Exploring the Solution Space of Sorting by Reversals, with Experiments and an Application to Evolution

Marília D.V. Braga, Marie-France Sagot, Celine Scornavacca, and Eric Tannier

Abstract—In comparative genomics, algorithms that sort permutations by reversals are often used to propose evolutionary scenarios of large-scale genomic mutations between species. One of the main problems of such methods is that they give one solution while the number of optimal solutions is huge, with no criteria to discriminate among them. Bergeron et al. started to give some structure to the set of optimal solutions, in order to be able to deliver more presentable results than only one solution or a complete list of all solutions. The structure is a way to group solutions into equivalence classes, and to identify in each class one particular representative. However, the design of an algorithm to compute this set of representatives without enumerating all solutions was stated to be an open problem. We propose, in this paper, an answer to this problem, that is, an algorithm which gives one representative for each class of solutions and counts the number of solutions in each class, with a better theoretical and practical complexity than the complete enumeration method. We give an example of how to reduce the number of equivalence classes obtained, using further constraints. Finally, we apply our algorithm to analyze the possible scenarios of rearrangements between mammalian sex chromosomes.

Index Terms—Computational biology, genome rearrangements, signed permutations, sorting by reversals, common intervals, perfect sorting, evolution.

1 Introduction

The combinatorics of genome rearrangements is a very prolific domain of computational biology. It consists of, given a set of actual genomes, inferring the large-scale evolutionary mutations that explain the differences in the organization of these genomes. For a general survey of the algorithmic aspects of genome rearrangements, see [17].

One of the most used mathematical models for representing and manipulating such genome rearrangements is given by signed permutations, where the elements are homologous markers, and by reversals as the main events that may alter the order of the markers along the genomes. The combinatorial problem consists of giving the shortest sequence of reversals that transforms one permutation into another. The problem of sorting signed permutations by reversals has been the subject of a huge literature (among others, [2], [7], [11], [14], [15], [24]), but all algorithms propose one optimal solution, whereas the solutions can be very numerous. This kind of delivery may be useless for biological purposes, and the algorithms are therefore mainly useful to compute a distance between genomes.

One study by Siepel [20] resulted in a method to enumerate all solutions. This is, however, almost as useless as providing only one solution, because, often, the solutions are so many that the whole set cannot be presented (when it can be computed). A few studies tried to decrease the size of the set of optimal solutions by introducing some biological constraints, such as favoring small inversions [1], or inversions that do not cut some clusters of colocalized genes[1] [11], [4]. The number of solutions is decreased, but the whole set of solutions is never handled.

Bergeron et al. [5] then provided a way to group the solutions into equivalence classes. However, no algorithmic study was performed, and in particular, the problem of giving one element in each class without enumerating all the solutions was mentioned open. In this paper, we introduce a solution to this problem. Our solution gives one representative element per class of solutions and counts the number of solutions in each class. The complete enumeration of the solutions is not needed, and the theoretical complexity, as well as the practical execution time are lower than in any other current method for the enumeration of the solutions.

We show two examples of possible use of such a method. First, we consider the criterion of colocalized genes mentioned in [4] and [11], and show that the algorithm

- *M.D.V. Braga is with the Laboratoire de Biométrie et Biologie Évolutive (UMR 5558), Université de Lyon 1 (Claude Bernard), 43, Bd du 11 Novembre 1918, F-69622 Villeurbanne, France. E-mail: marilia@biomserv.univ-lyon1.fr*
- *M.-F. Sagot and E. Tannier are with the Laboratoire de Biométrie et Biologie Évolutive (UMR 5558), Université de Lyon 1 (Claude Bernard), 43, Bd du 11 Novembre 1918, F-69622 Villeurbanne, France and INRIA Rhône-Alpes, 622 avenue de l'Europe, 38334 Montbonnot, France. {marie-france.sagot, eric.tannier}@inria.fr.*
- *C. Scornavacca is with the Laboratoire d'Informatique, de Robotique et de Microéletronique de Montpellier, Université de Montpellier 2, 161, rue Ada, 34392 Montpellier cedex 5, France. E-mail: celine.scornavacca@lirmm.fr.*

Manuscript received 4 Aug. 2007; revised 4 Dec. 2007; accepted 10 Jan. 2008; published online 22 Feb. 2008.
For information on obtaining reprints of this article, please send e-mail to: tcbb@computer.org, and reference IEEECS Log Number TCBBSI-2007-08-0094.
Digital Object Identifier no. 10.1109/TCBB.2008.16.

1. Clusters of colocalized genes are intervals of the genomes composed by the same genes but not necessarily in the same order and orientations. Formal definitions and their possible usage in the present context are provided in Section 4.2.

we propose not only reduces the space of solutions of sorting by reversals (this was the motivation of [4] and [11]) but also reduces the number of classes of solutions, showing the interest of applying this criterion together with the classification of solutions presented in this paper. As a second example, we applied our algorithm to analyze the possible scenarios of rearrangements that differentiate mammalian X and Y sex chromosomes. These chromosomes are very different, but are believed to have evolved from an identical pair of chromosomes [18]. Current theories claim that reversals have an important role to explain the differences observed between the X and Y chromosomes [19], [21]. In [19], one scenario of reversals is proposed, whereas the studied permutation allows thousands of solutions. Using the possibilities provided by our algorithm, we were able to analyze the whole set of solutions, and to evaluate the proposed sequence of reversals to explain the evolution of the human X and Y chromosomes.

This paper is an extended version of a previously published one [8] and is organized as follows: We present the usual model for dealing with gene order and orientation in Section 2. In Section 3, we describe the algorithm. Section 4 is dedicated to practical experiments on simulated and biological data, including the X and Y chromosomes, and to an analysis of the performance of our implementation.

2 Sorting by Reversals and its Solution Space

2.1 Signed Permutations

Genome rearrangements such as reversals may change the order of some segments in a genome, and also the DNA strand the segment is on. We identify homologous genomic markers with the integers $1, \ldots, n$, with a plus or minus sign to indicate the strand they lie on. The order and orientation of genomic markers of one species in relation to another is represented by a *signed permutation* of size n, that is, a bijection π over $\{-n, \ldots, -1, 1, \ldots, n\}$, such that $\pi_{-i} = -\pi_i$, where π_i is the image of i by π. A signed permutation is usually written using the list of elements $(\pi_1, \ldots, \pi_n)$. The *identity permutation* $(1, \ldots, n)$ is denoted by Id.

A subset of numbers $\rho \subseteq \{1, \ldots, n\}$ is said to be an *interval* of a permutation π if there exist $i, j \in \{1, \ldots, n\}$, $1 \leq i \leq j \leq n$, such that $\rho = \{|\pi_i|, \ldots, |\pi_j|\}$. Two intervals are said to *overlap* if they intersect but none is contained in the other. For example, if π is (1, −3, −5, 2, 4, −6), then $\rho_1 = \{1, 3, 5\}$ and $\rho_2 = \{2, 3, 5\}$ overlap, while $\rho_3 = \{2, 3, 4, 5\}$ and $\rho_4 = \{2, 5\}$ do not.

2.2 Sorting by Reversals

Given a permutation π and an interval ρ of π, we can apply a *reversal* on π, that is, the operation which reverses the order and flips the signs of the elements of ρ: if $\rho = \{|\pi_i|, \ldots, |\pi_j|\}$,

$$\pi \cdot \rho = (\pi_1, \ldots, \pi_{i-1}, -\pi_j, \ldots, -\pi_i, \pi_{j+1}, \ldots, \pi_n).$$

Due to this, an interval ρ can also be used to denote a reversal. For example, if the permutation π is (4, −3, −1, 2) and $\rho = \{1, 2\}$, then $\pi \cdot \rho = (4, -3, -2, 1)$. We say that a sequence of reversals $\rho_1 \ldots \rho_k$ *sorts* a permutation π if ρ_i is an interval of $\pi \cdot \rho_1 \cdots \rho_{i-1}$ for all i, and $\pi \cdot \rho_1 \cdots \rho_k = Id$. The length of the shortest sequence of reversals sorting a permutation π is called the *reversal distance* of π, and is denoted by $d(\pi)$. The shortest sequence of reversals sorting π is called an *optimal* sorting sequence. For example, if the permutation π is (4, −3, −1, 2), one optimal sorting sequence is {1}{1, 2}{4}{1, 2, 3, 4}.

Computing the reversal distance and finding an optimal sorting sequence has been the topic of a huge literature. The first polynomial algorithm to find an optimal sorting sequence appeared in [15]. The fastest algorithm to compute the distance was given in [2]; the fastest way to find an optimal sequence can be retrieved from a compilation of [7], [14], and [24].

However, all these studies give one sequence among possibly many. For example, for the permutation (−12, 11, −10, 6, 13, −5, 2, 7, 8, −9, 3, 4, 1), the number of solutions is 8,278,540, and it can be useless when attempting a biological interpretation to know only one among them.

The set of all solutions may be retrieved thanks to an algorithm by Siepel [20], that, given a permutation, computes all the reversals that are the first step of an optimal sequence, but in the aforementioned example, listing the 8,278,540 sequences is almost as useless as giving only one of them.

2.3 Traces

More interesting for our study is the representation of the set of solutions that is given in [5]. Let s be a sequence of reversals, and ρ and θ be two reversals of s, which appear consecutively in s. The operation of *commutation* of ρ and θ in s consists of replacing the sequence $\rho\theta$ by $\theta\rho$. Two sequences are said to be *equivalent* if one can be obtained from another by a sequence of commutations of nonoverlapping reversals.

Observe that if a sequence of reversals is a sorting sequence for a permutation π, every equivalent sequence is also a sorting sequence, because if two reversals do not overlap, it does not matter in which order they are applied.

For example, if the permutation π is (4, −3, −1, 2), consider the solution given by the sequence of reversals {1}{1, 2}{4}{1, 2, 3, 4}. Here, {4} and {1, 2} do not overlap, so {1}{1, 2}{4}{1, 2, 3, 4} is equivalent to {1}{4}{1, 2}{1, 2, 3, 4}. By the way, all permutations of these four reversals can be obtained from one another by a sequence of commutations and are, therefore, equivalent.

For the same permutation, if now we consider the solution {1, 3, 4}{2, 4}{2, 3}{3}, it is equivalent, by commutation of {3} with all the other reversals, to {1, 3, 4}{2, 4}{3}{2, 3}, {1, 3, 4}{3}{2, 4}{2, 3}, and {3}{1, 3, 4}{2, 4}{2, 3}. There is, in this case, no other equivalent sequence, as the other pairs of consecutive reversals all overlap.

An equivalence class of optimal sequences of reversals over this equivalence relation is called a *trace*. The concept of traces is well studied in combinatorics, see, for example, [10]. It is particularly relevant in our study because of a result proven in [5], which states that the set of all optimal sequences of reversals sorting a signed permutation is a union of traces.

As a consequence, if the set of solutions is too big to be enumerated, the set of traces may be a more relevant result for the problem of sorting by reversals. It remains to find a good way to represent the traces in a compact manner.

2.4 Normal Form of a Trace

A trace T is, thus, a set of equivalent sequences of subsets of $\{1, \ldots, n\}$. Two subsets of $\{1, \ldots, n\}$ may be compared by writing their elements in increasing lexicographic order, and comparing the two obtained sequences.

A sequence s of T is said to be in *normal form* if it can be decomposed into substrings[2] $s = u_1| \ldots |u_m$ such that

- every pair of elements of a substring u_i is nonoverlapping,
- for every element ρ of a substring u_i $(i > 1)$, there is at least one element θ of the substring u_{i-1} such that ρ and θ overlap, and
- every substring u_i is increasing according to the lexicographic order.

A theorem by Cartier and Foata (cited in [5]) states that, for any trace, there is a unique element that is in the normal form. We may, therefore, represent a trace by its element in the normal form.

For example, the permutation (4, −3, −1, 2) has two traces of optimal sequences, one is {1}{1, 2}{1, 2, 3, 4}{4}, and the other is $\{1, 3, 4\}\{3\}|\{2, 4\}|\{2, 3\}$. In this example, giving the two normal forms of the traces allows to describe the whole set of 28 solutions in a compact way.

2.5 The Algorithmics of Traces

Bergeron et al. [5] provide no algorithmic insight for this way of representing the solutions of sorting by reversals. They state, as an open problem, the complexity of giving one element in each trace. The best algorithm so far to enumerate the traces is, therefore, to do a complete enumeration of all the solutions, and from each solution, to compute the associated trace and add it to the list of found traces if it is not already in it.

We give, in the next section, an algorithm that enumerates the normal form of all the traces of solutions given a signed permutation, and counts the number of solutions in each trace, without enumerating all the solutions.

3 The Algorithm and its Complexity

For comparison purposes, it will be useful to describe first the only available algorithm that is up to now able to enumerate all the traces of the solution space of sorting by reversals, and to examine its theoretical complexity. Then, we describe our algorithm and achieve some comparisons between the two, both from the point of view of theoretical complexity and implementation behavior. All complexities are functions of n, the size of the permutation, which is the size of the input, and N, the number of traces of optimal solutions, which is the size of the output. The reversal distance is bounded by n, as in a permutation without adjacencies; it always has order n (see, for example, [15]).

2. The "substrings" are all *contiguous* subsets of the sequence of reversals.

3.1 Enumeration of the Solutions

A sequence of reversals $s = \rho_1\rho_2 \ldots \rho_i$ is called an *optimal i-sequence* if $d(\pi \cdot \rho_1 \cdots \rho_i) = d(\pi) - i$. Note that if $i = d(\pi)$, then s is an optimal sorting sequence.

The set of all optimal 1-sequences of a permutation can be computed with the help of an algorithm by Siepel [20]. It has time complexity $O(n^3)$, and the number of possible optimal 1-sequences is bounded by n^2.

The set of all optimal i-sequences can then be computed from the set of optimal $(i-1)$-sequences by iterating the same algorithm for finding all 1-sequences. The set of optimal i-sequences has, therefore, size of at most $O(n^{2i})$, and such an algorithm has time complexity of at most $O(n^3 * \sum_{k=1}^{i} n^{2k})$. In this way, we can enumerate the set of all optimal sorting sequences in time $O(n^{2n+3})$.

There remains to group the sorting sequences by trace, and to construct the normal form of each trace.

For any optimal i-sequence s of reversals, and under the equivalence relation deduced from the commutation of reversals, is defined the trace that contains s, that we call an *i-trace*.

Given an optimal sorting sequence $s = \rho_1\rho_2 \ldots \rho_d$ for a permutation π with reversal distance d, the normal form of the trace T that contains s is constructed by iteratively adding the elements ρ_i, $1 \leq i \leq n$, to the normal form f of the $(i-1)$-trace containing the sequence $\rho_1 \ldots \rho_{i-1}$. This adding procedure is represented by $f + \rho_i$ and described by Algorithm 1.

```
Algorithm 1: Adding an element $\rho_i$ to a normal form $f$ of an $(i-1)$-trace: $f + \rho_i$
Input: The normal form $f = u_1|u_2| \ldots |u_k$ of an $(i-1)$-trace
    and the next element $\rho_i$
Output: The normal form of the $i$-trace containing the
    sequence $u_1 u_2 \ldots u_k \rho_i$
    Let $j$ be the maximum index such that $u_j$ contains an
    element that overlaps with $\rho_i$, or 0 if such a $u_j$ does not
    exist
    if $j = k$ then
        Add a new substring $u_{k+1} \leftarrow \rho_i$
    else
        Add $\rho_i$ to the substring $u_{j+1}$, according to the
        lexicographic order
    end if
```

As the reversal distance and the interval sizes are bounded by n, the procedure has complexity $O(n^2 \log n)$, considering that the elements of each reversal have to be sorted, and comparing reversals may be done in $O(n)$.

The constructed solution is compared to a list of already constructed normal forms of traces, so that one trace is not written several times. This may take $O(n^2 \log N)$ operations, where $O(n^2)$ is the size of a trace and N is the number of found traces. As N is bounded by the number of solutions, we have $n^2 \log N \leq n^2 \log(n^{2n}) = 2n^3 \log n$.

Eventually, the total time complexity for enumerating all the normal forms of the traces is bounded by

$$O(n^{2n+3}) + O(n^{2n}(n^2 \log n + 2n^3 \log n)) = O(n^{2n+3} \log n).$$

This upper bound on the theoretical complexity does not give hope that this method can be applied to big permutations. We shall actually see in practice that it is intractable for permutations π above around $d(\pi) = 10$.

This method can be implemented with the help of the GRAPPA software,[3] which contains an implementation of Siepel's algorithm, and it is the only one that, among all available applications about sorting by reversals, is able to give more than one unique solution.

3.2 Enumeration of the Traces

We now come to the description of the method to enumerate all the normal forms of the traces of solutions.

A k-trace T' is a *prefix* of an i-trace T $(k \leq i)$ if each k-sequence of T' is a prefix of an i-sequence of T. It is equivalent [10] to saying that T' contains a k-sequence which is a prefix of an i-sequence of T.

The idea of the algorithm to enumerate the traces is almost naturally contained in this notion. It is easy to see that every prefix of size k of an optimal i-sequence is in a k-trace of optimal k-sequences. So, instead of enumerating all the i-sequences and then computing and comparing the traces, it is therefore more valuable to enumerate and compare directly all the i-traces.

We have seen in Algorithm 1 a way to construct the normal form of an i-trace from the one of an $(i-1)$-trace. We may use this method to construct all i-traces simultaneously in an incremental way, without computing all the solutions. With no additive cost, we also compute the number of sequences in each i-trace.

The method is detailed in Algorithm 2.

Algorithm 2: Enumerating all the traces of a signed permutation
Input: A signed permutation π
Output: The normal form and size (f, s) of each trace of optimal sequences of reversals sorting π

```
$d \leftarrow$ reversal distance of $\pi$
$\mathcal{T} \leftarrow \emptyset$
$S \leftarrow \{\rho | \rho$ is an optimal 1-sequence for $\pi\}$ [Siepel [20]]
  for each reversal $\rho \in S$ do
  insert $(\rho, 1)$ in $\mathcal{T}$ [each first reversal is a 1-trace]
end for
  for each integer $i$ from 2 to $d$ do
    $\mathcal{T}' \leftarrow \emptyset$[contains the normal forms/sizes of all the
          $i$-traces]
    for each $(f, s)$ in $\mathcal{T}$ [$(f, s)$ represents an $(i-1)$-trace]
        do
      $\pi_f \leftarrow \pi \cdot f$[apply the $(i-1)$-sequence $f$ to $\pi$]
      $S \leftarrow \{\rho | \rho$ is an optimal 1-sequence for $\pi_f\}$ [Siepel [20]]
      for each reversal $\rho \in S$ do
        $f_\rho \leftarrow f + \rho$ [Algorithm 1]
        if there is $(f', s') \in \mathcal{T}'$ such that $f' = f_\rho$ then
          $s' \leftarrow s' + s$ [upd. the size of the $i$-trace repr. by $f'$]
        else
          insert $(f_\rho, s)$ in $\mathcal{T}'$ [$(f_\rho, s)$ represent an $i$-trace]
        end if
      end for
    end for
    $\mathcal{T} \leftarrow \mathcal{T}'$
  end for
  return $\mathcal{T}$ [$\mathcal{T}$ is the final set of $d$-traces]
```

Theorem 1. *At the end of Algorithm 2, $\mathcal{T}$ contains, for every trace T of solutions for sorting π, one element of T (the normal form) and the number of solutions in T.*

Proof. The proof is by induction. We prove that at the end of step i of the main loop of Algorithm 2, the set $\mathcal{T}$ contains all the normal forms and the size of the i-traces of optimal sequences for π.

For $i = 1$, each 1-trace is generated and the size of a 1-trace is 1.

For an arbitrary $2 \leq i \leq d(\pi)$, by hypothesis, $\mathcal{T}$ contains all the normal forms and the size of the optimal $(i-1)$-traces. Every i-trace has a prefix in this set, since a prefix of size $i-1$ of an optimal i-sequence is an optimal $(i-1)$-sequence. So, every i-trace is found from an $(i-1)$-trace by adding a 1-sequence.

Now, it remains to prove that the cardinality of an i-trace T is the sum of the cardinalities of its $(i-1)$-prefixes, so that the right size of all traces is computed. Let $\rho_1, \ldots, \rho_k$ be the reversals that are in the last position of at least one sequence in T. Let x_j be the number of elements of T which have ρ_j as their last position. Then, the number of sequences of T is $\sum_j x_j$. Now, for all j, as ρ_j is the last reversal of an optimal i-sequence $x_1 \ldots x_{i-1}\rho_j$ of T, $x_1 \ldots x_{i-1}$ is an optimal $(i-1)$-sequence of reversals, so it belongs to an $(i-1)$-trace T' of size x_j. By the induction hypothesis, the size of the trace T is therefore the sum of the sizes of all $(i-1)$-prefixes of T, and the algorithm provides this size, since it generates all prefixes. □

3.3 Theoretical Complexity

The complexity of the algorithm depends on the number $\sum_{i=1}^{d(\pi)} n(i)$, where $n(i)$ is the number of i-traces of optimal i-sequences. As every i-trace is a prefix of a d-trace, where $d = d(\pi)$, this number is bounded by the number of d-traces times the number of prefixes of each trace.

To give an estimation of the number of prefixes of a trace, we need to adopt a representation of the traces as partially ordered sets (posets). It is possible to represent a trace T that contains an optimal sequence $\rho_1 \ldots \rho_d$ by a partial ordering of the set $\mathcal{P}_T = \{\rho_1, \ldots, \rho_d\}$ (if two reversals of the sequence are equal, they should both appear in the set $\mathcal{P}_T$, differentiated by their indices in the sequence $\rho_1 \ldots \rho_d$). The relation $<_T$ is defined as the transitive closure of the relation $\lhd$, itself defined by $\rho_i \lhd \rho_j$ if and only if $i < j$ and ρ_i and ρ_j overlap.

In other words, $\rho_i <_T \rho_j$ if and only if ρ_i is always before ρ_j in the elements of T (see [10]).

For example, $T = \{1, 3, 4\}\{3\}|\{2, 4\}|\{2, 3\}$ is a trace of optimal sorting sequences of reversals for the permutation (4, -3, -1, 2). The elements of $\mathcal{P}_T$ are $\{1, 3, 4\}$, $\{3\}$ $\{2, 4\}$, and $\{2, 3\}$,

3. http://www.cs.unm.edu/~moret/GRAPPA/. We reimplemented the algorithm in Java in order to include it in the package BaobabLuna that implements all the methods that we describe here, in order to compare the running times on the same basis.

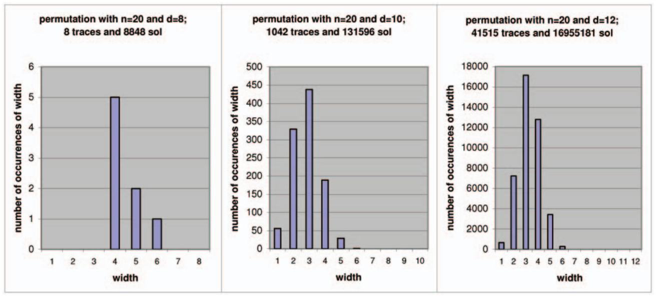

Fig. 1. Distribution of the width of posets for three random permutations of size 20 and different reversal distances ($d = 8$, $d = 10$, and $d = 12$). For each permutation, we computed the traces of optimal sequences and we calculated the number of occurrences of each width in the traces.

and the relations are $\{1, 3, 4\} <_T \{2, 4\}$, $\{2, 4\} <_T \{2, 3\}$, and $\{1, 3, 4\} <_T \{2, 3\}$.

The set $\mathcal{P}_T$ with the relation $<_T$ is a partially ordered set (poset). A *linear extension* of a poset is a total order $<_{tot}$ which satisfies $\rho <_T \theta \Rightarrow \rho <_{tot} \theta$. The set of all linear extensions of $(\mathcal{P}_T, <_T)$ is exactly the set of elements of the trace T (see [10]). We may, therefore, identify the trace T and the poset $(\mathcal{P}_T, <_T)$, and simply speak about the poset T.

The *height* of a trace (or poset) is the cardinality of the maximum set of elements of $\mathcal{P}_T$ that is totally ordered by the relation $<_T$. It is also the number of subsequences u_i in the normal form of a trace.

The *width* of a trace (or poset) is a maximum cardinality set of elements of $\mathcal{P}_T$ that are not comparable by the relation $<_T$. It is at least (but in general not equal to) the maximum size of a subsequence u_i in the normal form of a trace. The width of a poset can be computed in polynomial time thanks to a reduction of Fulkerson [13] to a bipartite matching problem.

The representation of a trace as a poset allows us to use the parameters of the poset in the computations of the complexity of the algorithms, and it is also a nice way to present the solution of sorting by reversals. Indeed, a poset corresponds to a set of reversals that may have occurred during evolution and that could, therefore, help explain the difference between the organization of two genomes. It indicates what we know and what we do not know about the order in which these potential reversals occurred. Instead of giving a list of sequences, or a unique sequence representing an equivalence class, the poset therefore gives *one* possible solution, with uncertainties as concerns the exact shape of the solution.

An *ideal* of a poset $(\mathcal{P}_T, <_T)$ is a subset U of $\mathcal{P}_T$ such that if $\rho \in U$ and $\theta <_T \rho$, then $\theta \in U$.

It is very easy to see that ideals of posets and prefixes of traces correspond to the same notions, and that in particular, the number of prefixes of a trace T is exactly the number of ideals of the poset $(\mathcal{P}_T, <_T)$.

The advantage of this notation is that the number of ideals of a poset can be estimated. It is bounded by n^k, where n is the size of $\mathcal{P}_T$ and k is the width of the poset [22].

The number of i-traces that we generate is therefore bounded by $Nn^{k_{max}}$, where N is the number of d-traces and k_{max} is the maximum width of a d-trace.

PERMUT.	N$_s$	N$_t$	Enum. sol.	Enum. + traces	Traces
random n =17 d=11	57019369	18255	$\simeq$ 4 h	$\simeq$ 4.5 h	$\simeq$ 5 min
random n =18 d=12	327905046	34317	$\simeq$ 24 h	$\simeq$ 43 h	$\simeq$ 18 min

Fig. 2. Computation results (1). Columns from left to right contain: 1) the origin of the permutation, its number of elements, and reversal distance; 2) the number of solutions of sorting this permutation by reversals; 3) the number of traces; 4) the execution time of an algorithm that enumerates all the solutions; 5) the execution time of the same program, adding the computation of all the traces from the solutions; and 6) the execution time of the enumeration of the traces, according to Algorithm 2. All algorithms are part of the BaobabLuna Package. Experiments were made on a personal computer with a 1.8-GHz CPU and 1-Gbyte random-access memory (RAM).

Given this estimation, we may give a bound for the complexity of our algorithm. Indeed, for every i-trace, $1 \leq i \leq d - 1$, we apply an $O(n^3)$ algorithm to find all the 1-sequences. For all these 1-sequences (there are at most n^2 of them), we then apply Algorithm 1 to construct the normal form of the $(i + 1)$-trace, and compare the constructed normal form to the current list of normal forms of $(i + 1)$-traces.

This gives a final complexity of

$$O(Nn^{k_{max}}(n^3 + n^2(n^2 + n\log(Nn^{k_{max}})))) = O(Nn^{k_{max}+4}).$$

Observe that computing the number of linear extensions of a poset is $\#P$-complete [9], and the best-known algorithms run in $O(n^k)$, where n is the size of the poset and k is its width [23]. Our algorithm counts the number of elements in each d-trace, that is the number of linear extensions of the associated posets. Our time complexity, thus, nearly reaches the best-known complexity for the counting part.

If, in general, the width of a poset may be as large as its number of elements, we have made some experiments on simulated permutations (see Fig. 1) which show that, in practice, this parameter is often lower, which explains the speedup of our algorithm compared to a total enumeration procedure.

4 Experimental Results and Applications

4.1 Implementation and Performance

We implemented[4] our algorithm and tested it on randomly simulated permutations to evaluate its performance.

Some results are recorded in Fig. 2. These numbers may be useful to give an idea of the quantities that we are dealing with, numbers of solutions and number of traces.

Even if we are quickly limited in the size of the permutations that it is possible to treat, there is a solid gain in relation to the existing methods. Observe that the main limit concerns the amount of memory that needs to be used, more than the time. In the next section, we propose an idea to deal with this problem.

4. The implementation is part of the BaobabLuna package, available online at http://biomserv.univ-lyon1.fr/~marilia/luna.htm.

4.2 Perfect Solutions and Perfect Traces

We now show how the algorithm presented in this paper can be useful to study a variant of sorting by reversals called "perfect" sorting.

A *common interval* of a permutation π is an interval of π that is also an interval of the identity permutation. Common intervals are used to model groups of homologous genes colocalized in the two species that are represented by the permutations. The idea behind common intervals is that, if these genes are together in both species, then probably they were together in the common ancestor of the two species and were not separated by evolution. A sequence of reversals $\rho_1 \dots \rho_k$ that sorts a permutation π is said to be *perfect* if no reversal among $\rho_1, \dots, \rho_k$ overlaps a common interval of π. We are interested in finding optimal scenarios (i.e., that sort a permutation in a minimum number of reversals) that respect the principle of conserved segments. The problem of finding such a scenario is investigated in [4] and [11]. However, the algorithm presented in [4] always gives only one scenario, if this exists.

We analyze the behavior of perfect scenarios in relation to traces. First, we remark that the classification into traces is well adapted to this constraint because of the following property.

Proposition 1. *Every trace of solutions for sorting a signed permutation by reversals contains either only perfect solutions or no perfect solution.*

Proof. If any sequence $s = \rho_1 \dots \rho_k$ for sorting a permutation π is perfect, by definition, none of $\rho_1, \dots, \rho_k$ overlap some common interval of π. So, any sequence with the same reversals in a different order is perfect. This is the case for all the sequences of a trace, so if one sequence of a trace is perfect, they all are. □

We, therefore, call *perfect* a trace that contains only perfect solutions of length $d(\pi)$. Such a trace does not always exist: All parsimonious scenarios may break common intervals (see [11]).

In order to find the perfect traces, we may modify the algorithm by stopping the exploration of an i-trace when a found reversal overlaps a common interval. In this way, we can also reduce the running time and decrease considerably the amount of memory we need. In the case where a perfect trace does not exist, we can easily generalize this approach by fixing a threshold on the number of reversals that may overlap a common interval. A trace that is below this threshold is called a *near-perfect* trace.

We have investigated how the solution space is reduced when only perfect or near-perfect traces are taken into account. In order to do that, we ran a simulation generating 100 random permutations, with size $n = 12$ and reversal distance $2 \leq d \leq 11$. For each permutation, we computed the number of reversals that overlap common intervals for each trace (if this number is equal to zero, we have a perfect trace).

By choosing as near-perfect traces those that minimize this number among all traces (which means only the perfect traces if one exists), we computed the ratio of solutions and traces that are near perfect.

On average, over the 100 permutations performed, the solution space is divided by three when the near-perfect criterion is applied. What is interesting in the structure of traces in this case is that the number of traces is itself divided by 20. This is not surprising since the posets of perfect traces usually have a greater width than nonperfect ones. This may indicate that finding perfect or near-perfect traces could provide a better representation of the solution space of sorting by reversals.

4.3 Analysis of a Scenario for the Evolution of the X and Y Human Chromosomes

We then applied our algorithm to analyze some scenarios concerning the evolution of the mammalian X and Y chromosomes. These chromosomes are very different; in particular, in humans, for instance, while the X chromosome is 155 Mbp long, the Y chromosome is 58 Mbp long. Nevertheless, both are believed to have evolved from an identical autosomal pair[5] [18]. This process is at the origin of sexual differentiation: the female XX and the male XY pairs. Due to the recombination mechanism, female organization favors conservation of the X chromosome. On the other hand, evolution of the male XY pair causes the divergence of the Y chromosome, as it gradually loses the capacity of recombining with its X partner.

The X and Y chromosomes still share a "pseudoautosomal" region at one of their extremity, where recombination occurs as between autosomes. Ninety percent of the Y chromosome is, however, male specific, and shows major differences in sequence as well as in gene order with the X. Current theories suggest that the pseudoautosomal region, which originally covered the whole chromosomes, was successively pruned by a few big reversals on the Y chromosome [16], whose extremities stood on each side of the limit of the pseudoautosomal region. The successive limits of the pseudoautosomal region on the X chromosome, from the origin to where it is located now, represent the limits of what have been called the "evolutionary strata" of the sex chromosomes.

Several arguments seem to indicate the presence of five strata on the X chromosome [19], [21]. The strata are ordered according to their creation time. Thus, the stratum that is the closest to the pseudoautosomal region is numbered 5, while the stratum that is at the other extremity of the X chromosome is numbered 1. A scenario of reversals on a signed permutation representing the relative ordering of the genes common to chromosomes X and Y has been published in [19], and is given as an argument to support the existence and bounds of the most recent strata. The scenario is represented in Fig. 3.

However, for the same permutation, there are many scenarios that are possible, including other scenarios that are in agreement with a model of evolution by strata, which we now describe.

4.3.1 Model of Evolution by Strata

For a signed permutation $\pi = (\pi_1, \dots, \pi_n)$, a k-**strata** is defined as a partition of π into a sorted set $B = (I_k, I_{k-1}, \dots, I_1)$ of k intervals, such that

5. Autosomes are all nonsex chromosomes.

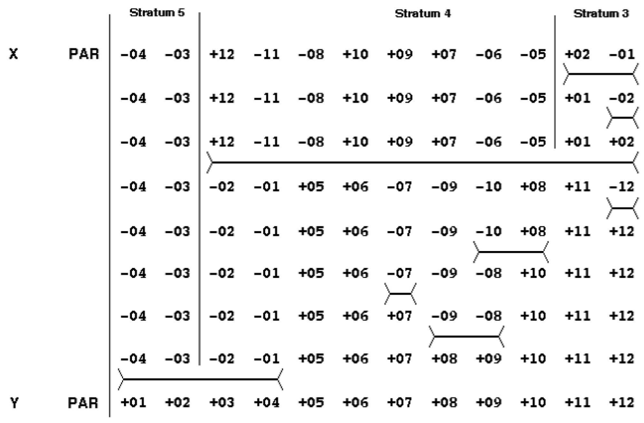

Fig. 3. Scenario of rearrangement of human Y chromosome that shows the formation of the last three strata (numbered 3, 4, and 5) on X chromosome (extracted from [19]). The PAR symbol represents the pseudoautosomal region in each chromosome.

$$I_k = \{|\pi_1|, \ldots, |\pi_{n_k}|\},$$
$$I_{k-1} = \{|\pi_{n_k+1}|, \ldots, |\pi_{n_k+n_{k-1}}|\}, \ldots,$$
$$I_1 = \{|\pi_{n_k+\ldots+n_2+1}|, \ldots, |\pi_{n_k+\ldots+n_1}|\},$$

where n_i is the size of the interval I_i. Observe that the intervals are ordered by their positions, but they are indexed in a decreasing way from the beginning to the end of the permutation.

We say that an optimal sequence of reversals $r = \rho_1\rho_2\ldots\rho_d$ produces a k-strata $B = (I_k, I_{k-1}, \ldots, I_1)$ in π if r has a subsequence[6] $b = \theta_1\theta_2\ldots\theta_k$, such that for $1 \le i \le k$, the reversal θ_i contains the interval I_i and, for any $j > i$, no element of I_j is in θ_i. In addition, for any two consecutive reversals θ_i and θ_{i+1} of b, if ρ is a reversal that occurs between θ_i and θ_{i+1} in r, then ρ is a subset of $I_1 \cup I_2 \ldots \cup I_i$. The reversals in b are said to be **big reversals** (each big reversal creates a new stratum), while the reversals of r that are not in b are said to be **small reversals**. A sequence of reversals that produces a k-strata has k big reversals and $d - k$ small reversals (we recall that d is the reversal distance for the given permutation).

Consider a permutation π and a

$$k\text{-strata } B = (I_k, I_{k-1}, \ldots, I_1)$$

for π. If T is a trace of optimal reversal sequences for π, we call B-**induced subtrace** T_B the subset of T defined as $T_B = \{s | s \in T \text{ and } s \text{ produces the } k\text{-strata } B \text{ in } \pi\}$.

We have developed a version of our exploration algorithm that, given a k-strata B, outputs the set of solutions that produces B. This requires a slight modification of the algorithm described as follows.

4.3.2 Algorithm for Exploring the Solutions that Stratify a Permutation

Besides a signed permutation π, the modified algorithm requires a k-strata $B = (I_k, I_{k-1}, \ldots, I_1)$ for π. The algorithm returns the traces whose B-induced subtraces are not empty, and, for each trace, the size of its B-induced subtrace.

6. A "subsequence" b of a sequence r is obtained by eliminating some of the elements (here reversals) of r while preserving the order of the remaining elements.

It is the first step (Siepel's step) that is mainly modified. After searching all of the next reversals, we must select only those that are compatible with the given k-strata: the first reversal is fixed and corresponds exactly to the first stratum (it is a big reversal); then, at each step, suppose stratum I_k has been moved by a big reversal and not stratum I_{k+1}, we can choose between performing a big reversal including I_{k+1} and no elements from the following ones, or a small reversal, included in $I_1 \cup \ldots \cup I_k$.

The complexity of the algorithm is not changed under this modification: to select a reversal, we only need to compare its boundaries to the strata boundaries. The selection step can be done in time $O(n^2)$ since the number of reversals returned by Siepel's algorithm is $O(n^2)$ (n is the size of the permutation). The number of selected big and small reversals is also bounded by n^2.

The trace construction remains unchanged, but, as a consequence of the selection of the reversals to be performed, we in fact construct the B-induced subtraces that compute only the solutions that produce the given k-strata. At the end of the execution, we have the complete set of nonempty B-induced subtraces (represented by the normal forms of the corresponding traces) and their sizes for a given permutation and a k-strata B.

4.3.3 Analysis of the Results

We applied our modified algorithm to the permutation $\pi = (-4, -3, 12, -11, -8, 10, 9, 7, -6, -5, 2, -1)$, derived from the genes that are shared by the last three strata of the human X and Y chromosomes (Fig. 3). It was used in [19] to account for the positions of the strata 3, 4, and 5 in the human sex chromosomes, by giving one solution to sorting by reversals.

We are now able to handle the whole set of solutions: the solution space of sorting this permutation by reversals contains 31,752 solutions, distributed among the following six traces:

$$\{1,2\}\{1,2,5,...,12\}\{2\}\{7\}\{8,10\}\{12\}|\{1,...,4\}\{8,9\}$$
$$\{1,...,12\}\{2\}\{3,4,12\}\{5,...,11\}\{7\}\{8,10\}|\{3,...,11\}\{8,9\}$$
$$\{1,...,12\}\{2,...,12\}\{2,5,...,12\}\{7\}\{8,10\}\{12\}|\{2,3,4\}\{8,9\}$$
$$\{2,5,...,12\}\{5,...,11\}\{7\}\{8,10\}|\{1,12\}\{8,9\}|\{1,5,...,11\}|\{1,2,3,4\}$$
$$\{1,2\}\{7\}\{8,10\}|\{1,5,...,11\}\{8,9\}|\{1,3,4,12\}|\{2,...,12\}\{3,...,11\}$$
$$\{2,...,12\}\{7\}\{8,10\}|\{1,3,4,12\}\{8,9\}|\{1,5,...,11\}|\{1,2\}\{3,...,11\}$$

(each trace contains, respectively, 10080, 10080, 10080, 840, 336, and 336 solutions)

When we search only for the scenarios of sorting by reversals that respect the formation of the last three strata as they are defined in [19], that is,

$$B = (\{3,4\}, \{5,6,7,8,9,10,11,12\}, \{1,2\}),$$

we get 420 solutions, corresponding to a unique B-induced subtrace, represented by the normal form of its corresponding trace:

$$\{1,2\}\{1,2,5,\ldots,12\}\{2\}\{7\}\{8,10\}\{12\}|\{1,\ldots,4\}\{8,9\},$$
$$(420 \text{ sol.}).$$

There are, in consequence, 420 solutions out of 31,752 that support the bounds given in [19], and they are all in the

$\pi = (-4, -3, 12, -11, -8, 10, 9, 7, -6, -5, 2, -1)$
$B = (I_3 = \{3, 4\}, I_2 = \{5, 6, 7, 8, 9, 10, 11, 12\}, I_1 = \{1, 2\})$
(human X and Y chromosomes)

Algorithm	Number of solutions	Number of traces	Execution time
all traces	31752	6	$\simeq$ 1.3 seconds
B-induced subtraces	420	1	$\simeq$ 0.5 seconds

$\pi = (-4, 12, -11, -9, 10, 8, -7, 3, 5, -2, 6, 14, -15, 1, 13)$
$B = (I_3 = \{4, 12\}, I_2 = \{7, 8, 9, 10, 11\}, I_1 = \{1, 2, 3, 5, 6, 13, 14, 15\})$
(extended human X and Y chromosomes, from an ongoing study)

Algorithm	Number of solutions	Number of traces	Execution time
all traces	316793943648	87983	$\simeq$ 5 hours
B-induced subtraces	512340774	1878	$\simeq$ 5 min

Fig. 4. Computation results (2). Columns from left to right contain: 1) the algorithm, 2) the number of solutions according to the algorithm, 3) the number of traces according to the algorithm, and 4) the execution time of the algorithm. Both algorithms are part of the BaobabLuna Package. Experiments were made on a personal computer with a 1.8-GHz CPU and 1-Gbyte RAM.

same trace. Here, more relevant than the number of solutions is the fact that they are all part of a unique subtrace, which means that the reversals have been identified correctly. So, if we suppose the bounds are known, the scenario given in [19] is accurate.

Nevertheless, the limits between strata may be not so clear, and one could be interested in testing other hypotheses. For instance, we could extend stratum 4 incorporating to it the markers 1 and 2 which were part of stratum 3 in the previous scenario. According to this hypothesis, which could be biologically meaningful as well, we have $B' = (\{3, 4\}, \{1, 2, 5, 6, 7, 8, 9, 10, 11, 12\})$, and there are 2,520 solutions in the B'-induced subtrace:

$$\{1, 2\}\{1, 2, 5, \ldots, 12\}\{2\}\{7\}\{8, 10\}\{12\}|\{1, \ldots, 4\}\{8, 9\}, \\ (2520 \text{ sol.}).$$

Thus, using our algorithm, we are able to evaluate the different hypotheses of stratification and find all the subtraces (that is, a representation of all solutions) that produce each stratification.

4.3.4 On the Execution Time of the Strata Variant

In order to evaluate the execution time of this modified algorithm, we ran both the original algorithm (that searches for all the traces) and the modified one (that searches for the strata-induced subtraces) over the previous permutation and over an extended permutation, also extracted from human X and Y chromosomes' evolution.[7] The results are presented in Fig. 4 and shows that searching for strata-subtraces runs much faster than searching for all traces. This is an advantage of modifying the implementation of the algorithm instead of making all treatments a posteriori, like in the previous section, for searching perfect traces.

5 Conclusions, Limitations and Perspectives

We have devised an algorithm that gives a representation of all the solutions of sorting signed permutations by reversals, without enumerating each solution. It is the first algorithm that achieves this, to our knowledge, and it performs better than the complete enumeration on all the data on which we tested it.

The implementation of the algorithm is online, integrated to a package for the manipulation of signed permutations.

Although this program is faster than the previously published methods, it is still limited (mainly because of memory) to small permutations, with $d(\pi)$ of at most 20, on a personal computer that has 1-Gbyte RAM.[8] It is sufficient for some biological applications, as we show on the comparison of the human X and Y chromosomes (see Fig. 4). For many data sets however, it is still insufficient because of the size of the output.

Indeed, if the solution space is dramatically reduced when dealing with traces of solutions, it is often still too big to be handled by biologists on large permutations. The algorithmic limit coincides, therefore, with the limit of the utility of the solution. An idea to try to solve this problem is to add further biological criteria (besides the principle of conserved elements we have shown in Section 4.2) about perfect traces to reduce the number of traces. If the criteria can be checked during the computation of traces, and not only a posteriori, then we can also reduce the amount of memory and push further the limits of our algorithm. It is also the case for the strata-induced subtraces, where the property can be checked during the construction of the traces, speeding up the execution of the constrained version related to the evolution of the mammalian sex chromosomes.

The analysis of the traces of solutions related to the gene positions on the X and Y chromosomes shows the utility of such an algorithm to explore the solution space of sorting by reversals. Indeed, while giving one scenario as in [19] may support some strata boundaries, the examination of the whole set of traces and strata-induced subtraces gives different insights on the value of this support.

Of course, one may also consider that the scenario of evolution that separated the X and Y chromosomes (and more generally, any instance of the problem) was not an optimal scenario, or may contain other events than reversals. For the present methods, if we consider near-optimal solutions, the space of solutions is bigger and harder to handle. It is, however, important to go toward this direction in the future.

Acknowledgments

This work is funded by the Agence Nationale pour la Recherche (ANR), project REGLIS NT05-3_45205, and by the Action de Recherche Coopérative (ARC), project IBN, from the INRIA Institute. Marilia D.V. Braga is supported by the Programme Alβan, the European Union Programme of High Level Scholarships for Latin America (scholarship no. E05D053131BR).

References

[1] Y. Ajana, J.F. Lefebvre, E. Tillier, and N. El-Mabrouk, "Exploring the Set of All Minimal Sequences of Reversals—An Application to Test the Replication-Directed Reversal Hypothesis," *Proc. Second Int'l Workshop Algorithms in Bioinformatics (WABI '02),* vol. 2452, pp. 300-315, 2002.

7. This extended permutation is the object of an ongoing study on the analysis of the human X and Y chromosomes' evolution.

8. This extensive use of memory is due to the fact that, in order to create the i-traces, we have to store all the $(i-1)$-traces.

[2] D.A. Bader, B.M.E. Moret, and M. Yan, "A Linear-Time Algorithm for Computing Inversion Distances Between Signed Permutations with an Experimental Study," *J. Computational Biology,* vol. 8, no. 5, pp. 483-491, 2001.

[3] S. Berard, A. Bergeron, and C. Chauve, "Conserved Structures in Evolution Scenarios," RCG 2004, *Lecture Notes in Bioinformatics,* vol. 3388, pp. 1-15, 2005.

[4] S. Berard, A. Bergeron, C. Chauve, and C. Paul, "Perfect Sorting by Reversals Is Not Always Difficult," *IEEE/ACM Trans. Computational Biology and Bioinformatics,* vol. 4, no. 1, pp. 4-16, Jan.-Mar. 2007.

[5] A. Bergeron, C. Chauve, T. Hartmann, and K. St-Onge, "On the Properties of Sequences of Reversals That Sort a Signed Permutation," *Proc. Journées Ouvertes Biologie, Informatique et Mathématiques (JOBIM '02),* pp. 99-108, 2002.

[6] A. Bergeron, S. Heber, and J. Stoye, "Common Intervals and Sorting by Reversals: A Marriage of Necessity," *Bioinformatics,* vol. 18 (Suppl. 2), pp. S54-63, 2002.

[7] A. Bergeron, J. Mixtacki, and J. Stoye, "The Inversion Distance Problem," *Math. of Evolution and Phylogeny,* O. Gascuel, ed., pp. 262-290, Oxford Univ. Press, 2005.

[8] M.D.V Braga, M.-F. Sagot, C. Scornavacca, and E. Tannier, "The Solution Space of Sorting by Reversals," *Proc. Int'l Symp. Bioinformatics Research and Applications (ISBRA '07),* vol. 4463, pp. 293-304, 2007.

[9] G. Brightwell and P. Winkler, "Counting Linear Extensions is #P-Complete," *Proc. 23rd Ann. ACM Symp. Theory of Computing (STOC),* 1991.

[10] *The Book of Traces,* V. Diekert, G. Rozenberg, eds. World Scientific, 1995.

[11] Y. Diekmann, M.F. Sagot, and E. Tannier, "Evolution under Reversals: Parsimony and Conservation of Common Intervals," *IEEE/ACM Trans. Computational Biology and Bioinformatics,* vol. 4, no. 2, pp. 301-309, Apr.-June 2007. (A preliminary version appeared in COCOON 2005, LNCS 3595, 42-51, 2005.)

[12] R.P. Dilworth, "A Decomposition Theorem for Partially Ordered Sets," *Annals of Math.,* vol. 51, pp. 161-166, 1950.

[13] D.R. Fulkerson, "Note on Dilworth's Decomposition Theorem for Partially Ordered Sets," *Proc. Am. Math. Soc.,* vol. 7, pp. 701-702, 1956.

[14] Y. Han, "Improving the Efficiency of Sorting by Reversals," *Proc. Int'l Conf. Bioinformatics and Computational Biology,* 2006.

[15] S. Hannenhalli and P. Pevzner, "Transforming Cabbage into Turnip (Polynomial Algorithm for Sorting Signed Permutations by Reversals)," *J. ACM,* vol. 46, pp. 1-27, 1999.

[16] B.T. Lahn and D.C. Page, "Four Evolutionary Strata on the Human X Chromosome," *Science,* vol. 286, pp. 964-967, 1999.

[17] Z. Li, L. Wang, and K. Zhang, "Algorithmic Approaches for Genome Rearrangement: A Review," *IEEE Trans. Systems, Man, and Cybernetics,* vol. 36, pp. 636-648, 2006.

[18] S. Ohno, *Sex Chromosomes and Sex-Linked Genes.* Springer, 1967.

[19] M.T. Ross et al., "The DNA Sequence of the Human X Chromosome," *Nature,* vol. 434, pp. 325-337, 2005.

[20] A. Siepel, "An Algorithm to Enumerate Sorting Reversals for Signed Permutations," *J. Computational Biology,* vol. 10, pp. 575-597, 2003.

[21] H. Skaletsky et al., "The Male-Specific Region of the Human Y Chromosome is a Mosaic of Discrete Sequence Classes," *Nature,* vol. 423, pp. 825-837, 2003.

[22] G. Steiner, "An Algorithm to Generate the Ideals of a Partial Order," *Operations Research Letters,* vol. 5, no. 6, pp. 317-320, 1986.

[23] G. Steiner, "Polynomial Algorithms to Count Linear Extensions in Certain Posets," *Congressus Numerantium,* vol. 75, pp. 71-90, 1990.

[24] E. Tannier, A. Bergeron, and M.-F. Sagot, "Advances on Sorting by Reversals," *Discrete Applied Math.,* vol. 155, nos. 6-7, pp. 881-888, 2007.

Marília D.V. Braga received the master's degree in computational biology from the University of Campinas, Campinas, Brazil, in 2000. Since 2005, she has been working toward the PhD degree in the Laboratoire de Biométrie et Biologie Evolutive of the CNRS, University of Lyon 1, France. From 2000 to 2005, she was a bioinformatics analyst on three Brazilian genome projects and two Brazilian companies. Her research interests include genome rearrangements and algorithmics.

Marie-France Sagot received the BSc degree in computer science from the University of São Paulo, São Paulo, Brazil, in 1991 and the PhD degree in theoretical computer science and applications and the Habilitation degree from the University of Marne-la-Vallée, Paris, France, in 1996 and 2000, respectively. From 1997 to 2001, she was a research associate at the Pasteur Institute, Paris, France. In 2001, she moved to Lyon, France, as a research associate at the INRIA. She is currently a research director at the INRIA Rhône-Alpes, in the Helix project. She is the head of the Baobab team at the Laboratoire de Biométrie et Biologie Evolutive of the CNRS, University of Lyon 1, France. Her research interests include computational biology, algorithmics, and combinatorics.

Celine Scornavacca received the master's degree in bioinformatics from the Laboratoire de Biométrie et Biologie Evolutive of the CNRS, University of Lyon 1, in 2006. She is currently working toward the PhD degree in the Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier, France.

Eric Tannier received the PhD degree in discrete mathematics from the University of Grenoble 1. He is currently a research associate at the INRIA Rhône-Alpes, in the Helix project. He works at the Laboratoire de Biométrie et Biologie Evolutive of the CNRS, University of Lyon 1, France. His research interests include comparative genomics and combinatorics.