ 7\ \ 9\ \ 8\ \ 10\ ).$$

*Framed common intervals are easily identified as* $(0 \dots 7)$, $(-6 \dots -1)$, $(3 \dots 5)$, *and* $(7 \dots 10)$. *The interval* $(0 \dots 10)$ *is not considered because it is the union of* $(0 \dots 7)$ *and* $(7 \dots 10)$. *Each of these intervals outlines a connected component whose reversals can be readily identified.*

### Oriented Components

Inclusion of span induces a partial order on the non-trivial connected components. A component $C$ contains a non-empty elementary reversal $r$ if $r$ is included in the span of $C$, but not in the span of components smaller than $C$.

DEFINITION 5. *A connected component with non-empty span is* oriented *if it contains at least one oriented elementary reversal, otherwise, it is* unoriented.

Define the *elements* of a connected component as the set of endpoints of the elementary reversals it contains. In the permutation

$$\pi = (\ 0\ \ -6\ \ 3\ \ -4\ \ 5\ \ 2\ \ -1\ \ 7\ \ 9\ \ 8\ \ 10\ ),$$

for example, the connected component $\{r_0, r_6\}$ has elements $0$, $-6$, $-1$, and $7$. The connected component $\{r_7, r_8, r_9\}$ has elements $7$, $9$, $8$, and $10$.

The following proposition will allow easy identification of oriented and unoriented components.

PROPOSITION 2. *A connected component is unoriented if and only if all its elements have the same sign in the permutation.*

*Proof:* Clearly, if a component is oriented, its elements do not have the same sign. Consider the case of an unoriented component $C$ with $m$ and $M$ as extremal (unsigned) values. Both $m$ and $M$ are elements of the connected component since $r_m$ and $r_{M-1}$ belong to the component. The elements of $C$ are all the integers from $m$ to $M$, with "gaps" corresponding to spans of components smaller than $C$. Since the two framing elements of these smaller connected components have the same sign, any change of sign in the ordered sequence of elements of $C$ must occur within this sequence. ■

## A LINEAR TIME ALGORITHM TO DETECT UNORIENTED COMPONENTS

The goal of this section is to develop a linear time algorithm to identify the (un)oriented connected component of a permutation using the characterization of Theorem 2. Its basic principle is quite simple, but the general case requires a little care. For reasons of clarity, we will begin with the case of positive permutations.

### Connected Components of Positive Permutations

Our first algorithm identifies framed common intervals in a positive permutation. It is based on the following property:

LEMMA 1. *If* $F = (\ m\ \ \pi_k \dots \pi_j\ \ M\ )$ *is a framed common interval of a positive permutation, then each* $\pi_i$, $k \leq i \leq j$, *has either a smaller element following it in the interior of* $F$, *or a greater element preceding it.*

*Proof:* If all elements smaller than $\pi_i$ come before $\pi_i$, and all elements greater come after, then both $(m \dots \pi_i)$ and $(\pi_i \dots M)$ are framed common intervals, and $(m \dots M)$ is the union of the two. ■

In order to use Lemma 1, we need to be able to refer to greater elements preceding a given element. Therefore define $M_i$ to be the nearest element of the permutation that precedes $\pi_i$ and is greater than $\pi_i$ (set $M_i$ to $n$, if such an element does not exists). The basic algorithm to find the span of connected components in a positive permutation has the following structure:

**Algorithm 1**
$S$ is a stack that contains the index 0. The top of $S$ is always denoted by $s$.

For $i$ from 1 to $n$ do
1. While ($\pi_i < \pi_s$ or $\pi_i > M_s$)
   Unstack the top index from S
2. Test whether the interval $[s, i]$ is a framed common interval.
3. Stack the index $i$.

Before discussing how to perform the test in line 2, we argue that all framed common intervals will eventually be tested. Indeed, if $(\pi_s \dots \pi_j)$ is a framed common interval, then:

PROPOSITION 3. *The index $s$ will not be unstacked before $j$ is stacked.*

*Proof:* Any element between $\pi_s$ and $\pi_j$ is greater than $\pi_s$, since $\pi_s$ is the minimum value of the common interval.

Any element between $\pi_s$ and $\pi_j$ is smaller than $M_s$, since $M_s$ must be at least greater than $\pi_j$ (remember that all elements between $\pi_s$ and $\pi_j$ follow $\pi_s$ in the permutation).

Thus, no element between $\pi_s$ and $\pi_j$ can unstack $s$. ■

PROPOSITION 4. *All indices between $s$ and $j$ are eventually unstacked before $j$ is stacked.*

*Proof:* Let $i$ be between $s$ and $j$. By Lemma 1, $\pi_i$ has either a smaller element following it, or a greater element preceding it in the interval between $s$ and $i$. In the first case, the first such element will unstack $i$. In the second case, since $M_i$ is smaller than $\pi_j$, if $i$ stays in the stack up to the end, $\pi_j$ will unstack it. ■

We now turn to the problem of testing whether the interval $[s, i]$ is a framed common interval. The first elementary test is to count the number of elements between these two indices. Indeed, a necessary condition for $[s, i]$ to be a framed common interval is:

$$\pi_i - \pi_s = i - s.$$

If $s$ is the top of the stack, then all elements in the interval $[s + 1, i]$ are greater than $\pi_s$. One must also check if only values smaller than $\pi_i$ are in the interval. This is done by keeping track of the maximal element that occurred between two consecutive stacked indices. The implementation is rather straightforward and details can be found in the Appendix.

Finally, in order to complete the analysis of the algorithm, we must show how to compute efficiently the values $M_i$ used in the main loop.

**Algorithm 2**
$S$ is a stack that contains the value $n$. The top of $S$ is always denoted by $s$.

$M_0 \leftarrow n$
For $i$ from 1 to $n$ do
  If $\pi_{i-1} > \pi_i$
    $M_i \leftarrow \pi_{i-1}$
    Stack the value $\pi_{i-1}$
  Else
    While $s < \pi_i$
      Unstack the top element from S
    $M_i \leftarrow s$

The correctness of this algorithm is based on the following remark. If $M_i = \pi_k$, then $M_i$ is greater than all values in the interval $(\ \pi_{k+1} \dots \pi_i\ )$, implying that $M_i$ will be stacked when reading $\pi_{k+1}$, and that no value in the interval can unstack $M_i$. But $\pi_i$ will unstack all values in the interval, given the chance. The same argument holds for $M_i = n$, by setting $k = 0$.

The results of this section are summarized by:

THEOREM 3. *All framed common intervals of a positive permutation can be found with Algorithms 1 and 2 in $\mathcal{O}(n)$ time and space.*

## Connected Components of Signed Permutations

We can now adapt the algorithms of the preceding section to the problem of identifying connected components of signed permutations. For components framed by positive elements $(m \dots M)$, a simple variant of Algorithm 1 will do. Indeed, if $\pi$ is a signed permutation, and $\pi^+$ its unsigned version, any framed common interval with positive endpoints of $\pi$ is the union of framed common intervals of $\pi^+$. The idea is to use Algorithm 1 on $\pi^+$, but avoid stacking negative elements. In order to show that this indeed works, we first state a modified version of Lemma 1:

LEMMA 2. *If $F = (\ m\ \ \pi_k \dots \pi_j\ \ M\ )$ is a framed common interval of a signed permutation, then each $\pi_i$, $k \le i \le j$, has either a smaller element following it in the interior of $F$, or a greater element preceding it, or $\pi_i$ is negative.*

The proof is similar to the proof of Lemma 1 and highlights the possible problem with signed permutations. For example, the permutation $\pi = (0\ \ 2\ \ 1\ \ {-3}\ \ 4)$ has only one component. Its span is $(0\ \ 2\ \ 1\ \ {-3}\ \ 4)$. The unsigned version of $\pi$ has two connected components whose framed spans are $(0\ \ 2\ \ 1\ \ 3)$ and $(3\ \ 4)$.

Using Lemma 2, it is possible to show the equivalents of Propositions 3 and 4, remembering that negative elements are not stacked, but that their absolute values can be used to unstack elements.

Identifying components that are framed by $-M$ and $-m$ can be done by reversing $\pi$, except for its endpoints 0 and $n$, and applying the above algorithm. It can also be done by "reversing" the algorithm. This latter possibility, implemented in the code given in the Appendix, allows the detection of components in a single pass on the permutation, using four stacks.

We have also the equivalent of Theorem 3:

THEOREM 4. *All framed common intervals of a signed permutation can be found with the modified versions of Algorithms 1 and 2 in $\mathcal{O}(n)$ time and space.*

Finally, the problem of detecting if all elements of a component have the same sign can be done with an appropriate marking of the top of the stack, without affecting the running time. Here is the version for components with span $(m \dots M)$. In this case, the component will be unoriented if all the negative elements in the interval are properly "shielded" by the positive frames of smaller components. A suitable way to mark the top of the stack is (with the notation of Algorithm 1):

1. If $\pi_i$ is negative, mark the top of the stack.
2. If a marked element is removed from the stack, mark the new top of the stack.
3. If $[s, i]$ is a connected component, unmark $s$, the top of the stack.

This is also implemented in the code of the Appendix.

## COMPUTING THE REVERSAL DISTANCE

Having an efficient algorithm for the detection of unoriented components, we can now delineate the complete algorithm to compute the reversal distance in linear time.

In (Hannenhalli and Pevzner, 1999), the minimal number of reversals $d(\pi)$ necessary to sort a permutation $\pi = (0\ \ \pi_1\ \pi_2 \dots \pi_{n-1}\ \ n)$ is shown to be equal to:

$$d(\pi) = n - c + h + f,$$

where $c$ is the number of *cycles*, $h$ is the number of *hurdles*, and $f$ is a correction factor equal to 1 or 0 according to whether $\pi$ is a *fortress* or not. We recall the definitions of these parameters in the following paragraphs.

Consider the set of unoriented components of a permutation that have more than two elements. These components are partially ordered by span inclusion. Each minimal element of this order is a *hurdle*. In addition, the maximal element is a hurdle if it exists, and if none of its elements lies between two hurdles.

For example, the permutation of Figure 1, in which all spans of unoriented components are boxed, has two hurdles. There are three unoriented components, and two of them are hurdles: the minimal one, whose elements are $\{4, 6, 5, 7\}$, and the maximal one, whose elements are $\{0, 2, 8, 1, 9\}$. The permutation of Figure 2 also has three components and two hurdles. However, the maximal element is not a hurdle since it has elements between the two minimal components.

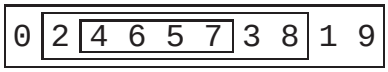



**Fig. 1.** A permutation with two hurdles

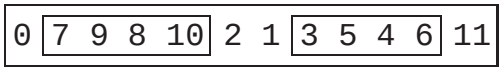


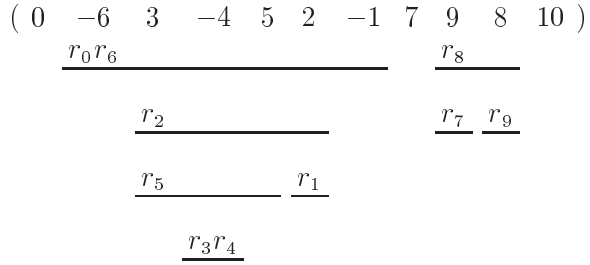

**Fig. 2.** Another permutation with two hurdles

If one sorts the elements of a hurdle, it is eliminated. A *super-hurdle* is a hurdle whose elimination does not diminish the number of hurdles in the resulting permutation. For example, if one sorts the hurdle $(4\ 6\ 5\ 7)$ in the permutation of Figure 1, the resulting permutation has still two hurdles.

A *fortress* is a permutation that has an odd number of hurdles, all of which are super-hurdles.

Finally, two elementary reversals are *linked* if they are consecutive, or if one is a prefix or suffix of the other. A *cycle* is a closed chain of linked reversals, and they are easily enumerable.

Applying all these definitions to the permutation of Example 1:




we get $n = 10$, $c = 4$, $h = 1$, and $f = 0$. Thus $d(\pi) = 7$.

The main problem in computing the reversal distance is thus the classification of unoriented components as hurdles, super-hurdles, and non-hurdles. Identifying hurdles and super-hurdles that are minimal elements is easy. Indeed, the algorithm given in the Appendix enumerates the components from left to right, for components whose spans are not included into one another, and from inner components to outer components, for components whose spans are nested.

An unoriented component $U$ is a hurdle if it is the first one to be identified by the algorithm, or if its span does not contain the span of the previous unoriented component. If the permutation has only one minimal hurdle, then the

maximal unoriented component, if it exists, will also be a hurdle.

For hurdles that are minimal elements, a simple way to decide if they are super-hurdles is to test whether their immediate ancestor in the partial order contains only one oriented component. As noted for example by Hannenhalli and Pevzner (1999) and Kaplan *et al.* (1999), by considering the circular order on the interval $[0 \dots n]$, the maximal hurdle, if it exists, becomes a minimal element if the elements of the permutation are shifted such that the element $0$ lies within any other hurdle.

Identifying hurdles as super-hurdles is relevant only when the permutation has at least three hurdles. A hurdle $U$ is a super-hurdle if the span of the following unoriented component contains the span of $U$ and does not contain the span of the hurdle preceding $U$.

In order to test whether the maximal component is a hurdle or a super-hurdle, we apply the algorithm of the Appendix to the shifted elements of the permutation, using an element of the last minimal hurdle as the new element $0$.

## CONCLUSIONS

This paper focused on the detection of oriented and unoriented connected components of a signed permutation, using a characterization of these components in terms of common intervals that can be applied directly to the permutation. We showed that connected components can be identified without the usual constructions of the positive permutation on $2n$ points, or the overlap graph.

Using the definition of connected components as common intervals, we were able to deduce an elementary linear time algorithm to identify the oriented and unoriented components. This algorithm can be used to test whether a reversal is safe without actually performing the reversal. Indeed, since the effect of a reversal is to change the order of a block of elements, a simple function of the indices can simulate the reversed permutation.

Finally, we have also shown how the algorithm has to be extended in order to compute the reversal distance of a signed permutation in linear time.

## REFERENCES


Bader,D., Moret,B. and Yan,M. (2001) A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. *Proceedings of WADS 2001*, 365–376.

Bafna,V. and Pevzner,P. (1996) Genome rearrangements and sorting by reversals. *SIAM J. Computing*, **25**, 272–289.

Bergeron,A. (2001) A very elementary presentation of the Hannenhalli-Pevzner theory. *Proceedings of CPM 2001*, 106–117.

Bergeron,A., Chauve,C., Hartman,T. and St-Onge,K. (2002) Enumerating optimal sequences of reversals: Tools and experiments. (work in progress).

Hannenhalli,S. and Pevzner,P. (1999) Transforming cabbage into turnip: Polynomial algorithm for sorting signed permutations by reversals. *J. ACM*, **46**, 1–27.

Heber,S. and Stoye,J. (2001) Finding all common intervals of $k$ permutations. *Proceedings of CPM 2001*, 207–218.

Kaplan,H., Shamir,R. and Tarjan,R. (1999) A faster and simpler algorithm for sorting signed permutations by reversals. *SIAM J. Computing*, **29**, 880–892.

Sankoff,D. (1992) Edit distances for genome comparisons based on non-local operations. *Proceedings of CPM 1992*, 121–135.

Siepel,A. (2002) An algorithm to find all sorting reversals. *Proceedings of RECOMB 2002*, 281–290.

Uno,T. and Yagiura,M. (2000) Fast algorithms to enumerate all common intervals of two permutations. *Algorithmica*, **26**, 290–309.


## APPENDIX

The following algorithm uses four stacks to implement the detection of oriented and unoriented components in a signed permutation. Arrays `pi[0..n]` and `si[0..n]` are given as input and store, respectively, the elements and the signs of the permutation. The algorithm prints the annotated list of the positions of the connected components of the permutation.

```
EMPTYSTACK(M1); PUSH(M1,n);
EMPTYSTACK(M2); PUSH(M2,0);
EMPTYSTACK(S1); PUSH(S1,0);
EMPTYSTACK(S2); PUSH(S2,0);

M[0] = n;
m[0] = 0;
max[0] = 0;
min[0] = n;
mark1[0] = FALSE;
mark2[0] = FALSE;

for(i=1; i<=n; i++) {

  //Compute the M_i
  if(pi[i-1] > pi[i]) {
    M[i] = pi[i-1];
    PUSH(M1,pi[i-1]);
  }
  else {
    while(TOP(M1) < pi[i])
      POP(M1);
    M[i] = TOP(M1);
  }

  //Find connected components of type (m ... M)
  while(pi[i]<pi[(s=TOP(S1))] || pi[i]>M[s]) {
    POP(S1);
    max[TOP(S1)] = MAX(max[TOP(S1)],max[s]);
    mark1[TOP(S1)] |= mark1[s];
  }
  if(si[i]==PLUS && pi[i]==max[(s=TOP(S1))]+1 && i-s==pi[i]-pi[s] && i-s>1) {
    if(mark1[s] == FALSE)
      printf("[%d,%d] (unoriented, plus)\n",s,i);
    else
      printf("[%d,%d] (oriented, plus)\n",s,i);
    mark1[TOP(S1)] = FALSE;
  }
```

```
  // And now the "reverse" algorithm

  //Compute the m_i
  if(pi[i-1] < pi[i]) {
    m[i] = pi[i-1];
    PUSH(M2,pi[i-1]);
  }
  else {
    while(TOP(M2) > pi[i])
      POP(M2);
    m[i] = TOP(M2);
  }

  //Find connected components of type (-M ... -m)
  while((pi[i]>pi[(s=TOP(S2))] || pi[i]<m[s]) && s>0) {
    POP(S2);
    min[TOP(S2)] = MIN(min[TOP(S2)],min[s]);
    mark2[TOP(S2)] |= mark2[s];
  }
  if(si[i]==MINUS && pi[i]==min[(s=TOP(S2))]-1 && i-s==pi[s]-pi[i] && i-s>1) {
    if(mark2[s] == FALSE)
      printf("[%d,%d] (unoriented, minus)\n",s,i);
    else
      printf("[%d,%d] (oriented, minus)\n",s,i);
    mark2[TOP(S2)] = FALSE;
  }

  //Update stacks and marks
  if(si[i] == PLUS)
    PUSH(S1,i);
  else
    PUSH(S2,i);
  max[i] = pi[i];
  min[TOP(S2)] = MIN(min[TOP(S2)],pi[i]);
  min[i] = pi[i];
  max[TOP(S1)] = MAX(max[TOP(S1)],pi[i]);
  mark1[TOP(S1)] = si[i]==MINUS;
  mark2[TOP(S2)] = si[i]==PLUS;

}
```