Reversal distance without hurdles and fortresses

Anne Bergeron[1], Julia Mixtacki[2], and Jens Stoye[3]

[1] LaCIM, Université du Québec à Montréal, Canada.
[2] Fakultät für Mathematik, Universität Bielefeld, Germany.
[3] Technische Fakultät, Universität Bielefeld, Germany.

Abstract. This paper presents an elementary proof of the Hannenhalli-Pevzner theorem on the reversal distance of two signed permutations. It uses a single PQ-tree to encode the various features of a permutation. The parameters called *hurdles* and *fortress* are replaced by a single one, whose value is computed by a simple and efficient algorithm.

1 Introduction

Computing the reversal distance of two signed permutations is a delicate task since some reversals unexpectedly affect deep structures in permutations. In 1995, Hannenhalli and Pevzner proposed the first polynomial-time algorithm to solve this problem [7], developing along the way a theory of how and why some permutations were particularly resistant to sorting by reversals.

Hannenhalli and Pevzner relied on several intermediate constructions that have been subsequently simplified [8], [10], but grasping all the details remained a challenge. Before [1], all the criteria given for choosing a *safe* reversal involved the construction of an associate permutation on $2n$ points, and the analysis of cycles and/or connected components of graphs associated to this permutation.

Another puzzling aspect of the Hannenhalli-Pevzner theory is the complex, but always colorful, classification of *hurdles*. In this paper, we show that simple results on trees are at the root of all results on hurdles, either maximal or simple, super-hurdles, and fortresses. We give an elementary proof of the Hannenhalli-Pevzner *duality theorem* in terms of a PQ-tree associated to the permutation, yielding efficient and simple algorithms to compute the reversal distance.

The next section presents classical material and results in a simpler form, and describes the tree associated to the permutation. Section 3 gives a new proof and formula for the Hannenhalli-Pevzner theorem, and Section 4 discusses the algorithms.

2 Background

A *signed permutation* is a permutation on the set of integers $\{0, 1, 2, \ldots, n\}$ in which each element has a sign. We also assume that all permutations begin with 0 and end with n, for example: $P_1 = (0\ -2\ -1\ \ 4\ \ 3\ \ 5\ -8\ \ 6\ \ 7\ \ 9)$. A *point* $p \cdot q$ is defined by a pair of consecutive elements in the permutation. For example, $0 \cdot -2$

and $-2 \cdot -1$ are the first two points of P_1. When a point is of the form $i \cdot i+1$, or $-(i+1) \cdot -i$, it is called an *adjacency*, otherwise it is called a *breakpoint.* For example, P_1 has two adjacencies, $-2 \cdot -1$ and $6 \cdot 7$. All other points of P_1 are breakpoints.

We will make an extensive use of intervals of consecutive elements in a permutation. An interval is easily defined by giving its *endpoints.* The *elements* of the interval are the elements between the two endpoints. When the two endpoints are equal, the interval contains no elements. A non-empty interval can also be specified by giving its first and last element, such as $(i..j)$, called the *bounding elements* of the interval.

A *reversal* of an interval of a signed permutation is the operation that consists of reversing the order of the elements of the interval, while changing their signs. The reversal of an interval modifies the points of a signed permutation in various ways. Points $p \cdot q$ that are inside the interval are transformed to $-q \cdot -p$, the endpoints of the interval exchange their flanking elements, and points that are outside the interval are unaffected.

The *reversal distance* $d(P)$ of a permutation P is the minimum number of reversals needed to transform P into the identity permutation.

2.1 Elementary intervals and cycles

Let P be a signed permutation on the set $\{0, 1, 2, \ldots, n\}$ that begins with 0 and ends with n. When sign is irrelevant, we will refer to an element as an *unsigned element.* The *right*, or *left*, point of an element of P is the point immediately to its right, or left.

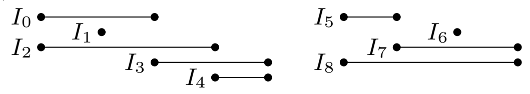
Definition 1. *For each pair of unsigned elements $(k, k+1)$, $0 \leq k < n$, define the* elementary interval I_k *associated to the pair to be the interval whose endpoints are:*

1) The right point of k, if k is positive, otherwise its left point.
2) The left point of $k+1$, if $k+1$ is positive, otherwise its right point.

Elements k and $k+1$ are called the extremities *of the elementary interval.*

Note that an elementary interval can contain zero, one, or both of its extremities. For example, in the following permutation, interval I_0 contains one of its extremities, interval I_3 contains both, and interval I_5 contains none.

(0 -2 -1 4 3 5 -8 6 7 9)



Note that empty elementary intervals correspond to the adjacencies in the permutation.

When the extremities of an elementary interval have different signs, the interval is said to be *oriented*, otherwise it is *unoriented.* Oriented intervals play a basic role in the problem of sorting by reversals since they can be used to create adjacencies. Namely, we have:

Proposition 1. *Reversing an oriented interval I_k creates, in the resulting permutation, either the adjacency $k \cdot k+1$ or the adjacency $-(k+1) \cdot -k$.*

When a point is the endpoint of two elementary intervals, these are said to *meet* at that point. Definition 1 implies that a point is used at most twice as an endpoint, and since there are as many non-empty elementary intervals as there are breakpoints, we have:

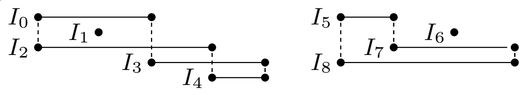
Proposition 2. *Exactly two elementary intervals meet at each breakpoint of a permutation.*

Therefore, starting from an arbitrary point, one can follow elementary intervals on a unique path that eventually comes back to the original point. More formally:

Definition 2. *A* cycle *is a sequence $b_1, b_2, \ldots, b_k$ of points such that two successive points are the endpoints of an elementary interval, including b_k and b_1.*

For example, the following permutation has four cycles, two of them are adjacencies, and the other two contain, respectively, 4 and 3 breakpoints.

(0 -2 -1 4 3 5 -8 6 7 9)



One of the cornerstones of the sorting by reversals problem is to study the effects of a reversal on elementary intervals and cycles. The following result, due to [9], quantifies the effect of a reversal on the number of cycles. It is a consequence of the fact that, for all points except the endpoints of a reversal, the elementary intervals that meet at those points will meet at the same points after the reversal.

Proposition 3. *A reversal modifies the number of cycles by $+1$, 0, or -1.*

Finally, note that the identity permutation on the set $\{0, 1, 2, \ldots, n\}$ is the only one with n cycles. Thus, Proposition 3 implies that $d(P) \geq n - c$, where c is the number of cycles of P.

2.2 Components

Elementary intervals and cycles are organized in higher structures called *components*. These were first identified in [5] as *subpermutations* since they are intervals that contain a permutation of a set of consecutive integers.

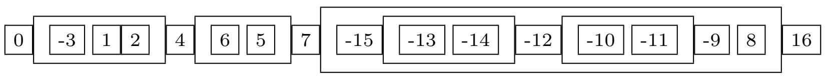
Definition 3. *Let P be a signed permutation on the set $\{0, 1, 2, \ldots, n\}$. A* component *of P is an interval from i to $(i+j)$ or from $-(i+j)$ to $-i$, for some $j > 0$, whose set of unsigned elements is $\{i, \ldots, i+j\}$, and that is not the union of two such intervals. Components with positive – respectively negative – bounding elements are referred to as* direct *– respectively* reversed *– components.*

For example, consider the permutation:

$$P_2 = (0\ {-3}\ 1\ 2\ 4\ 6\ 5\ 7\ {-15}\ {-13}\ {-14}\ {-12}\ {-10}\ {-11}\ {-9}\ 8\ 16).$$

It has 6 components, one of them being the interval $(0..4)$, which contains all unsigned elements between 0 and 4; another is the interval $(-15..-12)$. Note that a component, such as the interval $(1..2)$, can contain only two elements.

Components of a permutation can be represented by the following diagram, in which the bounding elements of each component have been boxed, and the elements between them are enclosed in a rectangle. Elements which are not bounding elements of any component are also boxed.



Proposition 4 ([3]). *Two different components of a permutation are either disjoint, nested with different endpoints, or overlapping on one element.*

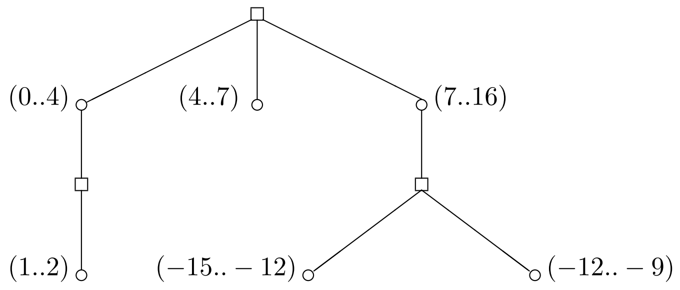
When two components overlap on one element, we say that they are *linked.* Successive linked components form a *chain.* A chain that cannot be extended to the left or right is called *maximal.* Note that a maximal chain may consist of only a single component. If one component of a chain is nested in a component A, then all other components of the chain are also nested in A.

The nesting and linking relations between components turn out to play a major role in the sorting by reversal problem. Another way of representing these relations is in form of the following tree:

Definition 4. *Given a permutation P on the set $\{0, 1, \ldots, n\}$ and its components, define the tree T_P by the following construction:*

1) *Each component is represented by a round node.*
2) *Each maximal chain is represented by a square node whose (ordered) children are the round nodes that represent the components of this chain.*
3) *A square node is the child of the smallest component that contains this chain.*

For example, the tree associated to permutation P_2 is:



It is easy to see that, if the permutation begins with 0 and ends with n, the resulting graph is a single tree with a square node as root. The tree is similar to the PQ-tree used in different context such as the consecutive ones test [4]. The following properties of paths in T_P are elementary consequences of the definition of T_P.

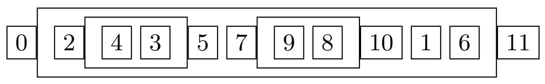
Proposition 5. *Let C be a component on the (unique) path joining components A and B, then C contains either A or B, or both.*

1) If C contains both A and B, it is unique.
2) If no component on the path contains both A and B, then A and B are included in two components that are in the same chain.

Components organize hierarchically the points of a permutation.

Definition 5. *A point $p \cdot q$* belongs *to the smallest component that contains both p and q.*

Note that this does not prevent p and q to be contained, separately, in smaller components, such as 5 and 7 in:



Proposition 6. *The endpoints of an elementary interval belong to the same component, thus all the points of a cycle belong to the same component.*

Proof: Consider an elementary interval I_k and any component C of the form

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

such that $i \leq k < i+j$. Then both endpoints of I_k are contained in C. This is obvious if k is different from i and $k+1$ is different from $i+j$, since both k and $k+1$ will be in the interior of the component. If $k = i$, then k and i have the same sign, and the first endpoint of I_k belongs to the component. If $k+1 = i+j$, then $k+1$ and $i+j$ have the same sign, and the second endpoint of I_k belongs to the component.

Thus endpoints of I_k are either both contained, or not, in any given component, and the result follows. ■

Finally, components can be classified according to the nature of the points they contain:

Definition 6. *The* sign *of a point $p \cdot q$ is positive if both p and q are positive, it is negative if both are negative. A component is* unoriented *if it has one or more breakpoints and all of them have the same sign. Otherwise the component is* oriented.

All the elementary intervals of an unoriented component are unoriented. Therefore, it is impossible to sort unoriented components using oriented reversals. In the next section, we discuss the type of reversals that can be used to create oriented intervals in unoriented components.

2.3 Effects of a reversal on components

The next two propositions describe the effects of a reversal whose endpoints are in the same unoriented component. These are classical results from the Hannenhalli-Pevzner theory.

Proposition 7. *If a component C is unoriented, no reversal with its two endpoints in C can split one of its cycles, or create a new component.*

Proposition 8. *If a component C is unoriented, the reversal of an elementary interval whose endpoints belong to C orients C, and leaves the number of cycles of the permutation unchanged.*

Orienting a component as in Proposition 8 is called *cutting* the component. Cutting an unoriented component is seldom used in optimal sorting of a permutation since it is possible, with a single reversal, to get rid of more than one unoriented component. The following proposition describes how to *merge* several components, and the relations of this operation to paths in T_P.

Proposition 9. *If a reversal has its two endpoints in different components A and B, then only the components on the path from A to B in T_P are affected.*

1) A component C is destroyed if and only if it contains either A or B, but not both.
2) If A or B is unoriented, any component C that contains both A and B, and that is on the path that joins A and B, will be oriented.
3) A new component C is created if and only if A and B are included in two components that are in the same chain. If either A or B is unoriented, C will be oriented.

Sketch of proof: 1) One of the bounding elements of C will change sign, but not the other. 2) By 1), all components between A and C, and all components between B and C will be destroyed. Suppose that A is unoriented, then reversing one bounding element of A will introduce, in C, at least one oriented interval. 3) If A is included in A', and B is included in B', such that $A' = (a..a')$ precedes $B' = (b..b')$ in the same chain, then $C = (a..b')$ will be a new component. ■

Proposition 9 thus states that merging two unoriented components destroys or orients all components on the path, without creating new unoriented components.

3 The Hannenhalli-Pevzner theorem

In this section, we develop a formula for computing the reversal distance of a permutation. There are two basically different problems: the contribution of oriented components to the total distance is treated in Section 3.1, and the general formula is given in Section 3.2.

3.1 Sorting oriented components

Sorting oriented components is done by choosing oriented reversals that do not create new unoriented components. Several different criteria for choosing such reversals exist in the literature, and we give here the simplest one.

Definition 7. *The* score *of a reversal is the number of oriented elementary intervals in the resulting permutation.*

Theorem 1 ([1]). *The reversal of an oriented elementary interval of maximal score does not create new unoriented components.*

Corollary 1. *If a permutation P on the set $\{0, \ldots, n\}$ has no unoriented components and c cycles, then $d(P) = n - c$.*

Proof: As stated following Proposition 3, we have that $d(P) \geq n - c$ since any reversal adds at most 1 cycle, and the identity permutation has n cycles. Any oriented reversal adds one cycle, thus Theorem 1 guarantees that there will be always enough oriented reversals to sort the permutation. ■

3.2 Computing the reversal distance

Definition 8. *A* cover *$\mathcal{C}$ of T_P is a collection of paths joining all the unoriented components of P, such that each terminal node of a path belongs to a unique path.*

By Propositions 8 and 9, each cover of T_P describes a set of reversals that orients all the components of P. A path that contains two or more unoriented components, called a *long* path, corresponds to merging the two components at its terminal nodes. A path that contains only one component, a *short* path, corresponds to cutting the component.

The *cost* of a cover is defined to be the sum of the costs of its paths, given that:

1) The cost of a short path is 1.

2) The cost of a long path is 2.

An *optimal* cover is a cover of minimal cost. Define t as the cost of any optimal cover T_P.

Theorem 2. *If a permutation P on the set $\{0, \ldots, n\}$ has c cycles, and the associated tree T_P has minimal cost t, then*

$$d(P) = n - c + t.$$

Proof: We first show that $d(P) \leq n - c + t$. Let $\mathcal{C}$ be an optimal cover. Apply to P the sequence of m merges and q cuts induced by the cover $\mathcal{C}$. Note that $t = 2m+q$. By Proposition 6, the resulting permutation P' has $c-m$ cycles, since merging two components always merges two cycles, and cutting components does not change the number of cycles. Thus, by Corollary 1, $d(P') = n - c + m$. Since $m + q$ reversals were applied to P, we have:

$$d(P) \leq d(P') + (m + q) = n - c + 2m + q = n - c + t.$$

In order to show that $d(P) \geq n - c + t$, consider any sequence of length d that optimally sorts the permutation. By Proposition 3, d can be written as

$$d = s + m + q,$$

where s is the number of reversals that split cycles, m is the number of reversals that merge cycles, and q is the number of reversals that do not change the number of cycles. Since the m reversals remove m cycles, and the s reversals add s cycles, we must have:

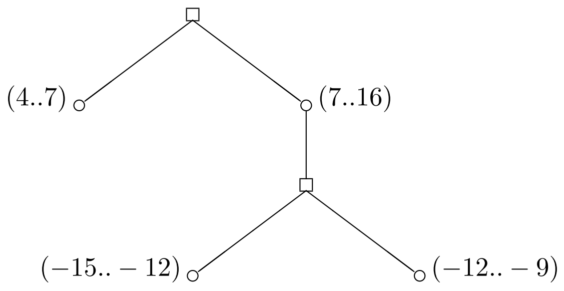
$$c - m + s = n, \text{ implying } d = n - c + 2m + q.$$

The sequence of d reversals induces a cover of T_P. Indeed, any reversal that merges a group of components traces a path in T_P, of which we keep the shortest segment that includes all unoriented components of the group. Of these paths, suppose that m_1 are long paths, and m_2 are short paths. Clearly, $m_1 + m_2 \leq m$. The $q' \leq q$ remaining unoriented components are all cut. Thus

$$2m_1 + m_2 + q' \leq 2m + q.$$

Since $t \leq 2m_1 + m_2 + q'$, we get $d \geq n - c + t$. ■

The last task is to give an explicit formula for t. Let T' be the smallest subtree of T_P that contains all unoriented components of P. Formally, T' is obtained by recursively removing from T_P all dangling oriented components and square nodes. All leaves of T' will thus be unoriented components, while internal round nodes may still represent oriented components. For example, the tree T' obtained from T_{P_2} contains one oriented and three unoriented components.



Define a *branch* of a tree as the set of nodes from a leaf up to, but excluding, the next node of degree $\geq$ 3. A *short* branch of T' contains 1 unoriented component, and a *long* branch contains 2 or more unoriented components. We have:

Theorem 3. *Let T' be the subtree of T_P that contains all the unoriented components.*

(1) If T' has $2k$ leaves, then $t = 2k$.
(2) If T' has $2k+1$ leaves, one of them on a short branch, then $t = 2k+1$.
(3) If T' has $2k+1$ leaves, none of them on a short branch, then $t = 2k+2$.

Proof: Let $\mathcal{C}$ be an optimal cover of T', with m long paths and q short ones. By joining any pair of short paths into a long one, $\mathcal{C}$ can be transformed into an optimal cover with $q = 0$ or 1.

Any optimal cover has only one path on a given branch, since if there were two, one could merge the two paths and lower the cost. Thus if a tree has only long branches, there always exists an optimal cover with $q = 0$.

Since a long path covers at most two leaves, we have $t = 2m + q \geq l$, where l is the number of leaves of T'. Thus cases (1) and (2) are lower bounds. But if $q = 0$, then t must be even, and case (3) is also a lower bound.

To complete the proof, it is thus sufficient to exhibit a cover achieving these lower bounds. Suppose that $l = 2k$. If $k = 1$, the result is obvious. For $k > 1$, suppose T' has at least two nodes of degree ≥ 3. Consider any path in T' that contains two of these nodes, and that connects two leaves A and B. The branches connecting A and B to the tree T' are incident to different nodes of T'. Thus cutting these two branches yields a tree with $2k-2$ leaves. If the tree T' has only one node of degree ≥ 3, the degree of this node must be at least 4, since the tree has at least 4 leaves. In this case, cutting any two branches yields a tree with $2k-2$ leaves.

If $l = 2k+1$ and one of the leaves is on a short branch, select this branch as a short path, and apply the above argument to the rest of the tree. If there is no short branch, select a long branch as a first (long) path. ■

4 Algorithms

In this section we present a simple algorithm to compute the reversal distance of a permutation P based on Theorems 2 and 3. The algorithm consists of two parts. The components of P are first computed by an algorithm presented in [2], then the tree T_P is created by a simple pass over the components of P.

For completeness, we briefly recall the algorithm from [2], called Algorithm 1 here. The input of the algorithm is a signed permutation P, separated into an array of unsigned elements $\pi = (\pi_0, \pi_1, \ldots, \pi_n)$ and an array of signs $\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_n)$. The algorithm finds all components of P in linear time. It makes use of four stacks, two of which (M_1 and M_2) are used to compute two arrays M and m, defined as follows:

$M[i]$ is the nearest element of π that precedes π_i and is greater than π_i,
$m[i]$ is the nearest element of π that precedes π_i and is smaller than π_i.

The algorithm to find the components uses two stacks S_1 and S_2 that store potential start points s of components, which are then tested by the following criterion: $(s..i)$ is a direct component if and only if:

1) both σ_s and σ_i are positive,
2) all elements between π_s and π_i in π are greater than π_s and smaller than π_i, the latter being equivalent to the simple test $M[i] = M[s]$, and
3) no element "between" π_s and π_i is missing, i.e. $i - s = \pi_i - \pi_s$.

A symmetric criterion allows to find reverse components. For details, see Algorithm 1. Without much overhead it is also possible to tell whether each component is oriented or not. Again, details can be found in [2].

Note that Algorithm 1 reports the components in left-to-right order with respect to their right end. For each index i, $0 \leq i \leq n$, at most one component can start at position i and at most one component can end at position i. Hence, it is possible to create a data structure that tells in constant time if there is a component beginning or ending at position i and, if so, reports such components. Given this data structure, it is a simple procedure to construct the tree T_P in one left-to-right scan along the permutation. Initially one square root node and one round node representing the component with left bounding element 0 are created. Then, for each additional component, a new round node p is created as the child of a new or an existing square node q, depending if p is the first component in a chain or not. For details, see Algorithm 2.

To generate tree T' from tree T_P, a bottom-up traversal of T_P recursively removes all dangling round leaves that represent oriented components, and square nodes. Given the tree T', it is easy to compute the reversal distance: perform a depth-first traversal of T' and count the number of leaves and the number of long and short branches. Then use the formula from Theorem 3 to obtain t, and the formula from Theorem 2 to obtain d.

Altogether we have:

Theorem 4. *Using Algorithms 1 and 2, the reversal distance $d(P)$ of a permutation P on the set $\{0, \ldots, n\}$ can be computed in linear time $O(n)$.*

5 Conclusion

In this paper, we presented a simpler formula for the Hannenhalli-Pevzner reversal distance equation. It captures the notion of hurdles, super-hurdles and fortresses in a single parameter whose value can be computed with the help of a *PQ*-tree. Our next goal is to apply this kind of simplification to the harder problem of comparing multi-chromosomal genomes, whose treatment currently involves half a dozen parameters [6].

References

1. A. Bergeron. A very elementary presentation of the Hannenhalli-Pevzner theory. In *CPM 2001 Proceedings*, volume 2089 of *LNCS*, pages 106–117. Springer Verlag, 2001.

Algorithm 1 (Find the components of a signed permutation $P = (\pi, \sigma)$)

```
1: $M_1$ and $M_2$ are stacks of integers; initially $M_1$ contains $n$ and $M_2$ contains 0
2: $S_1$ and $S_2$ are stacks of integers; initially $S_1$ contains 0 and $S_2$ contains 0
3: $M[0] \leftarrow n$, $m[0] \leftarrow 0$
4: for $i \leftarrow 1, \dots, n$ do
       (* Compute the $M[i]$ *)
5:     if $\pi[i-1] > \pi[i]$ then
6:         push $\pi[i-1]$ on $M_1$
7:     else
8:         pop from $M_1$ all entries that are smaller than $\pi[i]$
9:     end if
10:    $M[i] \leftarrow$ the top element of $M_1$
       (* Find direct components *)
11:    pop the top element $s$ from $S_1$ as long as $\pi[s] > \pi[i]$ or $M[s] < \pi[i]$
12:    if $\sigma[i] = +$ and $M[i] = M[s]$ and $i - s = \pi[i] - \pi[s]$ then
13:        report the component $(s..i)$
14:    end if
       (* Compute the $m[i]$ *)
15:    if $\pi[i-1] < \pi[i]$ then
16:        push $\pi[i-1]$ on $M_2$
17:    else
18:        pop from $M_2$ all entries that are larger than $\pi[i]$
19:    end if
20:    $m[i] \leftarrow$ the top element of $M_2$
       (* Find reversed components *)
21:    pop the top element $s$ from $S_2$ as long as $(\pi[s] < \pi[i]$ or $m[s] > \pi[i])$ and $s > 0$
22:    if $\sigma[i] = -$ and $m[i] = m[s]$ and $i - s = \pi[s] - \pi[i]$ then
23:        report the component $(s..i)$
24:    end if
       (* Update stacks *)
25:    if $\sigma[i] = +$ then
26:        push $i$ on $S_1$
27:    else
28:        push $i$ in $S_2$
29:    end if
30: end for
```

Algorithm 2 (Construct T_P from the components $C_1, \ldots, C_k$ of P)

```
 1: create a square node $q$, the root of $T_P$ and a round node $p$ as the child of $q$
 2: for $i \leftarrow 1, \ldots, n-1$ do
 3:   if there is a component $C$ starting at position $i$ then
 4:     if there is no component ending at position $i$ then
 5:       create a new square node $q$ as a child of $p$
 6:     end if
 7:     create a new round node $p$ (representing $C$) as a child of $q$
 8:   else if there is a component ending at position $i$ then
 9:     $p \leftarrow$ parent of $q$
10:     $q \leftarrow$ parent of $p$
11:   end if
12: end for
```

2. A. Bergeron, S. Heber, and J. Stoye. Common intervals and sorting by reversals: A marriage of necessity. *Bioinformatics*, 18(Suppl. 2):S54–S63, 2002. (Proceedings of ECCB 2002).
3. A. Bergeron and J. Stoye. On the similarity of sets of permutations and its applications to genome comparison. In *Proceedings of COCOON 03*, volume 2697 of *LNCS*, pages 68–79. Springer Verlag, 2003.
4. K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs and graph planarity using *PQ*-tree algorithms. *J. Comput. Syst. Sci.*, 13(3):335–379, 1976.
5. S. Hannenhalli. Polynomical algorithm for computing translocation distance between genomes. *Discrete Appl. Math.*, 71(1-3):137–151, 1996.
6. S. Hannenhalli and P. A. Pevzner. Transforming men into mice (polynomial algorithm for genomic distance problem). In *Proceedings of FOCS 1995*, pages 581–592. IEEE Press, 1995.
7. S. Hannenhalli and P. A. Pevzner. Transforming cabbage into turnip: Polynomial algorithm for sorting signed permutations by reversals. *J. ACM*, 46(1):1–27, 1999.
8. H. Kaplan, R. Shamir, and R. E. Tarjan. A faster and simpler algorithm for sorting signed permutations by reversals. *SIAM J. Computing*, 29(3):880–892, 1999.
9. J. D. Kececioglu and D. Sankoff. Efficient bounds for oriented chromosome inversion distance. In *Proceedings of CPM 94*, volume 807 of *LNCS*, pages 307–325. Springer Verlag, 1994.
10. Berman P. and Hannenhalli S. Fast sorting by reversal. In *CPM 1996 Proceedings*, volume 1075 of *LNCS*, pages 168–185. Springer Verlag, 1996.