A Very Elementary Presentation of the Hannenhalli-Pevzner Theory

Anne Bergeron

LACIM, Université du Québec à Montréal,
C.P. 8888 Succ. Centre-Ville, Montréal, Québec, Canada, H3C 3P8.
bergeron.anne@uqam.ca

Abstract. In 1995, Hannenhalli and Pevzner gave a first polynomial solution to the problem of finding the minimum number of reversals needed to sort a signed permutation. Their solution, as well as subsequent ones, relies on many intermediary constructions, such as simulations with permutations on $2n$ elements, and manipulation of various graphs.
Here we give the first completely elementary treatment of this problem. We characterize *safe reversals* and *hurdles* working directly on the original signed permutation. Moreover, our presentation leads to polynomial algorithms that can be efficiently implemented using bit-wise operations.

1 Introduction

In the last ten years, beginning with [8], many papers have been devoted to the subject of computing the *reversal distance* between two permutations. A *reversal* $\rho(i,j)$ transforms a permutation

$$\pi = (\ \pi_1 \ldots \underline{\pi_i\ \pi_{i+1}\ \ldots\ \pi_j} \ldots \pi_n\ )$$

$$\text{to}\ \ \pi' = (\ \pi_1 \ldots \pi_j\ \ \ldots\ \ \pi_{i+1}\ \pi_i\ \ldots \pi_n\ ).$$

and the *reversal distance* between two permutations is the minimum number of reversals that transform one into the other.

From a problem of unknown complexity, it graduated to an NP-Hard problem [3], but an interesting variant was proven to be polynomial [5]. In the *signed* version of the problem, each element of the permutation has a plus or minus sign, and a reversal $\rho(i,j)$ transforms π to:

$$\pi' = (\ \pi_1 \ldots -\pi_j \ldots -\pi_{i+1}\ -\pi_i \ldots \pi_n\ ).$$

Permutations, and their reversals, are useful tools in the comparative study of genomes. 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 two (signed) permutations. It is widely accepted that the reversal distance between these two permutations provides a good estimate of the evolutionary distance between the two species.

Computing the reversal distance between signed permutations is a delicate task, since some reversals unexpectedly affect deep structures in permutations. In 1995, Hannenhalli and Pevzner proposed the first polynomial algorithm to solve it [5], developing along the way a theory of how and why some permutations were particularly resistant to sorting by reversals. It is of no surprise that the label *fortress* was assigned to specially acute cases.

Hannenhalli and Pevzner relied on several intermediate constructions that have been simplified since [6,2,1], but grasping all the details remains a challenge. All the criteria given for choosing a *safe* reversal involve the construction of an associate permutation on $2n$ points, and the analysis of cycles and/or connected component of graphs associated with this permutation.

In this paper, we present an elementary treatment of the sorting of the *oriented components* of a permutation, together with a new definition of the concept of *hurdle*, that further simplifies the definition given in [6]. Our first algorithm is so simple that, for example, sorting a permutation of length 20, *by hand*, should be easy and straightforward.

The next section presents the basic algorithms. Section 3 contains the necessary links to the Hannenhalli-Pevzner theory, and the proofs of the claims of the Section 2. Finally, in the last section, we discuss complexity issues, and we give a *bit-vector* implementation of the sorting algorithm that runs in $\mathcal{O}(n^2)$ bit-vector operations.

2 Basic Sorting

The problem of computing the distance between two permutations is often recast as the problem of computing $d(\pi)$, the reversal distance between a permutation π and the identity permutation $(1 \quad 2 \ldots \ n)$. In this paper, we focus on the reconstruction of one possible sequence of reversals that realizes $d(\pi)$, also called the *sorting by reversals problem*. As usual, we will assume that the permutation is *framed* by 0 and $n+1$, and that those extra elements are always positive:

$$\pi = (0 \ \pi_1 \ \pi_2 \ldots \pi_n \ n+1).$$

An *oriented pair* (π_i, π_j) is a pair of consecutive integers, that is $|\pi_i| - |\pi_j| = \pm 1$, with opposite signs. For example, the oriented pairs of the permutation:

$$(\ 0 \ \ 3 \ \ 1 \ \ 6 \ \ 5 \ -2 \ \ 4 \ \ 7\ )$$

are $(1, -2)$ and $(3, -2)$.

Oriented pairs are useful, in the sense that they indicate reversals that create consecutive elements of the permutation that are also consecutive integers. For example, the pair $(1, -2)$ induces the reversal:

$$(\; 0 \quad 3 \quad 1 \quad \underline{6 \quad 5 \quad -2} \quad 4 \quad 7 \;)$$

$$(\; 0 \quad 3 \quad 1 \quad 2 \quad -5 \quad -6 \quad 4 \quad 7 \;),$$

creating the consecutive integers 1 2.

In general, the reversal induced by an oriented pair (π_i, π_j) will be,

$$\begin{array}{ll} \rho(i, j-1), & \text{if } \pi_i + \pi_j = +1, \text{ and} \\ \rho(i+1, j), & \text{if } \pi_i + \pi_j = -1. \end{array}$$

Note that reversals that create consecutive integers are always induced by oriented pairs. Such a reversal is called an *oriented* reversal. We define the *score* of an (oriented) reversal as the number of oriented pairs in the resulting permutation. For example, the score of the reversal:

$$(\; 0 \quad \underline{3 \quad 1 \quad 6 \quad 5} \quad -2 \quad 4 \quad 7 \;)$$

$$(\; 0 \quad -5 \quad -6 \quad -1 \quad -3 \quad -2 \quad 4 \quad 7 \;)$$

is 4, since the resulting permutation has 4 oriented pairs. Computing the score of a reversal is tedious but elementary, and we will discuss efficient algorithms to do so in Section 4. The fact that oriented reversals have a beneficial effect on the ordering of a permutation suggests a first sorting strategy:

Algorithm 1 As long as π has an oriented pair, choose the oriented reversal that has maximal score.

For example, the scores of two oriented pairs $(1, -2)$ and $(3, -2)$ of the permutation:

$$(\; 0 \quad \underline{3 \quad 1 \quad 6 \quad 5} \, -2 \quad 4 \quad 7 \;)$$

are respectively 2 and 4. So we choose the reversal induced by $(3, -2)$, yielding the new permutation:

$$(\; 0 \quad -5 \quad -6 \quad -1 \quad \underline{-3 \quad -2} \quad 4 \quad 7 \;).$$

This permutation has now four oriented pairs $(0, -1)$, $(-3, 4)$, $(-5, 4)$ and $(-6, 7)$, all of which have score 2, except $(-3, 4)$. Acting on this pair yields:

$$(\; 0 \quad -5 \quad -6 \quad \underline{-1} \quad 2 \quad 3 \quad 4 \quad 7 \;).$$

which has four oriented pairs. Note here that the score of the pair $(0, -1)$ is 0. The corresponding oriented reversal would produce a permutation with no oriented pair, and the algorithm would stop, in this case with an unsorted permutation. Fortunately, the pair $(-1, 2)$ has a positive – and maximal – score, and we get, in a similar way, the last two necessary reversals to sort the permutation:

$$(\ 0\ \ -5\ \ \underline{-6\ \ 1\ \ 2\ \ 3\ \ 4}\ \ 7\ )$$

$$(\ 0\ \ \underline{-5\ \ -4\ \ -3\ \ -2\ \ -1}\ \ 6\ \ 7\ )$$

$$(\ 0\ \ 1\ \ 2\ \ 3\ \ 4\ \ 5\ \ 6\ \ 7\ )$$

Interestingly enough, this elementary strategy is sufficient to optimally sort most random permutations and almost all permutations that arise from biological data. The strategy is also optimal, and we will prove in the next section the following claim.

Claim 1: If the strategy of Algorithm 1 applies k reversals to a permutation π, yielding a permutation π', then $d(\pi) = d(\pi') + k$.

The output of Algorithm 1 will be a permutation of positive elements. Most reversal applied to such permutations will create oriented pairs, but the choice of an optimal reversal is delicate. We discuss this problem in the next paragraph.

2.1 Sorting positive permutations

Let π be a signed permutation with only positive elements, and assume that π is *reduced*, that is π does not contain consecutive elements. Suppose also that π is framed by 0 and $n+1$ and consider, as in [6], the circular order induced by setting 0 to be the successor of $n+1$.

Define a *framed interval* in π as an interval of the form:

$$i\ \ \pi_{j+1}\ \pi_{j+2}\ \ldots \pi_{j+k-1}\ \ i+k,$$

such that all integers between i and $i+k$ belong to the interval $[i \ldots i+k]$. For example, consider the permutation:

$$(\ 0\ \ 2\ \ 5\ \ 4\ \ 3\ \ 6\ \ 1\ \ 7\ ).$$

The whole permutation is a framed interval. But we have also the interval: 2 5 4 3 6, which can be reordered as 2 3 4 5 6, and, by circularity, the interval 6 1 7 0 2, which can be reordered as 6 7 0 1 2, since 0 is the successor of 7.

Definition 1 If π is reduced, a *hurdle* in π is a framed interval that contains no shorter framed interval.

Claim 2 Hurdles as defined in Definition 1 are the same hurdles that are defined in [5] and [6].

When a permutation has only one or two hurdles, one reversal is sufficient to create enough oriented pairs in order to completely sort the permutation with

Algorithm 1. Two operations are introduced in [5], the first one is *hurdle cutting* which consist in reversing the segment between i and $i+1$ of a hurdle:

$$i \; \underline{\pi_{j+1} \; \pi_{j+2} \; \ldots} \; i+1 \; \ldots \; \pi_{j+k-1} \; i+k.$$

This reversal is sufficient to sort all the interval using Algorithm 1. For example, the following permutation contains only one hurdle:

$$(\; 0 \quad 2 \quad 4 \quad 3 \quad 1 \quad 5 \;).$$

The reversal of elements 2, 4 and 3 cuts the hurdle, and the resulting permutation,

$$(\; 0 \;\; -3 \;\; -4 \; -2 \quad 1 \quad 5 \;),$$

can be sorted with 4 reversals by Algorithm 1.

The second operation is *hurdle merging*, which acts on the end points of two hurdles:

$$i \ldots \underline{i+k \ldots i'} \ldots i' + k',$$

and does the reversal $\rho(i+k, i')$. If a permutation has only two hurdles, merging them will produce a permutation that can be completely sorted by Algorithm 1.

Thus, for example, merging the two hurdles in the permutation:

$$(\; 0 \quad 2 \quad 5 \quad 4 \quad 3 \quad 6 \quad 1 \quad 7 \;),$$

yields the permutation:

$$(\; 0 \quad 2 \quad 5 \quad 4 \quad 3 \;\; -6 \quad 1 \quad 7 \;),$$

which can be sorted in 5 reversals using Algorithm 1.

Merging and cutting hurdles in a permutation that contains more than 2 hurdles must be managed carefully. Indeed, cutting some hurdles can create new ones!

Definition 2 A *simple* hurdle is a hurdle whose cutting decreases the number of hurdles. Hurdles that are not simple are called *super hurdles.*

For example, the permutation $(\; 0 \quad 2 \quad 5 \quad 4 \quad 3 \quad 6 \quad 1 \quad 7 \;)$ has two hurdles. Cutting and sorting the hurdle 2 5 4 3 6 yields the permutation,

$$(\; 0 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 1 \quad 7 \;),$$

which, by collapsing the sequence 2 3 4 5 6, reduces to:

$$(\; 0 \quad 2 \quad 1 \quad 3 \;),$$

which has only one hurdle.

However, the permutation $(\; 0 \quad 2 \quad 4 \quad 3 \quad 5 \quad 1 \quad 6 \quad 8 \quad 7 \quad 9)$ also contains two hurdles, and when one cuts the hurdle 2 4 3 5, the resulting reduced permutation is:

$$(\; 0 \quad 2 \quad 1 \quad 3 \quad 5 \quad 4 \quad 6 \;),$$

which still has two hurdles.

The following algorithm is adapted from [6], and is discussed originally in [5].

Algorithm 2 If a permutation has $2k$ hurdles, $k \geq 2$, merge any two non-consecutive hurdles. If a permutation has $2k+1$, $k \geq 1$, then if it has one simple hurdle, cut it; If it has none, merge two non-consecutive hurdles, or consecutive ones if $k = 1$.

Together with Algorithm 1, Algorithm 2 can be used to optimally sort any signed permutation. This completes the first part of the paper, and, in the next section, we turn to the task of proving our various claims.

3 Selected Results from the Hannenhalli-Pevzner Theory

The exposition of the complete results of the Hannenhalli-Pevzner theory is beyond the scope of this paper, and the reader is referred to the original paper [5], or the book on computational molecular biology by Pevzner [7]. Instead, we will show the soundness of our algorithms by directly using the *arc overlap graph* introduced in [6].

The first construction is the *breakpoint graph* associated with π. Each positive element x in the permutation π is replaced by the sequence $2x-1\ 2x$, and each negative element $-x$ by the sequence $2x\ 2x-1$. For example,

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

becomes:

$$\pi' = (\ 0\ \ 2\ \ 1\ \ 5\ \ 6\ \ 9\ \ 10\ \ 7\ \ 8\ \ 11\ \ 12\ \ 4\ \ 3\ \ 13\ )$$

Reversals $\rho(i,j)$ of π are simulated by unsigned reversals $\rho(2i-1, 2j)$ in π'.

The elements of π' are the vertices of the breakpoint graph. Straight edges join every other pair of consecutive elements of π', starting with 0, and curved edges, called *arcs*, join every other pair of consecutive integers, starting with $(0,1)$.

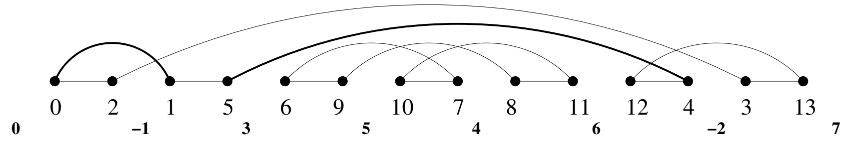


Fig. 1. The Breakpoint Graph of $\pi = (0\ -1\ 3\ 5\ 4\ 6\ -2\ 7)$

Every connected component of the breakpoint graph is a cycle, which is a consequence of the fact that each vertex has exactly two incident edges. The graph of Fig. 1 has 3 cycles.

The *support* of an arc is the interval of elements of π' between, and including, its endpoints. Two arcs *overlap* if their support intersect, without proper containment. An arc is *oriented* if its support contains an odd number of elements, otherwise it is *unoriented.* In Fig. 1, the oriented arcs are $(0,1)$ and $(4,5)$. Note that an arc is oriented if and only if its endpoints are images of an oriented pair of the original permutation.

The *arc overlap graph* is the graph whose vertices are arcs of the cycle graph, and whose edges join overlapping arcs. The overlap graph corresponding to the breakpoint graph of Fig. 1 is illustrated in Fig 2, in which each vertex is labeled by an arc $(2x, 2x+1)$. *Oriented* vertices – those for which the corresponding arc is oriented – are marked by black dots. Orientation extends to connected component in the sense that a connected component with at least one oriented vertex is oriented. It is easy to show that a vertex is oriented if and only if its degree is odd.

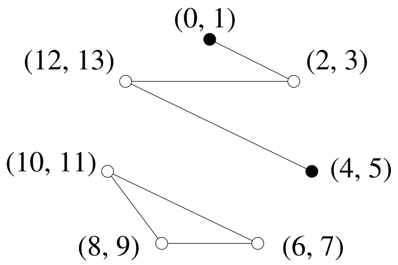


Fig. 2. The Arc Overlap Graph of $\pi = (0\ \ -1\ \ 3\ \ 5\ \ 4\ \ 6\ \ -2\ \ 7)$

There is a natural bijection between the vertices of the overlap graph and pairs of (unsigned) consecutive integers x and $x+1$ in the original permutation. Indeed, a pair of consecutive integers will generate four consecutive integers in the unsigned permutation: $2x-1$, $2x$, $2x+1$, and $2x+2$. The vertex $(2x, 2x+1)$ is associated with the pair x and $x+1$. For example, the oriented pair $(3, -2)$ in π corresponds to the vertex $(4,5)$ in the overlap graph. We will refer to the *reversal induced by a vertex* meaning the reversal induced by the corresponding oriented pair of the original permutation. The following facts, mostly from [6], pinpoint the important relations between a signed permutation and its overlap graph.

Fact 1: *A vertex has an odd degree if and only if it is oriented.*

Proof. Let $2x-1$, $2x$, $2x+1$, and $2x+2$, be the four integers associated with the oriented pair (π_i, π_j). Since π_i and π_j have different signs, the positions of $2x$ and $2x+1$ will not have the same parity in the unsigned permutations. Thus, the interval between $2x$ and $2x+1$ has an odd length, implying that it overlaps an odd number of other intervals. On the other hand, any interval that overlaps an odd number of intervals must have an odd length. Therefore the positions of its endpoints must have different parities, implying that the corresponding pair of consecutive integers is oriented. ∎

Fact 2: *If one performs the reversal corresponding to an oriented vertex v, the effect on the overlap graph will be to complement the subgraph of v and its adjacent vertices.*

Proof. The reversal corresponding to an oriented vertex v has the effect of collapsing the associated interval, thus v will become isolated. Let u and w be two intervals overlapping v, meaning that exactly one of their end points lies in the interval spanned by v. The reversal induced by v will reverse these two points. Here, a picture is worth a thousand words:

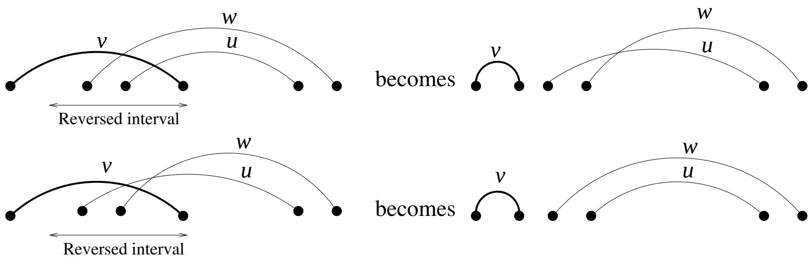


Fig. 3. Complementation of Sub-Graphs

∎

Fact 3: *If one performs the reversal corresponding to an oriented vertex v, each vertex adjacent to v will change its orientation.*

Proof. Since v is oriented, it has an odd number $2k+1$ of adjacent vertices. Let w be a vertex adjacent to v, with j neighbors also adjacent to v. With the reversal, w will loose $j+1$ neighbors, and gain $2k-j$ new ones. Thus the degree of w will change by $2k-2j-1$, changing its orientation. ∎

Fact 4: *The score of the oriented reversal corresponding to an oriented vertex v is given by:*

$$T + U - O - 1$$

where T is the total number of oriented vertices in the graph, U is the number of unoriented vertices adjacent to v, and O is the number of oriented vertices adjacent to v.

Proof. This follows trivially from the preceding facts. ∎

We now state a basic result that is proved, in different ways, in [5] and [6]. Define an *oriented component* of the overlap graph as a connected component that contains at least one oriented vertex, otherwise the component is *unoriented*.

A *safe* reversal is a reversal that does not create new unoriented components, except for isolated vertices.

Theorem 1 (Hannenhalli and Pevzner, 1995). *Any sequence of oriented safe reversals is optimal.*

The difficulties in sorting oriented components lie in the detection of safe reversals. Hannenhalli and Pevzner deal with the problem by computing several statistics on cycles and breakpoints of various graphs. Kaplan et al. [6] solve it by searching for particular cliques in the overlap graph. The next theorem argues that the elementary strategy of choosing the reversal with maximal score is optimal, thus proving Claim 1.

Theorem 2. *An oriented reversal of maximal score is safe.*

Proof. Suppose that vertex v has maximal score, and that the reversal induced by v creates a new unoriented component C containing more than one vertex. At least one of the vertices in C must have been adjacent to v, since the only edges affected by the reversal are those between vertices adjacent to v. Let w be a vertex formerly adjacent to v and contained in C, and consider the scores of v and w:

$$score(v) = T + U - O - 1$$
$$score(w) = T + U' - O' - 1$$

All unoriented vertices formerly adjacent to v must have been adjacent to w. Indeed, an unoriented vertex adjacent to v and not to w will become oriented, and connected to w, contrary to the assumption that C is unoriented. Thus, $U' \geq U$.

All oriented vertices formerly adjacent to w must have been adjacent to v. If this was not the case, an oriented vertex adjacent to w but not to v would remain oriented, again contradicting the fact that C is unoriented. Thus, $O' \leq O$.

Now, if both $O' = O$ and $U' = U$, vertices v and w have the same set of adjacent vertices, and complementing the subgraph of v and its adjacent vertices will isolate both v and w. Therefore, we must have that $score(w) > score(v)$, which is a contradiction. ■

3.1 Hurdles

In this section, we assume that π is a positive and reduced permutation. These assumptions are equivalent to say that the overlap graph has no oriented components – all of which can be cleared by Algorithm 1 –, and no isolated vertices.

Consider again the circular order, this time on the interval $[0..2n-1]$, induced by setting 0 to be the successor of $2n-1$. The *span* of a set of vertices X in the overlap graph is the minimum interval that contains, in the circular order, all the intervals of vertices in X. For example, consider the permutation:

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

whose breakpoint graph and arc overlap graphs are illustrated in Fig. 4 and Fig. 5. The three connected components of its arc overlap graph have spans:

$$
\begin{aligned}
&[4,15] = [4\ 7\ 8\ 11\ 12\ 9\ 10\ 13\ 14\ 5\ 6\ 15] \\
&[8,13] = [8\ 11\ 12\ 9\ 10\ 13] \\
&[16,3] = [16\ 1\ 2\ 17\ 0\ 3].
\end{aligned}
$$

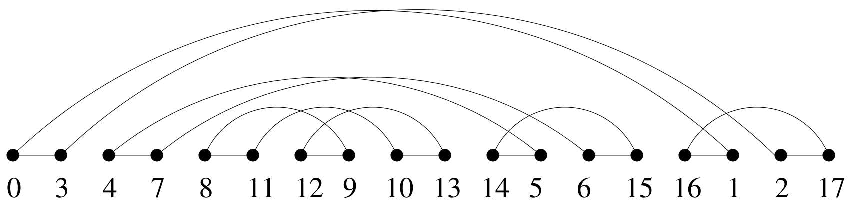


Fig. 4. The Breakpoint Graph of $\pi = (0\ \ 2\ \ 4\ \ 6\ \ 5\ \ 7\ \ 3\ \ 8\ \ 1\ \ 9)$

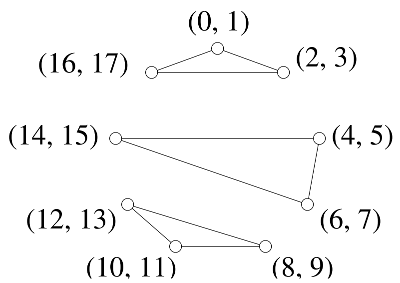


Fig. 5. The Arc Overlap Graph of $\pi = (0\ \ 2\ \ 4\ \ 6\ \ 5\ \ 7\ \ 3\ \ 8\ \ 1\ \ 9)$

Hurdles are defined in [5] as unoriented components which are minimal with respect to span inclusion. Moreover, in [6], it is shown that the span of a connected component is always of the form $[2i, 2j-1]$. The following Lemmas and Theorem detail the relationships between connected components and framed intervals, substantiating the second claim of Section 1.

Lemma 1. *Framed intervals of the form $[i, j]$ in a permutation on n elements are in one-to-one correspondence with framed intervals of the form $[2i, 2j-1]$ in the corresponding unsigned permutation on $2n$ elements.*

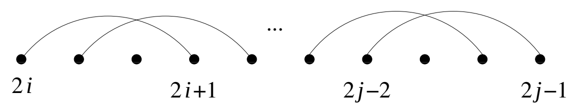
Proof. The end points i and j of a framed interval $[i, j]$ will be mapped, respectively, to the pairs $2i-1$, $2i$, and $2j-1$, $2j$. All integers between i and j appear in the interval $[i, j]$, if and only if all the integers between $2i$ and $2j-1$ appear in the interval $[2i, 2j-1]$.

Lemma 2. *Any framed interval* $[2i, 2j-1]$ *is the span of a union of connected components.*

Proof. If $[2i, 2j-1]$ is a framed interval, it contains exactly the integers between $2i$ and $2j-1$, thus the only arcs in this interval are: $(2i, 2i+1)$, $(2i+2, 2i+3), \ldots, (2j-2, 2j-1)$, and no other arc intersects this set. Therefore, the corresponding set of vertices is not connected to any other vertex. ∎

Lemma 3. *The span* $[2i, 2j-1]$ *of a connected component is always a framed interval.*

Proof. If vertex $(2i, 2i+1)$ is connected to $(2j-2, 2j-1)$, there must be a sequence of intersecting arcs linking $2i$ to $2j-1$.



Any arc with only one end point between $2i$ and $2j-1$ would therefore intersect one of the arcs in the sequence, so there are none. Thus, if integer $2k$ is in the interval, then $2k+1$ is also in the interval, and if $i \leq k < j$, then $2k+2$ is also in the interval. ∎

Theorem 3. *If* π *is reduced, an unoriented component is minimal if and only if its span is a framed interval that contains no other.*

Proof. By Lemma 3, the span of a connected component is always a framed interval. If the component is minimal with respect to span inclusion, by Lemma 2, its span cannot contain properly another framed interval.

On the other hand, a framed interval $[2i, 2j-1]$ that contains no other yields a single connected component C whose vertices endpoints are exactly the integers between $2i$ and $2j-1$. Thus the vertices of C are consecutive on the circle, and component C is minimal. ∎

The main consequence of Theorem 3 is to give an elementary characterization of the concept of hurdles, that does not rely on the construction of the overlap graph.

4 Settling Scores

We now turn to the problem of computing efficiently the score of a reversal. Performing a reversal, or computing its score, are "local" operations on the graph, and this locality suggests the possibility of a parallel algorithm to keep the scores and to compute the effects of a reversal.

The parallelism of the algorithm will exploit the inherent parallelism of basic operations of a processor. We will work with bit-vectors, and use only three operations on these vectors: the exclusive-or operator $\oplus$; the conjunction $\wedge$; and the negation $\neg$.

Subsets of vertices of the arc overlap graph will be represented by *characteristics* bit-vectors: If s is a subset of the $\{0, \ldots, n\}$, then the bold symbol $\boldsymbol{s}$ is the bit-vector:

$$\boldsymbol{s} = (\boldsymbol{s}_0, \ldots \boldsymbol{s}_n) \text{ where } s_i = \begin{cases} 1 \text{ if } i \in s, \\ 0 \text{ otherwise.} \end{cases}$$

4.1 The Data Structure

Given an arc overlap graph, we first construct a bit-matrix in which each line $\boldsymbol{v_i}$ is the set of adjacent vertices to arc $(2i, 2i+1)$. For example, consider the permutation $(0\ \ 3\ \ 1\ \ 6\ \ 5\ \ -2\ \ 4\ \ 7)$, whose arc overlap graph is illustrated in Fig. 6.

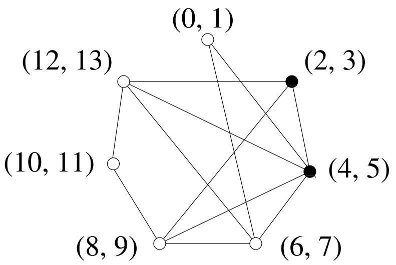


Fig. 6. The Arc Overlap Graph of $(0\ \ 3\ \ 1\ \ 6\ \ 5\ \ -2\ \ 4\ \ 7)$

The bit-matrix associated to this graph is the following:

	$\boldsymbol{v_0}$	$\boldsymbol{v_1}$	$\boldsymbol{v_2}$	$\boldsymbol{v_3}$	$\boldsymbol{v_4}$	$\boldsymbol{v_5}$	$\boldsymbol{v_6}$
$\boldsymbol{v_0}$	0	0	1	1	0	0	0
$\boldsymbol{v_1}$	0	0	1	0	1	0	1
$\boldsymbol{v_2}$	1	1	0	1	1	0	1
$\boldsymbol{v_3}$	1	0	1	0	1	0	1
$\boldsymbol{v_4}$	0	1	1	1	0	1	0
$\boldsymbol{v_5}$	0	0	0	0	1	0	1
$\boldsymbol{v_6}$	0	1	1	1	0	1	0
$\boldsymbol{p}$	0	1	1	0	0	0	0
$\boldsymbol{s}$	0	1	3	2	0	2	0

The last two lines contain, respectively, the parity $\boldsymbol{p}$, or orientation, of the vertex, and the score $\boldsymbol{s}$ of the associated reversal. We will discuss efficient ways to initialize the structure and to adjust scores in Sections 4.2 and 4.3.

Given the vectors $\boldsymbol{p}$ and $\boldsymbol{s}$, selecting the oriented reversal with maximal score is elementary. In the above example, vertex 2 would be the selected candidate.

The interesting part is how a reversal affects the structure. These effects are summarized in the following algorithm, which recalculates the bit-matrix $\boldsymbol{v}$, the parity vector $\boldsymbol{p}$, and the score vector $\boldsymbol{s}$, following the reversal associated to vertex i, whose set of adjacent vertices is denoted by $\boldsymbol{v_i}$.

1. $\boldsymbol{s} \leftarrow \boldsymbol{s} + \boldsymbol{v_i}$
2. $\boldsymbol{v_i}_i \leftarrow 1$
3. For each vertex j adjacent to i
4.   If j is oriented
5. $\boldsymbol{s} \leftarrow \boldsymbol{s} + \boldsymbol{v_j}$
6. $\boldsymbol{v_j}_j \leftarrow 1$
7. $\boldsymbol{v_j} \leftarrow \boldsymbol{v_j} \oplus \boldsymbol{v_i}$
8. $\boldsymbol{s} \leftarrow \boldsymbol{s} + \boldsymbol{v_j}$
9.   Else
10. $\boldsymbol{s} \leftarrow \boldsymbol{s} - \boldsymbol{v_j}$
11. $\boldsymbol{v_j}_j \leftarrow 1$
12. $\boldsymbol{v_j} \leftarrow \boldsymbol{v_j} \oplus \boldsymbol{v_i}$
13. $\boldsymbol{s} \leftarrow \boldsymbol{s} - \boldsymbol{v_j}$
14. $\boldsymbol{p} \leftarrow \boldsymbol{p} \oplus \boldsymbol{v_i}$

The logic behind the algorithm is the following. Since vertex i will become unoriented and isolated, each vertex adjacent to i will automatically gain a point of score in step 1. Next, if j is a vertex adjacent to i, vertices adjacent to j after the reversal are either existing vertices that were not adjacent to i, or vertices that were adjacent to i but not to j. This is the definition of the exclusive-or operator $\oplus$.

The exceptions to this rule are i and j themselves, and this problem is solved by setting the diagonal bits to 1 before computing $\boldsymbol{v_j} \oplus \boldsymbol{v_i}$ in step 6.

If j is oriented, each of its former adjacent vertices will gain one point of score, since j will become unoriented, and each of its new adjacent vertices will gain one point of score. We thus add these points of score with two instructions sandwiching step 6. Note that a vertex that *stays* connected to w will gain a total of two points. For unoriented vertices, the gains are converted to losses.

The amount of work done to process a reversal corresponding to vertex i, in terms of vector operations, is thus proportional to the number of adjacent vertices to vertex i.

4.2 Representing the Scores

The additions and subtractions to adjust the score vector are the usual arithmetic operations performed component-wise. In order to have a truly bit-vector implementation, we represented the score vector as a $\lceil \log(n) \rceil \times n$ bit-matrix, each column containing the binary representation of a score. With this representation, component-wise addition of a bit-vector $\boldsymbol{v}$ to $\boldsymbol{s}$ can be realized with the following:

$$
\begin{array}{l}
\text{For } k \text{ from 1 to } \lceil \log(n) \rceil \\
\quad \boldsymbol{t} \ \leftarrow \boldsymbol{v} \\
\quad \boldsymbol{v} \ \leftarrow \boldsymbol{v} \wedge \boldsymbol{s_k} \\
\quad \boldsymbol{s_k} \leftarrow \boldsymbol{t} \oplus \boldsymbol{s_k}
\end{array}
$$

Subtraction is implemented in a similar way. A side benefit of this structure is that the selection of the next reversal can be also done in parallel, by "sifting" the score matrix through the parity vector. The set $\boldsymbol{c}$ of candidates contains initially all the oriented vertices. Going from the higher bit of scores to the lower, if at least one of the candidates has bit i set to 1, we eliminate all candidates for which bit i is 0.

$$
\begin{array}{l}
\boldsymbol{c} \leftarrow \boldsymbol{p} \\
i \leftarrow \lceil \log(n) \rceil \\
\text{While } i \geq 0 \text{ do} \\
\quad \text{While } (\boldsymbol{c} \wedge \boldsymbol{s_i}) = 0 \\
\quad\quad i \leftarrow i - 1 \\
\quad \text{If } i \geq 0 \\
\quad\quad \boldsymbol{c} \leftarrow \boldsymbol{c} \wedge \boldsymbol{s_i} \\
\quad\quad i \leftarrow i - 1
\end{array}
$$

At the end of the loop, $\boldsymbol{c}$ is the set of oriented vertices of maximal score.

4.3 Initializing the Data Structure

We saw, in Section 3, that the overlap graph of a signed permutation $\pi = (\pi_1\ \pi_2 \ldots \pi_n)$ contains $n+1$ vertices corresponding to the arcs joining $2x$ and $2x+1$ in the equivalent unsigned permutation. In this section, we will construct a representation of the overlap graph without explicitly referring to the unsigned permutation, thus removing one more step between the actual algorithm, and the original formulation of the problem.

The construction is based on the following simple lemma. Let I be a set of intervals with distinct endpoints in an ordered set S. Denote an interval $i \in I$ by the ordered pair (b_i, e_i) of its endpoints. Define the sets $\boldsymbol{l_i}$ and $\boldsymbol{r_i}$ as follows:

$$
\begin{aligned}
\boldsymbol{r_i} &= \{ j \in I \mid b_j < e_i < e_j \} \\
\boldsymbol{l_i} &= \{ j \in I \mid b_j < b_i < e_j \}
\end{aligned}
$$

The set $\boldsymbol{r_i}$ is the set of intervals j in I that contains the right endpoint of interval i, and the set $\boldsymbol{l_i}$ is the set of intervals j in I that contains the left endpoint of interval i. We have the following:

Lemma 4. *The set $\boldsymbol{v_i}$ of intervals that overlap i in I is given by: $\boldsymbol{v_i} = \boldsymbol{l_i} \oplus \boldsymbol{r_i}$.* ■

Starting with a signed permutation $\pi = (\pi_1\ \pi_2 \ldots \pi_n)$, we first read the elements from left to right. Let $\boldsymbol{a}$ represent the set of arcs for which exactly one endpoint has been read. Initially, $\boldsymbol{a}$ is the set $\{0\}$, corresponding to the arc $(0, 1)$. When element π_i is read, we have to process two arcs: $(2\pi_i - 2, 2\pi_i - 1)$ and $(2\pi_i, 2\pi_i + 1)$. In increasing order, if π_i is positive, and decreasing order, otherwise. Processing an arc $(2j, 2j + 1)$ is done by the following instructions:

```
If a_j = 0
   Then a_j ← 1   (* First endpoint of arc (2j, 2j + 1) *)
   Else           (* Second endpoint *)
      a_j ← 0
      v_j ← a     (* a is the set r_j *)
```

We then repeat the process in the reverse order, reading the permutation from right to left, initializing $\boldsymbol{a}$ to the set {n}, and changing the last instruction to $\boldsymbol{v_j} \leftarrow \boldsymbol{v_j} \oplus \boldsymbol{a}$.

4.4 Analysis

The formal analysis of the algorithm of Section 4.1 raises interesting questions. For example, what is an elementary operation? Except for a few control statements, the only operations used by the algorithm are very efficient bit-wise logical operators on words of size w – typically 32 or 64, depending on implementation. The most expensive instructions in the main loop are additions and subtractions, such as

$$\boldsymbol{s} \leftarrow \boldsymbol{s} + \boldsymbol{v_j},$$

where $\boldsymbol{s}$ is a bit matrix of size $n\log(n)$, and $\boldsymbol{v_j}$ is a bit vector of size n. Such an operation requires a total of $(2n\log(n))/w$ elementary operations with the loop described in Section 4.2. Hopefully, $\log(n)$ is much smaller than w, and, in the range of biologically meaningful values, n is often a small multiple of w. In the actual implementation, the loop is controlled by the value of log(maximal score) which tends to be much less than $\log(n)$. We thus have a, very generous, $\mathcal{O}(n)$ estimate for the instructions in the main loop.

The overall work done by the algorithm depends on the total number v of vertices adjacent to vertices of maximal score. We can easily bound it by n^2, noting that the number d of reversals needed to sort the permutation is bounded by n, and the degree of a vertex is also bounded by by n. We thus get an $\mathcal{O}(n^3)$ estimate for the algorithm, assuming that $\log n < w$. However, in practical applications, such as [4], this algorithm outperforms consistently classical algorithms with smaller theoretical complexity.

References

1. Bader, D., Moret, B., Yan, M. (2001). *A Linear-Time Algorithm for Computing Inversion Distance Between Signed Permutations with an Experimental Study.* J. Comput. Biol. 8, 5: 483-491.

2. Piotr Berman, Sridhar Hannenhalli, *Fast Sorting by Reversal.* CPM 1996, LNCS 1075: 168-185.
3. Alberto Caprara, *Sorting by reversals is difficult.* RECOMB 1997, ACM Press: 75-83.
4. Adam Siepel, *An algorithm to find all sorting reversals.* To appear in *RECOMB'02*, 2002.
5. Sridhar Hannenhalli, Pavel A. Pevzner, *Transforming Cabbage into Turnip: Polynomial Algorithm for Sorting Signed Permutations by Reversals.* JACM 46(1): 1-27 (1999).
6. Haim Kaplan, Ron Shamir, Robert Tarjan, *A Faster and Simpler Algorithm for Sorting Signed Permutations by Reversals.* SIAM J. Comput. 29(3): 880-892 (1999).
7. Pavel Pevzner, Computational Molecular Biology, MIT Press, Cambridge, Mass., 314 p., (2000).
8. David Sankoff, *Edit Distances for Genome Comparisons Based on Non-Local Operations.* CPM 1992, LNCS 644,: 121-135, (1992)