Available online at www.sciencedirect.com
ScienceDirect
Information Processing Letters ••• (••••) •••–•••
Information Processing Letters
www.elsevier.com/locate/ipl

# A more efficient algorithm for perfect sorting by reversals ☆

Sèverine Bérard [a,b], Cedric Chauve [c,d,*], Christophe Paul [e,1]

[a] *Département de Mathématiques et d'Informatique Appliquée, INRA, Toulouse, France*
[b] *Université Montpellier 2, UMR AMAP, Montpellier, F-34000 France*
[c] *Department of Mathematics, Simon Fraser University, Canada*
[d] *Comparative Genomics Laboratory, Université du Québec à Montréal, Canada*
[e] *CNRS, LIRMM, Université Montpellier 2, France*

Received 22 June 2007; received in revised form 6 October 2007; accepted 17 October 2007

Communicated by F.Y.L. Chin

**Abstract**

We describe a new algorithm for the problem of perfect sorting a signed permutation by reversals. The worst-case time complexity of this algorithm is parameterized by the maximum prime degree $d$ of the strong interval tree, i.e., $f(d).n^{O(1)}$. This improves the best known algorithm which complexity was based on a parameter always larger than or equal to $d$.


*Keywords:* Algorithms; Sorting by reversals; Common intervals

## 1. Introduction

Sorting a signed permutation by reversals is an algorithmical problem that has several applications in the study of genome rearrangements (see [9] for example). In the classical approach a "good" sequence of reversals that sorts a given signed permutation is a parsimonious sequence of reversals. This problem can be solved in subquadratic time [10]. Recently, another combinatorial framework for sorting by reversals was proposed, called *perfect sorting by reversals* [8]. The principle is to look for a sequence of reversals that do not break any *common interval* of the considered signed permutation and is parsimonious among such sequences of reversals. This approach can be seen as a variant of the classical sorting by reversal problem, where the parsimony criterion has been relaxed and the conservation of common intervals is the main criterion.

The first algorithm proposed for this problem has an exponential worst-case running time [8]. It was latter improved in [1] by showing that the problem is fixed parameterized tractable [6] (FPT): i.e., the complexity function is $f(p).n^{O(1)}$ for some parameter $p$. The parameter $p$ proposed in [1] is the number $p$ of prime edges of the strong interval tree of the signed permutation to be sorted (see definition in Section 2). The algorithm proposed in this note is still FPT, but the parameter we

☆ Research supported by NSERC, FQRNT and the "60$^{eme}$ Commission Permanente Franco-Québecoise de Coopération Scientifique". The third author was supported by the French ANR project "Graph Decomposition and Algorithms".

* Corresponding author at: Department of Mathematics, SFU, 8888 University Drive, Burnaby (BC) V5L 1S6, Canada.
*E-mail addresses:* severine.berard@cirad.fr (S. Bérard), cedric.chauve@sfu.ca (C. Chauve), christophe.paul@lirmm.fr (C. Paul).

[1] Research conducted while the third author was on sabbatical at the School of Computer Science, McGill University, Montréal, Canada.

0020-0190/$ – see front matter 
doi:10.1016/j.ipl.2007.10.012

use is always smaller than or equal to the number of prime edges. Hence our new algorithm is then the most efficient, in terms of worst-case time complexity, among published algorithms solving this problem. One motivation for having FPT algorithm is that whenever the parameter remains small, then the problem is still tractable, which for the problem we are interested is most likely to be the case both in real instances and in random instances [5].

## 2. Preliminaries

We first summarize the combinatorial and algorithmical frameworks for perfect sorting by reversals. For a more detailed treatment, we refer to [1].

### 2.1. Permutations, reversals, common intervals and perfect scenarios

A *signed permutation* on $[n]$ is a permutation on the set of integers $[n] = \{1, 2, \ldots, n\}$ in which each element has a sign, positive or negative. Negative integers are represented by placing a bar over them. We denote by $Id_n$ (resp., $\overline{Id_n}$) the identity (resp., reversed identity) permutation, $1\ 2 \ldots n$ (resp., $\bar{n} \ldots \bar{2}\ \bar{1}$). When the number $n$ of elements is clear from the context, we will simply write $Id$ or $\overline{Id}$.

An *interval* $I$ of a signed permutation $P$ on $[n]$ is a segment of consecutive elements of $P$. The *content* of $I$ is the subset of $I$ defined by the absolute values of the elements of $I$. Given $P$, an interval is defined by its content and from now, when the context is unambiguous, we identify an interval with its content.

The *reversal* of an interval of a signed permutation reverses the order of the elements of the interval, while changing their signs. If $P$ is a permutation, we denote by $\overline{P}$ the permutation obtained by reversing the complete permutation $P$. A *scenario* for $P$ is a sequence of reversals that transforms $P$ into $Id_n$ or $\overline{Id_n}$. The *length* of such a scenario is the number of reversals it contains.

Two distinct intervals $I$ and $J$ *commute* if their contents trivially intersect, that is either $I \subset J$, or $J \subset I$, or $I \cap J = \emptyset$. If intervals $I$ and $J$ do not commute, they *overlap*. A *common interval* of a permutation $P$ on $[n]$ is a subset of $[n]$ that is an interval in both $P$ and the identity permutation $Id_n$. The singletons and the set $\{1, 2, \ldots, n\}$ are always common intervals.

A scenario $S$ for $P$ is called a *perfect scenario* if every reversal of $S$ commutes with every common interval of $P$. A perfect scenario of minimal length is called a *parsimonious perfect scenario*.

### 2.2. The strong interval tree

A common interval $I$ of a permutation $P$ is a *strong* interval of $P$ if it commutes with every other common interval of $P$.

The inclusion order of the set of strong intervals defines an $n$-leaf tree, denoted by $T_{\mathcal{S}}(P)$, whose leaves are the singletons, and whose root is the interval containing all elements of the permutation. The strong interval tree of $P$ can be computed in linear time and space (see [2,3] for example). We call the tree $T_{\mathcal{S}}(P)$ the *strong interval tree* of $P$, and we identify a vertex of $T_{\mathcal{S}}(P)$ with the strong interval it represents.

Let $I$ be a strong interval of $P$ and $\mathcal{I} = \{I_1, \ldots, I_k\}$ a partition of the elements of $I$ into maximal strong intervals. The *quotient permutation* of $I$, denoted $P_I$, is defined as follows: $i$ precedes $j$ in $P_I$ if any element of $I_i$ is smaller than any element of $I_j$. The vertex $I$, or equivalently the strong interval $I$ of $P$, is either:

(i) *Increasing linear*, if $P_I$ is the identity permutation, or
(ii) *Decreasing linear*, if $P_I$ is the reversed identity permutation, or
(iii) *Prime*, otherwise.

An edge whose both vertices are prime is called a *prime edge*. The *prime degree* of a prime vertex $I$, denoted $\deg_p(I)$ is the number of prime edges whose two vertices are $I$ and a child of $I$. The *maximum prime degree* of $T_{\mathcal{S}}(P)$ is defined as the maximum, over all prime vertices $I$ of $T_{\mathcal{S}}(P)$, of $\deg_p(I)$. See Fig. 1 for an example of strong intervals tree, linear and prime nodes, and prime degree.

## 3. Computing perfect scenarios with the strong interval tree

### 3.1. Using the strong interval tree as a guide

We first describe the algorithms given in [1] to compute a perfect scenario for a given signed permutation $P$ (Algorithms 1 and 2). The basis of the algorithm is to give signs, $+$ or $-$, to the vertices of $T_{\mathcal{S}}(P)$ using the following set of rules:

(S.1) a leaf receives the sign of the corresponding element in $P$;
(S.2) a linear vertex receives $+$ (resp., $-$) if $I$ is increasing (resp., decreasing);
(S.3) a prime vertex whose parent $J$ is linear receives the sign of $J$.

Please cite this article in press as: S. Bérard et al., A more efficient algorithm for perfect sorting by reversals, Information Processing Letters (2007), doi:10.1016/j.ipl.2007.10.012

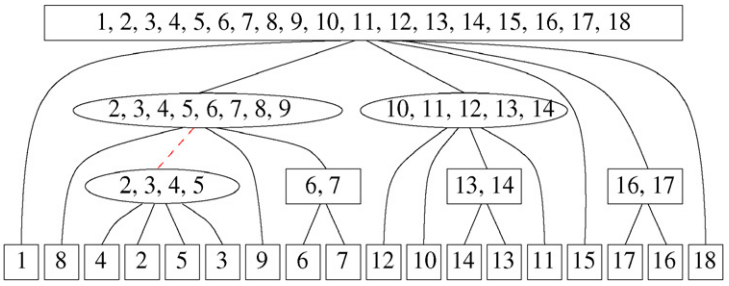


Fig. 1. The strong intervals tree $T_{\mathcal{S}}(P)$ of the permutation $P = (1\ \bar{8}\ 4\ 2\ \bar{5}\ 3\ 9\ \bar{6}\ 7\ 12\ 10\ \bar{14}\ 13\ \bar{11}\ 15\ \bar{17}\ 16\ 18)$. Prime and linear vertices are distinguished by their shape. There are three non-trivial linear vertices, the rectangular vertices, and three prime vertices, the round vertices. There is only one prime edge shown by the dashed line. The prime degree of $T_{\mathcal{S}}(P)$ is one.

Note that these rules can leave some vertices with no sign, when a prime vertex is child of a prime vertex, or is the root. A tree whose some (resp., all) vertices are unsigned (resp., signed) is called *ambiguous* (resp., *unambiguous*).

In the previous section, quotient permutations have been defined classically without a consideration about signs. In the following we need quotient permutations to be signed—fully or partially, as follows: Let I be a vertex of $T_{\mathcal{S}}(P)$, and $\{I_1, \ldots, I_k\}$ its children, the sign of the element $i$ of $P_I$ is either the sign of its corresponding node $I_i$, if $I_i$ is signed, or is not defined. Therefore, if $T_{\mathcal{S}}(P)$ is unambiguous, each quotient permutation defined from a vertex of $T_{\mathcal{S}}(P)$ is fully signed, otherwise, if $T_{\mathcal{S}}(P)$ is ambiguous, some quotient permutations are partially signed, and will need to be completed (see Definition 3).

From now on, we consider strong interval trees on which rules (S.1), (S.2) and (S.3) have been applied and signed quotient permutations.

It was shown in [1] that, for a given signed permutation $P$, if $T_{\mathcal{S}}(P)$ is unambiguous, then Algorithm 1 computes a parsimonious perfect scenario for $P$ in worst-case time $O(n\sqrt{n\log(n)})$, while Algorithm 2 can handle the case where $T_{\mathcal{S}}(P)$ is ambiguous in $O(2^p n\sqrt{n\log(n)})$ worst-case time, where $p$ is the number of unsigned prime vertices in $T_{\mathcal{S}}(P)$.

**Definition 1.** A *completion* of an ambiguous strong interval tree $T$ is an unambiguous strong interval tree obtained by giving signs to the unsigned vertices of $T$.

The correctness of Algorithm 2 (see [1, Theorem 4]) induces the following obvious lemma, that we state for the sake of completeness, as it will serve as a basis for the invariant of the algorithm we describe in Section 3.2.

**Definition 2.** A *completion* of an ambiguous strong interval tree $T$ is said to be *parsimonious* if the resulting perfect scenario computed with Algorithm 1 is a parsimonious perfect scenario among all completions of $T$.

$S$ is an empty scenario.
**For each** prime vertex $I$ of $T_{\mathcal{S}}(P)$ **Do**
  $P_I$ is the quotient permutation of $I$ over its children
  **If** the sign of $I$ is positive **Then**
    Compute a parsimonious scenario $S'$ from $P_I$ to $Id$
  **Else**
    Compute a parsimonious scenario $S'$ from $P_I$ to $\overline{Id}$
  **End if**
    Add to $S$ the sequence of reversals obtained by replacing in $S'$ every element by the corresponding interval in $P$
**End for**
  Add to $S$ the linear vertices and leaves having a linear parent and a sign different from the sign of their parent.

Algorithm 1. Computing a parsimonious perfect scenario for $T_{\mathcal{S}}(P)$, when $T_{\mathcal{S}}(P)$ is unambiguous.

**For each** of the $2^p$ completions of $T_{\mathcal{S}}(P)$ **Do**
  Apply Algorithm 1 on the resulting unambiguous tree.
**End for**
  Return a parsimonious scenario among the resulting $2^p$ perfect scenarios.

Algorithm 2. Computing a parsimonious perfect scenario for $T_{\mathcal{S}}(P)$ with $p$ unsigned vertices.

**Lemma 1.** *Let $T_{\mathcal{S}}(P)$ be the strong interval tree of a signed permutation $P$, signed according to rules* (S.1)*,* (S.2) *and* (S.3)*. There exists a parsimonious completion of $T_{\mathcal{S}}(P)$.*

### 3.2. A more efficient algorithm

The difficulty in computing a parsimonious perfect scenario then relies on non-root vertices $I$ that are not signed. Indeed such a vertex $I$, whose parent $J$ is prime by definition, having no sign implies that (1) we do not know if $P_I$ has to be sorted towards the identity or the

Please cite this article in press as: S. Bérard et al., A more efficient algorithm for perfect sorting by reversals, Information Processing Letters (2007), doi:10.1016/j.ipl.2007.10.012

reversed identity and (2) the quotient permutation $P_J$ of $J$ is not fully signed. The elements of $P_J$ corresponding to the unsigned children of $J$ are unsigned. Following Algorithm 2, computing a parsimonious perfect scenario consists of completing accurately an ambiguous strong interval tree and thereby the corresponding partially signed quotient permutation.

The principle of the new algorithm we propose is to detect patterns, in an ambiguous $T_S(P)$, that can be signed in such a way that the resulting completion can be extended into a parsimonious completion. The key notion to define such patterns is a classification of partially signed permutations.

**Definition 3.** Let $P$ be a partially signed permutation on $\{1, 2, \ldots, n\}$. A signed permutation $P'$ is a *completion* of $P$ if it is obtained from $P$ by giving signs to all its unsigned elements.

**Definition 4.** (1) Let $P$ be a partially signed permutation. We denote by $d^+(P)$ (resp., $d^-(P)$) the minimum, over all completions $P'$ of $P$, of the reversal distance to sort $P'$ into $Id$ (resp., $\overline{Id}$). We denote by $P^+$ (resp., $P^-$) a completion of $P$ whose reversal distance to $Id$ (resp., $\overline{Id}$) is minimum.

(2) If $d^+(P) > d^-(P)$, $P$ is said to be *negative*. If $d^+(P) < d^-(P)$, $P$ is said to be *positive*. If $d^+(P) = d^-(P)$, $P$ is said to be *neutral*.

(3) We extend naturally the notions defined in (1) and (2) to a vertex $I$ of $T_S(P)$ by considering its quotient permutation $P_I$.

**Remark 1.** The classification introduced in Definition 4 has been introduced in [8]. It was shown there that for an ambiguous strong interval tree, there always exists a parsimonious completion in which every positive vertex (resp. negative) received a sign $+$ (resp., $-$). However the signs defined by rules (S.1), (S.2) and (S.3) can contradict the signs given by Definition 4, as rule (S.3) will sign positively a negative prime vertex whose parent is linear increasing. So our algorithm is different in nature of the algorithm described in [8], as it uses first the strong interval tree, then Definition 4.

The general idea of our algorithm is as follows: when there is a path of consecutive prime vertices that are unsigned and neutral, no decision on the signs of these vertices can be taken, but as soon as such a path ends up on an ancestral non-neutral or signed vertex, then sign decision can be made for the whole path.

Let $\star$ be the sign ($+$ or $-$) of $I$
**For each** child $J$ of $I$ **Do**
  $J$ is given the sign of its associated element of $P_I^\star$
  Propagate the signs in the subtree rooted in $J$
**End for**

Algorithm 3. Propagating signs in a compact subtree of $T$ rooted at a signed vertex $I$.

**Definition 5.** Let $T$ be an ambiguous strong interval tree. A subtree of $T$ rooted at a vertex $I$ is *compact* if all unsigned descendants of $I$ are neutral and linked to $I$ by a path of unsigned neutral vertices.

We now introduce a recursive procedure (Algorithm 3) that propagates signs in a compact subtree whose root is signed. As proved in Lemmas 2, 3 and 4 below, Algorithm 3 computes a parsimonious completion of the input compact subtree.

**Lemma 2.** *Let $T$ be an ambiguous strong interval tree. If $T$ is compact and its root $I$ is unsigned and neutral then giving either sign ($+$ or $-$) to $I$ and propagating the signs in $T$ using Algorithm* 3 *produces a parsimonious completion of $T$.*

**Proof.** We proceed by induction on the maximum length of a path of unsigned neutral vertices starting at $I$. As by assumption $T$ is compact, all unsigned vertices are neutral (see Definition 5).

First assume that all these paths have length 1. Then the only signs that have to be added to complete $T$ are on $I$ and its unsigned children. As all these vertices are neutral, then sorting each of them into $Id$ or $\overline{Id}$ makes no difference on the perfect reversal distance. Thus $I$ can be arbitrarily signed and propagating its sign to its children will produce a parsimonious completion of $T$.

Now assume that the longest path of unsigned neutral vertices starting at $I$ has length $k > 1$. As $I$ is neutral, $P_I$ can be sorted to either $Id$ or $\overline{Id}$ using the same minimum number of reversals. Assume without loss of generality that $I$ is sorted to $Id$ (i.e., given sign $+$) and that sign has been propagated to its descendants. It follows that its unsigned children have been signed according to $P_I^+$. By induction, this choice for every unsigned child of $I$ was parsimonious, which shows that the completion of $T$ obtained by giving sign $+$ to $I$ is parsimonious. □

**Lemma 3.** *Let $T$ be an ambiguous strong interval tree. If $T$ is compact and its root is signed, then propagating the signs in $T$ using Algorithm* 3 *produces a parsimonious completion of $T$.*

Please cite this article in press as: S. Bérard et al., A more efficient algorithm for perfect sorting by reversals, Information Processing Letters (2007), doi:10.1016/j.ipl.2007.10.012

**Proof.** The proof follows immediately from Lemma 2. Assume, without loss of generality, that $I$ has sign $+$. Using Lemma 2 on the unsigned children of $I$, that are all unsigned neutral vertices that root compact subtrees, we can say that the sign that they are given does not matter in terms of parsimony: the number of reversals obtained from the corresponding subtrees with Algorithm 1 is parsimonious. Hence, we only have to ensure that the number of reversals used to sort the completion of $P_I$ to $Id$ (as $I$ has sign $+$) is minimal, which follows from the definition of $P_I^+$. □

**Lemma 4.** *Let $T$ be an ambiguous strong interval tree. If $T$ is compact and its root $I$ is unsigned and positive (resp., negative), then signing $I$ with $+$ (resp., $-$) and propagating the signs in $T$ using Algorithm* 3 *produces a parsimonious completion of $T$.*

**Proof.** For the same reason than in the proof of Lemma 3, the signs given to the children of $I$ do not prevent from completing $T$ in a parsimonious completion. Assume now, that $I$ is positive. To ensure the parsimony of the scenario that will be computed from the completed tree using Algorithm 1, we want that $P_I$ is completed in such a way that it is sorted using a minimum number of reversals. As $I$ is positive, this implies that $P_I$ has to be sorted to $Id$ and can be completed to $P_I^+$, which concludes the proof. □

**Lemma 5.** *Let $T$ be an ambiguous strong interval tree. Let $I$ be an unsigned positive (resp., negative) vertex of $T$ that roots a compact subtree of $T$. Then signing $I$ with $+$ (resp., $-$) and propagating the signs in the subtree it roots using Algorithm* 3 *produces a tree that can be completed parsimoniously.*

**Proof.** The case where $I$ is the root was proved in Lemma 4. Now assume that $I$ is not the root, is unsigned and positive (the case where $I$ is negative is symmetric). From Lemma 4, the signs given to the unsigned vertices belonging to the subtree rooted at $I$ complete parsimoniously this subtree. It then remains to show that signing $I$ with $+$ does not prevent to complete $T$ parsimoniously. Let $J$ denote the parent of $I$ and $P_J$ its quotient permutation. Assume that a parsimonious completion requires that the element of $P_J$ corresponding to $I$ receives sign $-$. So if $I$ has been given sign $+$, it will cost one extra reversal to sort $P_J$, for example, reverting the element corresponding to $I$. But as $I$ is positive, sorting $I$ towards $Id$ instead of towards $\overline{Id}$ saves one reversal. Thereby $I$ could as well received sign $+$ and this would not prevent a parsimonious completion of $T$. □

```
Traverse T_S(P) using a post-order traversal and
  For each prime vertex I, visited for the last time
    If I is not signed Then
      Compute all completions of P_I,
          d+(P_I), d−(P_I), P_I^+ and P_I^−.
      If I is positive (d+(P_I) < d−(P_I)), Then
          Sign I with +
      Else If I is negative (d+(P_I) > d−(P_I)) Then
          Sign I with −
      Else If I is the root Then
          Sign I with an arbitrary sign
      Else store with I the permutations P_I^+ and P_I^−.
      End if
    End if
    If I is signed Then
      Propagate signs from I using Algorithm 3
    End if
  End for
Apply Algorithm 1 on the resulting unambiguous tree.
```

Algorithm 4. Computing a parsimonious perfect scenario for $T_S(P)$.

Algorithm 4 computes a parsimonious perfect scenario. The principle is to traverse $T_S(P)$ and to give signs to the unsigned vertices using the rules described in Lemmas 2, 3, 4 and 5, before using Algorithm 1 on the resulting unambiguous tree.

**Theorem 1.** *Let $P$ be a signed permutation on $\{1, 2, \ldots, n\}$ and $T_S(P)$ its strong interval tree. Let $d$ be the maximum prime degree of $T_S(P)$. Algorithm* 4 *computes a parsimonious perfect scenario for $P$ in space $O(n^2)$ and time $O(2^d n\sqrt{n \log(n)})$.*

**Proof.** We first prove that Algorithm 4 computes a parsimonious perfect scenario for $P$. We will show that an invariant of the algorithm is that, at any time, the tree can be completed parsimoniously and that at the end, the tree is unambiguous.

Due to the post-order traversal, it is immediate that when signs are given to unsigned vertices, using the propagation from the root of a subtree, this subtree is compact. This implies that after signs have been propagated from a vertex $I$, then the whole subtree rooted at $I$ is unambiguous. Hence the tree is unambiguous at the end of the traversal.

From Lemma 1, at the beginning, $T_S(P)$ can be completed parsimoniously. Now assume that the current tree can be completed parsimoniously when signs are given to some unsigned vertices, by propagating them from a vertex $I$ that roots a compact subtree. This can happen only when $I$ is the root, or is signed or is not neutral. If $I$ is the root, either signed or unsigned, then Lemmas 2, 3 and 4 ensure that the resulting tree is completed parsimoniously. Assume that $I$ is not the root and is signed.

Please cite this article in press as: S. Bérard et al., A more efficient algorithm for perfect sorting by reversals, Information Processing Letters (2007), doi:10.1016/j.ipl.2007.10.012

It follows from Lemma 3 that the signs given in the subtree rooted in $I$ complete parsimoniously this subtree. As $I$ was signed and the tree could be completed parsimoniously this ensures that this property still holds after propagating signs from $I$. Finally the case where $I$ is unsigned and positive or negative follows from Lemma 5.

The space complexity follows from the fact that with each unsigned neutral vertex $I$ of $T_{\mathcal{S}}(P)$, we store $P_I^+$ and $P_I^-$ in order to be able to propagate the signs without having to re-compute these two completions of $P_I$. The time complexity follows from (1) there are $O(n)$ vertices in $T_{\mathcal{S}}(P)$, (2) the fact that for every unsigned vertex $I$, we have to compute all possible completions of $P_I$ to decide if $I$ is positive, negative or neutral, and (3) the fact that the most efficient known algorithm to sort a signed permutation by reversals has a $O(n\sqrt{n\log(n)})$ worst-case time complexity [10]. □

## 4. Conclusion

Our main result in this note is an algorithm that computes a parsimonious perfect scenario for a signed permutation more efficiently that the algorithm described in [1]. A very similar use of the notion of positive, negative and neutral permutations was used in [8], but in a framework that did not take advantage of the structure provided by the strong interval tree, in particular to propagate signs. This property allows to both describe a simpler algorithm and have a better understanding of its complexity.

The main algorithmical problem we face for perfect sorting by reversals, in the framework using the strong interval tree, is to decide, for every unsigned vertex $I$, independently of the other vertices, if this vertex is positive, negative or neutral. The solution we used in our algorithm involves trying all completions of $P_I$. Any advances on this problem of giving signs to a partially signed permutation in order to minimize the reversal distance could then be used immediately in our algorithm. However, based on the fact that sorting unsigned permutations by reversals is NP-hard [4], it is very likely that this problem is hard too.

Otherwise, aside of some recent works on the class of signed permutations that has a parsimonious scenario that is also perfect [7], it seems difficult to optimize the strong interval tree framework in order to compute perfect scenarios in polynomial time for larger classes of signed permutations.

## References

[1] S. Bérard, A. Bergeron, C. Chauve, C. Paul, Perfect sorting by reversals is not always difficult, IEEE/ACM Trans. Comput. Biology Bioinform. 4 (1) (2007) 4–16.

[2] A. Bergeron, C. Chauve, F. de Montgolfier, M. Raffinot, Computing common intervals of $K$ permutations, with applications to modular decomposition of graphs, in: Proc. 13th Annual European Symposium on Algorithms, in: Lecture Notes in Comput. Sci., vol. 3669, Springer, 2005, pp. 779–790.

[3] B.-M. Bui-Xuan, M. Habib, C. Paul, Revisiting T. Uno and M. Yagiura's algorithm, in: Proc. 16th International Symposium on Algorithms and Computation, in: Lecture Notes in Comput. Sci., vol. 3827, Springer, 2005, pp. 146–155.

[4] A. Caprara, Sorting permutations by reversals and Eulerian cycle decompositions, SIAM J. Discrete Math. 12 (1) (1999) 91–110.

[5] C. Chauve, M. Mishna, Average-case analysis of perfect sorting by reversals, 2007.

[6] R.G. Downey, M.R. Fellows, Parameterized Complexity, Springer, 1999.

[7] Y. Diekmann, M.-F. Sagot, E. Tannier, Evolution under reversals: parsimony and conservation of common intervals, IEEE/ACM Trans. Comput. Biology Bioinform. 4 (2) (2007) 301–309.

[8] M. Figeac, J.-S. Varré, Sorting by reversals with common intervals, in: Proc. 4th International Workshop on Algorithms, in: Bioinformatics, in Lecture Notes in Bioinformatics, vol. 3240, Springer, 2004, pp. 26–37.

[9] O. Gascuel (Ed.), Mathematics of Evolution and Phylogeny, Oxford University Press, 2005.

[10] E. Tannier, A. Bergeron, M.-F. Sagot, Advances on sorting by reversals, Discrete Applied Math. 155 (6–7) (2007) 881–888.

Please cite this article in press as: S. Bérard et al., A more efficient algorithm for perfect sorting by reversals, Information Processing Letters (2007), doi:10.1016/j.ipl.2007.10.012