Available online at www.sciencedirect.com

ScienceDirect

Discrete Applied Mathematics 155 (2007) 881–888

DISCRETE APPLIED MATHEMATICS

www.elsevier.com/locate/dam

Advances on sorting by reversals

Eric Tannier[a], Anne Bergeron[b], Marie-France Sagot[a]

[a]*INRIA Rhône-Alpes, Laboratoire de Biométrie et Biologie Évolutive, Université Claude Bernard, 69622 Villeurbanne cedex, France*
[b]*Laboratoire de combinatoire et d'informatique mathématique, Université du Québec à Montréal, Canada*

Received 13 July 2004; received in revised form 1 February 2005; accepted 17 February 2005
Available online 17 October 2006

Abstract

The problem of sorting signed permutations by reversals is inspired by genome rearrangement problems in computational molecular biology. Given two genomes represented as signed permutations of the same elements (e.g. orthologous genes), the problem consists in finding a most parsimonious scenario of reversals that transforms one genome into the other. Following the first polynomial solution of this problem, several improvements, simplifications, generalizations, tutorials or surveys have been published on the subject. While the reversal distance problem—i.e. the problem of computing the minimum number of reversals in a sorting sequence, without giving the sequence itself—seems to be well explored, the problem of giving a scenario realizing the distance still raises some open questions, one of which by Ozery-Flato and Shamir about whether an algorithm with subquadratic time complexity could ever be achieved for solving the problem. We give a positive answer to this question by describing an algorithm of time complexity $O(n^{3/2}\sqrt{\log n})$.

Keywords: Reversal sorting; Genome rearrangement; Computational biology

1. Introduction

The problem of sorting a permutation by reversals is inspired and motivated by comparative genomics. Given two or more ordered sets of genes, or other markers on a genome, biologists use methods that solve this problem to estimate an evolutionary distance (number of global mutation events) between genomes, to infer the localization of evolutionary breakpoints along a DNA molecule, or to reconstruct ancestral genomes.

Genome rearrangements such as reversals may change the order of the genes in a genome, and also the direction of transcription. We identify the genes with the integers $1, \dots, n$, with a plus or minus sign to indicate their direction. The order and direction of genomic markers will be represented by a *signed permutation* of $\{1, \dots, n\}$, that is a bijective function π over $[-n, n]\backslash\{0\}$ such that $\pi_{-i} = -\pi_i$, where $\pi_i = \pi(i)$.

To simplify exposition, we adopt the usual extension which consists in adding $\pi_0 = 0$, and $\pi_{n+1} = n + 1$ to the permutation. We usually denote a signed permutation by simply writing $(0\ \pi_1\ \dots\ \pi_n\ n+1)$. The *identity permutation* $(0\ 1\ \dots\ n+1)$ is denoted by *Id*. The inverse permutation π^{-1} of π is the (signed) permutation such that $\pi \cdot \pi^{-1} = Id$.

The *reversal* of the interval $[i, j] \subseteq [1, n]$ $(i \leqslant j)$ is the signed permutation $\rho_{i,j} = (0\ \dots\ i-1\ -j\ \dots\ -i\ j+1\ \dots\ n+1)$. Note that $\pi \cdot \rho_{i,j}$ is the permutation obtained from π by reversing the order and flipping the signs of the

E-mail addresses: Eric.Tannier@inria.fr (E. Tannier), bergeron.anne@uqam.ca (A. Bergeron), Marie-France.Sagot@inria.fr (M.-F. Sagot).

0166-218X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2005.02.033

elements in the interval $[i, j]$:

$$\pi \cdot \rho_{i,j} = (\pi_0 \ \ldots \ \pi_{i-1} \ -\pi_j \ \ldots \ -\pi_i \ \pi_{j+1} \ \ldots \ \pi_{n+1}).$$

If $\rho_1, \ldots, \rho_k$ is a sequence of reversals, we say that it *sorts* a permutation π if $\pi \cdot \rho_1 \cdots \rho_k = Id$. The length of a smallest sequence of reversals that sorts π is called the *reversal distance* of π, and is denoted by $d(\pi)$.

The problem of sorting by reversals has been the subject of an extensive literature. For an introduction and a general survey—oriented more specifically towards the distance computation problem—see [3]. The first polynomial algorithm was given by Hannenhalli and Pevzner [4], and took $O(n^4)$ time to give a sequence of reversals that optimally sorts a signed permutation. After many subsequent improvements on the running time, the currently fastest algorithms are those of Kaplan et al. [5], and another by Berman and Hannenhalli [2], both running in $O(n^2)$. Bader et al. [1] designed a linear time algorithm for computing $d(\pi)$, without giving the sequence of reversals. Noting that the most costly part of the algorithms that give a sequence is to check for very unfrequent configurations, Kaplan and Verbin [6] presented a random algorithm which runs in $O(n^{3/2}\sqrt{\log n})$, and gives, most of the time, an optimal sequence of reversals, but fails with very high probability on some precise permutations. In a recent paper [7], Ozery-Flato and Shamir compiled and compared the best algorithms, and wrote that: “A central question in the study of genome rearrangements is whether one can obtain a subquadratic algorithm for sorting by reversals”.

In this paper, we give a positive answer to Ozery-Flato and Shamir’s question. Inspired by the relations between the classical approaches [2,4,5], the data structure given in [6], and some remarks we first made in [9], we describe how it is possible to bypass the usual costly tests and give an optimal solution in time $O(n^{3/2}\sqrt{\log n})$.

Sorting a signed permutation π by reversals involves two successive procedures on what is called the *overlap graph* of π. The first one consists in transforming the overlap graph such that all its connected components become *oriented*. This can be done in linear time [1,3]. The computational bottleneck of the sorting by reversals problem occurs in the second procedure: sorting the oriented components. This paper addresses this problem.

In the next section, we define overlap graphs and basic operations on them. We next state, in a graph theoretical framework, a theorem by Hannenhalli and Pevzner that is the cornerstone of all sorting strategies. In Section 3, we give a new proof of this theorem that yields a constructive algorithm to sort oriented components without testing beforehand the safety of a reversal. However, with any classical data structure, such as vector arrays to represent permutations, the time complexity of this new algorithm is still $O(n^2)$, even if this bound is achieved much more naturally. In Section 4, we show that the data structure introduced in [6] can be adapted to achieve an overall subquadratic time complexity.

2. Ingredients

2.1. The overlap graph of a permutation

Let π be a signed permutation on $\{0, \ldots, n+1\}$. For each integer $i \in \{0, \ldots, n\}$, the pair $\pi_i\pi_{i+1}$ is called an *adjacency* if $\pi_i + 1 = \pi_{i+1}$, and a *breakpoint* otherwise. To each π_i, $i \in \{0, \ldots, n+1\}$, are associated two points, named π_i^- and π_i^+, except for 0 and $n+1$, for which we define only the points 0^+ and $(n+1)^-$. These points are ordered in the following way: $\pi_i^- < \pi_i^+$ if π_i is non negative, and $\pi_i^+ < \pi_i^-$ otherwise, and $\pi_i^x < \pi_j^y$ whenever $i < j$ for any combination of $x, y \in \{+, -\}$.

If $i \in \{0, \ldots, n\}$, v_i will denote an arc between the points π_i^+ and π_{i+1}^-. Thus, there are $n+1$ arcs and each point is the endpoint of a unique arc. Such arcs will be referred to as the *arcs* of π. Two arcs are said to *overlap* if the intervals they span (that is the set of points between the endpoints in the given order) intersect but none is contained in the other. Note that any arc v naturally induces a reversal $\rho(v) = \rho_{i,j}$: $[i, j]$ contains all the numbers k such that k^+ and k^- are between the endpoints of v. The arc v_i is said to be *oriented* if π_i and π_{i+1} have different signs, and *unoriented* otherwise.

The *overlap graph* $OV(\pi)$ of a permutation π is the graph whose vertices are the $n+1$ arcs v_i of π, and in which there is an edge between vertices v_i and v_j if they overlap. The overlap graph of a permutation has thus oriented and unoriented vertices. Isolated vertices are unoriented, and correspond to adjacencies of the permutation.

A *component* of π is a connected component of $OV(\pi)$. It is *oriented* if one of its vertices is oriented, and *unoriented* otherwise. For simplicity, when we speak of unoriented components of graphs or permutations, we exclude isolated vertices (they correspond to adjacencies of the permutation).

An example of overlap graph is given in Fig. 1. In this example, there is a unique component, which is oriented.

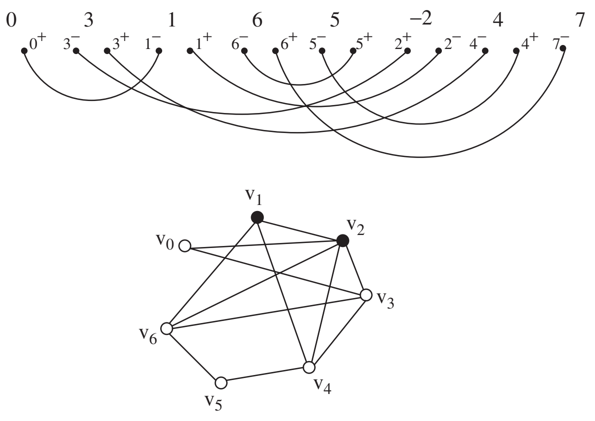


Fig. 1. The permutation (0 3 1 6 5 −2 4 7), with associated points and arcs; the overlap graph of the permutation, in which oriented vertices are filled in black, and unoriented ones in white.

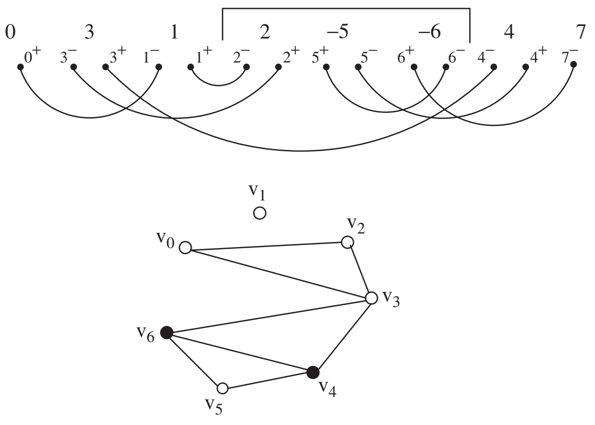


Fig. 2. The reversal $\rho(v_1) = \rho_{3,5}$ in permutation (0 3 1 6 5 −2 4 7) of Fig. 1, and the corresponding local complementation of vertex v_1 in the overlap graph.

A graph with oriented and unoriented vertices is not necessarily the overlap graph of a permutation. However, the graph theoretical definitions and results of the next section are general, and will hold in particular for overlap graphs of permutations.

2.2. The local complementation of a graph

Let G be a graph whose vertices are labelled as oriented or unoriented. If V is a subset of the vertices of G, the *subgraph induced* by V is the graph with vertex set V, with an edge between two vertices if and only if it is an edge of G.

The *local complementation* of the subgraph induced by V is the operation which consists in adding an edge between $x, y \in V$ if there is no edge $x, y \in G$, deleting x, y if there is an edge x, y in G, and changing the orientation of all vertices in V.

Given a vertex v, we denote by $\Gamma(v)$ the *neighbourhood* of v, that is, the set of vertices adjacent to v, and $\Gamma^+(v) = \Gamma(v) \cup \{v\}$ the *closed neighbourhood* of v.

If v is an oriented vertex, we denote by G/v the result of the local complementation of $\Gamma^+(v)$, which we simply call the local complementation of v. Note that in G/v, v is unoriented and isolated. An example of local complementation is given in Fig. 2.

The relation between sorting by reversals and local complementation is given by the following lemma (see [4,5]).

Lemma 1. *For a permutation π, and an oriented vertex v of the overlap graph, $OV(\pi \cdot \rho(v)) = OV(\pi)/v$.*

2.3. The theorem of Hannenhalli and Pevzner

We finish this section by stating a version of Hannenhalli and Pevzner's theorem [4], which is the basis of the sorting by reversals theory. We split the result into two because we give an alternate proof of the first part of the theorem along the lines of this paper, and we restrict the result to permutations without unoriented component.

Theorem 1 (*Hannenhalli and Pevzner [4]*). *If G is a graph without unoriented component, then there exists an oriented vertex v such that G/v has no unoriented component.*

An oriented vertex v in G such that there is no unoriented component in G/v, is called *safe*. The relation to sorting by reversals is given by the following:

Theorem 2 (*Hannenhalli and Pevzner [4]*). *If v is a safe vertex of the overlap graph of a permutation π, then $d(\pi \cdot \rho(v)) = d(\pi) - 1$.*

Theorem 2 means that, in order to sort a permutation π whose unoriented components have been cleared, we just have to find a safe vertex at each step, and we know that it exists from Theorem 1. This may be achieved by choosing an oriented vertex, applying the local complementation, and testing if there is an unoriented component in the resulting graph; if there is one, we undo the local complementation, and try another oriented vertex. This yields an $O(n^3)$ algorithm, and was the principle of Hannenhalli and Pevzner's one. Faster techniques have been discovered, surveyed in [7], to find a safe vertex in linear time, providing $O(n^2)$ algorithms. Our method consists in bypassing the safety test of oriented vertices, and making some repairs later if a vertex chosen at some point was not safe. This further decreases the complexity.

3. New recipe

We provide a new proof of Theorem 1, with a slight detour. We use a graph theoretical proof, because the result holds for general graphs with oriented and unoriented vertices, and we apply it in the next section to the particular case of overlap graphs of permutations.

We proceed by constructing a sequence of vertices of a graph, such that at each step, the vertex we select is oriented, and after the last step, the graph has only unoriented isolated vertices. The local complementation always acts on a single component, so each component is treated separately, and we may suppose, without loss of generality, that the graph has only one component.

Let G be a graph with n vertices. A *sequence of oriented vertices* for G is a sequence $v_1, \ldots, v_k$, such that for all $i \in \{1, \ldots, k\}$, v_i is an oriented vertex in $G/v_1/\ldots/v_{i-1}$.

A sequence of oriented vertices is said to be *maximal* if no vertex of $G/v_1/\ldots/v_k$ is oriented. It is *total* if it is maximal and every vertex of $G/v_1/\ldots/v_k$ is isolated. We prove that there always exists a total sequence in any connected graph with at least one oriented vertex. This is equivalent to Theorem 1.

3.1. The main step

The algorithm for constructing a total sequence is based on a technique that, given a maximal but not total sequence of reversals, increases the length of the sequence by adding some vertices, not at the end since it is not possible by definition, but within the sequence.

Theorem 3. *If S is a maximal but not a total sequence of oriented vertices for a graph G with a unique oriented component, then there exists a nonempty sequence S' of vertices of G such that S may be split into two parts $S = S_1, S_2$, and S_1, S', S_2 is a sequence of oriented vertices for G.*

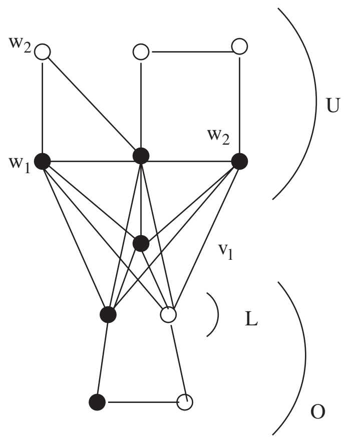


Fig. 3. An example of the decomposition of the graph G_1, with two possible choices of w_2. Oriented vertices are filled in black, and unoriented ones in white. The subgraph induced by O is the same after the complementation of v_l as the one obtained after the complementation of w_1, w_2, v_l.

Proof. Let $S = v_1, \ldots, v_k$ be a maximal sequence of oriented vertices of G. Let U be the set of nonisolated vertices in $G/v_1/\ldots/v_k$. The vertices of U all belong to unoriented components of $G/v_1/\ldots/v_k$. Since by hypothesis there is only one component in G, there is a vertex v_l in the sequence, such that all vertices of U are in unoriented components in $G/v_1/\ldots/v_l$, but not in $G/v_1/\ldots/v_{l-1}$. Let $S_1 = v_1, \ldots, v_{l-1}$ and $S_2 = v_l, \ldots, v_k$. This will be the way of splitting S in two, as described in the statement of the theorem. We will show that it is always possible to apply two local complementations to vertices of U, and that S_2 is a sequence of oriented vertices in the resulting graph.

Let $G_1 = G/v_1/\ldots/v_{l-1}$. Let O be the set of vertices in oriented components of G_1/v_l, and L be the subset of vertices of O adjacent to v_l in G_1. In G_1, the following properties are straightforward consequences of the definition of local complementation, and are illustrated by Fig. 3.

(1) A vertex of U is oriented if and only if it is adjacent to v_l.
(2) There are all possible edges between $\Gamma(v_l) \cap U$ and L.
(3) There is no edge between vertices of $U \backslash \Gamma(v_l)$ and the vertices outside U, and there is no edge between vertices of O$\backslash L$ and the vertices outside O.

There is at least one vertex w_1 in $\Gamma(v_l) \cap U$ such that $\Gamma^+(v_l) \neq \Gamma^+(w_1)$: if not, then the local complementation of any oriented vertex in U has the same effect than the local complementation of v_l, and in G_1/v_l, all vertices of $\Gamma(v_l)$ are isolated, thus not in U; this contradicts the definition of v_l.

Thus, there exists $w_2 \in U$, either in $\Gamma^+(v_l) \backslash \Gamma^+(w_1)$, or in $\Gamma^+(w_1) \backslash \Gamma^+(v_l)$.

In G_1/w_1, v_l is unoriented and adjacent to w_2, and w_2 is oriented. The properties (1)–(3) still hold (note that $\Gamma(v_l)$ changes but the set of vertices L is invariant), v_l has no neighbour in O and the subgraph induced by L is the complement of what it is in G_1.

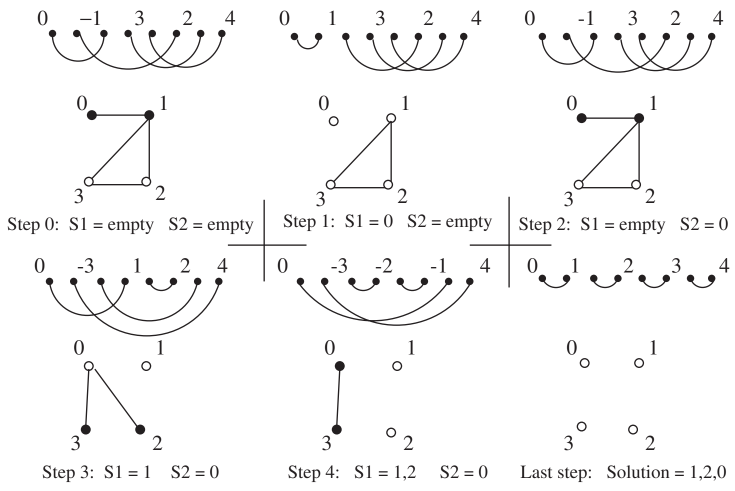
In $G_1/w_1/w_2$, properties (1)–(3) still hold, v_l is oriented, and the subgraph induced by L is complemented again, so it is identical in $G_1/w_1/w_2$ to what it is in G_1; therefore the subgraph induced by O is also identical in $G_1/w_1/w_2$ to what it is in G_1, and S_2 is a sequence of oriented vertices in $G_1/w_1/w_2$, since it is one in G_1.

Then S_1, w_1, w_2, S_2 is a sequence of oriented vertices of G, and this concludes the proof. □

3.2. The algorithm

Theorem 3 and its proof provides a way to construct a total sequence of oriented vertices, by constructing a maximal sequence, and then augmenting it until it is total. This yields an algorithm of time complexity O(n^2) for arbitrary graphs. For graphs that are overlap graphs of signed permutations, it will be possible to reduce this complexity.

We therefore return to permutations, and we first write the algorithm to sort a permutation efficiently. It relies on Theorem 3 and its proof, but does not need an explicit representation of the overlap graph, that can have O(n^2) edges.



Step 0: S1 = empty S2 = empty
Step 1: S1 = 0 S2 = empty
Step 2: S1 = empty S2 = 0
Step 3: S1 = 1 S2 = 0
Step 4: S1 = 1,2 S2 = 0
Last step: Solution = 1,2,0

Fig. 4. The trace of the algorithm on the permutation (0 -1 3 2 4). The overlap graph is drawn as a help to visualize components, but note that it is not computed in the algorithm. It will be handled by the data structure we present in Section 4.

Sequence augmentation sorting algorithm:

```
Init. Clear the unoriented components of the initial permutation yielding $\pi$.
      $V$ is the set of all arcs of $\pi$.

      $S_1$, the left sequence, is the empty set.

      $S_2$, the right sequence, is the empty set.
(1)   While there is an oriented arc $v$ in $V$, add the corresponding reversal $\rho(v)$ at the end of the sequence $S_1$, and apply
      it to the current permutation ($\pi \leftarrow \pi \cdot \rho(v)$). Remove $v$ from $V$, as well as other possible adjacencies created by
      the application of $\rho(v)$.
      If the first element of $S_2$ is not oriented, go one step back.[1]
(2)   If $V$ is empty, go to step (3);
      else suppose $S_1 = \rho_1, \ldots, \rho_k$. (Re-)Apply the reversals of $S_1$ in the reverse order: $\pi := \pi \cdot \rho_k \cdots \rho_l$, until there is
      an oriented arc in $V$. Remove $\rho_l, \ldots, \rho_k$ from $S_1$; add $\rho_l, \ldots, \rho_k$ at the beginning of $S_2$. Gotostep.
(3)   The sequence $S_1, S_2$ sorts $\pi$.
```

For the permutation (0 -1 3 2 4), the algorithm works as follows (see Fig. 4): the unsafe reversal $\rho(v_0)$ is applied, then re-applied because no other arc is oriented ($v_0 = v_l$); then $\rho(v_1)$ is applied, followed by $\rho(v_2)$, and $\rho(v_3)$. Since v_0 is not oriented and there is no other oriented arc in V, it goes one step back, and the sequence $\rho(v_1), \rho(v_2), \rho(v_0)$ sorts the permutation.

4. Complexity

In the sequence augmentation sorting algorithm, any reversal is applied at most twice, once at Step 1 when it is in V, and perhaps once backwards at Step 2. When it is removed from V, it cannot be applied anymore. The complexity is therefore determined by the time necessary to detect and choose an oriented arc, and apply the corresponding reversal to the permutation. With a classical data structure, using for instance a vector array to represent the current permutation, this is easily achievable in linear time. As a consequence, the algorithm, as well as any algorithm for sorting by reversals that has to apply a reversal at each step, will run in $O(n^2)$.

[1] This is the case when the last arc of S_1 has the same neighbourhood in $OV(\pi)$ as the first arc in S_2. Then it makes no difference between leaving one or the other in the final sequence.

In practice, even this $O(n^2)$ implementation should be an improvement on the existing algorithms. For small permutations, such quadratic implementation is probably better than one that would be subquadratic but use another complex data structure. However, it is an important mathematical question whether sorting a signed permutation by reversals can be done in a theoretical subquadratic time complexity. We therefore use a more sophisticated implementation to make it run in $O(n^{3/2}\sqrt{\log n})$.

In [6], Kaplan and Verbin described a clever data structure which allows to choose an oriented arc and apply the corresponding reversal in sublinear time. We use the same data structure, just adding some flags in order to be able to choose an oriented arc in a specific subset of the arcs, whereas the original structure is designed to choose one at random in the whole set of arcs.

As in [6], we store the permutation in a vector array split into blocks, each of size at least $\frac{1}{2}\sqrt{n\log n}$ and at most $2\sqrt{n\log n}$. We assign a flag to each block, raised if the block should be read in the reverse order, changing the sign of the elements. In this way, a reversal can be achieved in $O(\sqrt{n\log n})$. The procedure is as follows:

(1) split at most two blocks so that the endpoints of the reversal correspond to endpoints of blocks;
(2) reverse the order of the blocks between the endpoints of the reversal;
(3) reverse the flag of each block between the endpoints of the reversal;
(4) concatenate and split blocks in such a way that the size of each block lies within the interval $[\frac{1}{2}\sqrt{n\log n}, 2\sqrt{n\log n}]$.

We must now show that it is possible to choose an oriented arc in sublinear time. In order to do so, we assign to each block a balanced binary tree, whose nodes store the elements of the block. The nodes of the tree are ordered such that in any subtree, all the nodes to the left of the root precede the root, and all the nodes to the right of the root come after. A node i precedes a node j in the tree, according to the position of their successor in π, that is $|\pi^{-1}_{i+1}| < |\pi^{-1}_{j+1}|$. Given a node i, it is possible to split the tree in two by a rotation procedure that takes $O(\log n)$ time [8]. This procedure yields two balanced trees, one that contains all the nodes that precede i, and the other all the nodes that come after i. It is also possible, in $O(\log n)$ time, to concatenate two trees in a balanced tree, if all elements of the first one precede all elements of the second one.

Each node i of the tree corresponds to an arc v_i of the permutation π. The node i stores the orientation of the arc v_i, the number of oriented arcs in the subtree rooted at node i, and a flag that indicates whether this subtree is *reversed* or not, that is, its nodes should be ordered backwards with respect to the original order, and all nodes change orientation. The "reversed" property is propagated to all subtrees, and cancelled when reaching a subnode whose flag is raised.

We now have to show how to maintain this structure while performing a reversal. Let us return to the reversal procedure step by step:

1. after splitting blocks, we must reconstruct from scratch the associated trees;

2, 3. after reversing the order and flags of the blocks inside the reversal, we process every tree, even those outside the reversal, separately. Each tree T is split into three parts T_1, T_2 and T_3, with T_1 containing all elements of T that occur before the first endpoint of the reversal, T_2, the elements of T that occur within the reversal, and T_3, the elements of T that occur after the second endpoint of the reversal.
- If T is a tree corresponding to a block outside the reversal, flip the flag at the root of T_2, and concatenate T_1, T_2 and T_3;
- If T is a tree corresponding to a block inside the reversal, flip the flag at the root of T_1 and T_3, and concatenate T_1, T_2 and T_3;

4. after concatenating and splitting blocks, reconstruct from scratch the associated trees.

In this way, it is possible to apply a reversal and maintain the data structure in time $O(\sqrt{n\log n})$.

Up to this point, we have described exactly the data structure of Kaplan and Verbin. We now add flags and numbers to the nodes of the trees in order to perform our own queries. Let V be a subset of the arcs of π. At the beginning, V is the set of all arcs of π. To each node of each tree, a new flag is added, raised if the corresponding arc is in V, and a "running total" stores the number of oriented arcs that belong to V in the subtree. Fig. 5 gives a representation of the data structure applied to the permutation of Fig. 1 split in two blocks. Observe that we do not need to know the total number of oriented arcs, but only the number of those that are in V.

In this way, it is possible to know in constant time whether there is an oriented arc in V, and to choose one in time $O(\log n)$.

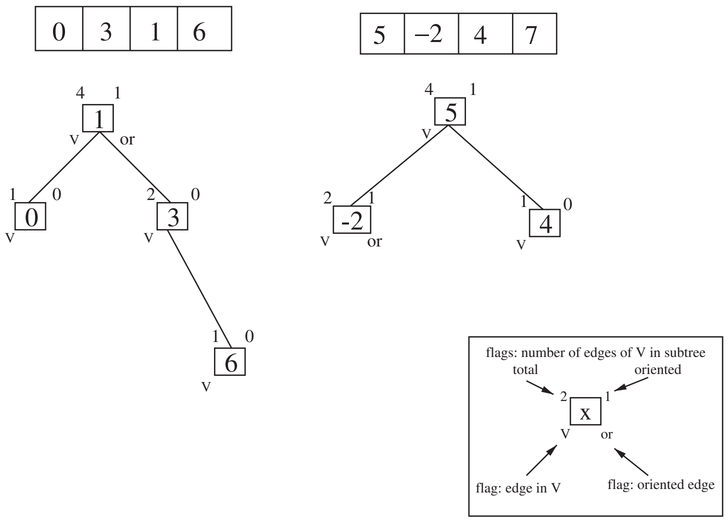


Fig. 5. The data structure for the permutation of Fig. 1 at the beginning of the algorithm (all nodes are in V), if the permutation is split into two blocks.

Finally, we can state the last theorem, which answers Ozery-Flato and Shamir's question [7] on whether it is possible to improve the lower bound on the running time of the algorithms for sorting by reversals.

Theorem 4. *The sequence augmentation sorting algorithm finds a sequence of reversals sorting a permutation in time* $O(n^{3/2}\sqrt{\log n})$.

The remaining bottleneck of the new algorithm is now the complexity at each step of applying a reversal to a permutation and keeping the set of oriented arcs. This is what has to be improved in the future if one wishes to obtain a lower theoretical complexity for sorting a signed permutation by reversals.

Acknowledgments

This work was supported by the French Program Bioinformatique Inter-EPST 2002 "Algorithms for Modelling and Inference Problems in Molecular Biology".

References

[1] D.A. Bader, B.M.E. Moret, M. Yan, A linear-time algorithm for computing inversion distance between signed permutations with an experimental study, in: Proceedings of the Workshop on Algorithms and Data Structures, 2001, pp. 365–376.

[2] P. Berman, S. Hannenhalli, Fast sorting by reversals, in: Proceedings of the CPM'96, Lecture Notes in Computer Science, vol. 1075, Springer, Berlin, 1996, pp. 168–185.

[3] A. Bergeron, J. Mixtacki, J. Stoye, The inversion distance problem, in: O. Gascuel (Ed.), Mathematics of Evolution and Phylogeny, Chapter 10, Oxford University Press, Oxford, 2005, pp. 262–290.

[4] S. Hannenhalli, P. Pevzner, Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals), J. Assoc. Comput. Mach. 46 (1999) 1–27.

[5] H. Kaplan, R. Shamir, R.E. Tarjan, Faster and simpler algorithm for sorting signed permutations by reversals, SIAM J. Comput. 29 (1999) 880–892.

[6] H. Kaplan, E. Verbin, Efficient data structures and a new randomized approach for sorting signed permutations by reversals, in: Proceedings of the CPM'03, Lecture Notes in Computer Science, vol. 2676, Springer, Berlin, 2003, pp. 170–185.

[7] M. Ozery-Flato, R. Shamir, Two notes on genome rearrangement, J. Bioinformatics Computat. Biology 1 (2003) 71–94.

[8] D.D. Sleator, R.E. Tarjan, Self-adjusting binary search trees, J. Assoc. Comput. Mach. 32 (1985) 652–686.

[9] E. Tannier, M.-F. Sagot, Sorting by reversals in subquadratic time, in: Proceedings of the CPM'04, Lecture Notes in Computer Science, vol. 3109, Springer, Berlin, 2004, pp. 1–13.