Evolution under reversals: parsimony and conservation of common intervals

Yoan Diekmann, Marie-France Sagot, and Eric Tannier

Abstract— **In comparative genomics, gene order data is often modelled as signed permutations. A classical problem for genome comparison is to detect common intervals in permutations, that is, genes that are co-localised in several species, indicating that they remained grouped during evolution. A second largely studied problem related to gene order is to compute a minimum scenario of reversals that transforms a signed permutation into another. Several studies began to mix the two problems, and it was observed that their results are not always compatible: often parsimonious scenarios of reversals break common intervals. If a scenario does not break any common interval, it is called perfect. In two recent studies, Bérard *et al.* defined a class of permutations for which building a perfect scenario of reversals sorting a permutation was achieved in polynomial time, and stated as an open question whether it is possible to decide, given a permutation, if there exists a minimum scenario of reversals that is perfect. In this paper, we give a solution to this problem, and prove that this widens the class of permutations addressed by the aforementioned studies. We implemented and tested this algorithm on gene order data of chromosomes from several mammal species, and we compared it to other methods. The algorithm helps to choose among several possible scenarios of reversals, and indicates that the minimum scenario of reversals is not always the most plausible.**

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

I. Introduction

In computational biology, it is commonly accepted, using a parsimony argument, that if a group of homologous genes (that is genes having a common ancestry) is co-localised in two different species, then these genes were probably together in the common ancestor and were not later separated during evolution. The detection of such common clusters of homologous genes, that is of what is called *common intervals* of permutations, has already been the subject of several algorithmic studies (see for instance [2], [10]).

In the theory of rearrangements, applying the parsimony principle means minimising the number of events in a reconstruction of possible evolutionary mutations between species. Rearrangement algorithms are used in biology to estimate an evolutionary distance between species, to infer the localisation of the evolutionary breakpoints along a DNA molecule, or to reconstruct the genomes of ancestors of actual species. The algorithmics related to the rearrangements theory has also been intensively studied. The main results have been obtained on the problem of sorting by reversals [9], [5], which is a common event in evolution. The problem in this case concerns finding a shortest sequence of reversals that transforms one genome into the other.

A drawback of the methods developed so far to find such parsimonious evolutionary scenarios is that they do not respect the principle of common intervals: it has indeed been noticed several times that in the case of reversals, the two criteria are not always compatible. A minimum rearrangement scenario may break common intervals and then put them back together later again.

A scenario that does not break any common interval is called *perfect*. A few studies [3], [4], [8] began to mix the two principles by designing perfect scenarios, with an NP-completeness result, and an algorithm which is polynomial for an identified class of permutations. We go further in this direction, answering an open question mentioned in [3] and showing through an example that the class defined

Yoan Diekmann is at the Technische Fakultät of the University of Bielefeld, Germany

Marie-France Sagot and Eric Tannier are at the Laboratoire de Biométrie et Biologie Evolutive (UMR 5558) of the CNRS and University of Lyon 1, Villeubanne, and from the HELIX project of the INRIA Rhône-Alpes, France

Marie-France Sagot is a visiting professor of the King's College, London, UK

This work is funded by the ACI "New interfaces of Mathematics" (project "Mathematical and algorithmical aspects of biochemical and evolutionary networks"), by the INRIA coordinated action ARC (project "Integrated Biological Networks"), and by the Agence Nationale de la Recherche (project REGLIS, "From the molecule to the cell").

in [4] is widened by our study. The open question concerned the possibility of designing a polynomial time algorithm to decide whether there exists a minimum scenario of reversals that is perfect. The principles of our solution were first presented in [11].

This study has also several other motivations. Indeed, a second drawback of the algorithms for obtaining optimal scenarios of rearrangements is the huge number of solutions they provide. Trying to add criteria coming from constraints on common intervals could be a solution to discriminate more plausible solutions.

Moreover, in another study on the global alignment of mammalian genomes [6], the authors use common intervals to cluster markers along a genome, and then apply rearrangement algorithms on the clustered data. To cope with the possible incompatibility between the two principles (of common intervals and of a most parsimonious scenario), they have to use an arbitrary threshold for deciding whether a rearrangement is inside or outside a common interval: below this threshold, a rearrangement is ignored and data are clustered, and above the threshold, it is considered as a *bona-fide* rearrangement. The authors of [6] put as an open problem how to get rid of this artificial parameter. Knowing how to cluster genes during the execution of a rearrangement algorithm would be a more natural way to answer this question. We tested our method on some chromosomes of several mammal species, and we report the cases when the optimal scenario of reversals is not compatible with common intervals, indicating that one should choose between the two principles and not use both.

This paper is organised as follows: we describe the usual model for dealing with gene order and orientation in the next section. In Section III, we recall some basic facts about the structure of common intervals of a permutation, and introduce a property that characterises some common intervals in terms of rearrangements. This will lead in Section IV to the algorithm that is the keystone of the paper. Section V relates this study to the problem of perfect sorting by reversals, generalising the results of [4] by handling a class of permutations not included in the latter. Last, in Section VI, we provide some insights on the theoretical and practical complexity of the algorithm, and summarise how it behaves on data from some mammalian chromosomes.

II. Chromosomes as Signed Permutations

A. Generalities

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 $1, \ldots, n$, with a plus or minus sign to indicate their orientation. The order and orientation of genomic markers of one species related to another will be represented by a *signed permutation* of $\{1, \ldots, n\}$, that is, by a bijection π over $[-n, n] \setminus \{0\}$, with the constraint that $\pi_{-i} = -\pi_i$, where $\pi_i = \pi(i)$. The permutation may thus be given by the image of $1 \ldots n$.

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 therefore often define a signed permutation by writing $(0\ \pi_1\ \ldots\ \pi_n\ n+1)$. The *identity permutation* $(0\ 1\ \ldots\ n\ n{+}1)$ is denoted by Id.

For all $i \in \{0, \ldots, n\}$, the pair $\pi_i\pi_{i+1}$ is called a *point* of π, and more precisely an *adjacency* if $\pi_i + 1 = \pi_{i+1}$ and a *breakpoint* otherwise. The number of points of a permutation π is denoted by $p(\pi)$, and the number of its breakpoints by $b(\pi)$.

The *reversal* of the interval $[i, j] \subseteq [1, n]$ $(i \leq j)$ is the signed permutation $\rho_{i,j} = (0\ \ldots\ i-1\ -j\ \ldots\ -i\ j+1\ \ldots\ n+1)$. Note that $\pi \cdot \rho_{i,j}$ is the permutation obtained from π by reversing the order and flipping the signs of the 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 $\pi \cdot \rho_1 \cdots \rho_k = Id$. In this case, the sequence is called a *scenario* of reversals for π. The length of a shortest sequence of reversals that sorts π is called the *reversal distance* of π, and is denoted by $d(\pi)$. A shortest sequence of reversals sorting π is called a *parsimonious* scenario.

An *interval* of a permutation π is a set $\{|\pi_a|, \ldots, |\pi_b|\}$, with $1 \leq a < b \leq n$. The numbers π_a and π_b are the *extremities* of the interval. Two intervals are said to *overlap* if they intersect but one is not contained in the other. A reversal $\rho_{i,j}$ *breaks* $\{|\pi_a|, \ldots, |\pi_b|\}$ if $[i, j]$ and $[a, b]$ overlap. A sequence of reversals *breaks* an interval I if at least one reversal of the sequence breaks I.

B. Common Intervals

Let π be a signed permutation of $\{1, \ldots, n\}$, and $I = \{|\pi_a|, \ldots, |\pi_b|\}$ an interval of π, for $[a, b] \subseteq [1, n]$. Let $m = \min_{i \in [a,b]} |\pi_i|$ and $M = \max_{i \in [a,b]} |\pi_i|$.

The interval I is said to be *oriented* if there exist $i, j \in [a, b]$, such that π_i and π_j have different signs, and *unoriented* otherwise.

The interval I is said to be *sorted* if for all $i \in [a, b-1]$, the point $\pi_i \pi_{i+1}$ is an adjacency. A sorted interval is always unoriented. It is sorted *positively* if $\pi_a > 0$ and *negatively* if $\pi_a < 0$. In $\pi = (0\ -7\ 3\ -1\ 4\ 2\ 8\ -6\ -5\ 9)$, $\{6, 5\}$ is sorted negatively.

The interval I is said to be a *common interval* if $M - m = b - a$ (it is common to π and Id). In $\pi = (0\ -7\ 3\ -1\ 4\ 2\ 8\ -6\ -5\ 9)$, $\{3, 1, 4, 2\}$ is a common interval.

The common interval I is said to be *framed* if either $\pi_a = m$ and $\pi_b = M$, or $\pi_a = -M$ and $\pi_b = -m$. A *component* of π is a framed interval that is not the union of framed intervals properly contained in it. In $\pi = (0\ -7\ -4\ 2\ -3\ -1\ 8\ -6\ -5\ 9)$, the interval $\{4, 2, 3, 1\}$ is framed, and it is a component.

The interval I is said to be *perfect* if there exists a parsimonious sequence of reversals which does not break I.

A perfect interval is a common interval, but the converse is not true. For example, in $(0\ -2\ -3\ 1\ 4)$, $\{2, 3\}$ is a common interval but any parsimonious scenario breaks it.

According to [3], we say that a sorting sequence of reversals is *perfect* if it breaks no common interval. If a permutation has a perfect parsimonious scenario, then all its common intervals are perfect. The converse is however not true: for example, in $(0\ -3\ 4\ -1\ 2\ 5)$, both $\{1, 2\}$ and $\{3, 4\}$ are perfect, but any parsimonious sequence of reversals breaks one of them.

Perfect sorting sequences of minimum size have been studied in [3], [4], [8]. In [8], it is proved that given a permutation and a subset $\mathcal{I}$ of its common intervals, it is NP-hard to compute the minimum scenario that does not break the intervals of $\mathcal{I}$. The problem of finding a perfect sequence of reversals of minimum length is still open, to our knowledge. In [4], a class of permutations is presented, for which this problem is polynomial. We show in Section V that we can enlarge this class with infinite families of permutations.

The parsimonious scenario such that the smallest possible number of reversals break some common intervals is evoked in [3], but not solved. The authors study a special class of permutations for which there exists a perfect parsimonious scenario, and the question is asked whether it is possible to decide in polynomial time, given a permutation, if there is a perfect parsimonious scenario that sorts it. In Section IV, we give this algorithm. Before that, we still need some preliminaries.

C. Sorting by reversals

The main result about sorting by reversals is a theorem of Hannenhalli and Pevzner [9], which yielded the first polynomial algorithm to find a parsimonious sequence of reversals sorting any signed permutation.

We mention here a weaker version of this theorem, to avoid introducing notions which are useless for our purpose. One of the consequences of the general version of Hannenhalli and Pevzner's theorem is that it is possible to characterise the permutations for which all parsimonious sequences of reversals have to break some framed interval. According to the standard vocabulary, they are the permutations that need a "hurdle merging". They can be characterised in this way. In [5], it is noted that two components are disjoint, nested with different extremities, or overlapping on one element. The following lemma is extrapolated from the structure of components studied in [5].

Lemma 1: [5] For a permutation π, any scenario of reversals has to break some component if and only if π has at least three unoriented components A, B, C, such that one of the following holds:

- $A \subset B \subset C$;
- $C \cap B = \emptyset$ and A contains B and C;
- $|C \cap B| \leq 1$ and A is contained in B or C.

We call such permutations *fools*. They obviously never have a perfect parsimonious scenario. We therefore start by assuming that the permutations we treat are not fools. It is easy to decide in linear time if a permutation is a fool or not (see for example [5]). We denote by $u(\pi)$ the number of unoriented components in a permutation π.

Another parameter is useful for stating the theorem, and is related to the so-called breakpoint graph. This is a usual tool for dealing with signed permutations, and it is present in almost every study

on sorting by reversals. We use it intensively in the proofs of correctness of our method.

The *breakpoint graph* $BG(\pi)$ of a permutation π is a graph with vertex set V defined as follows: for each integer i in $\{1, \ldots, n\}$, let i^- and i^+ be two vertices in V; add to V the two vertices 0^+ and $(n+1)^-$. Observe that all vertex labels are non negative numbers, but for simplicity and to avoid having to use absolute values, we may later refer to vertex $(-i)^+$ (or $(-i)^-$): this is the same as vertex i^+ (or i^-).

The breakpoint graph of a signed permutation has sometimes been called the *diagram of desire and reality* due to the edge set E of $BG(\pi)$, which is the union of two perfect matchings of V, denoted by R, the *reality edges* and D, the *desire edges*:

- D contains the edges $i^+(i+1)^-$ for all $i \in \{0, \ldots, n\}$;
- R contains an edge for all $i \in \{0, \ldots, n\}$, from π_i^+ if π_i is non negative, and from π_i^- otherwise, to π_{i+1}^- if π_{i+1} is non negative, and to π_{i+1}^+ otherwise.

Reality edges define the permutation π (what you have), and desire edges define Id (what you want to have). An example of breakpoint graph is drawn in Figure 1.

To avoid case checking, in the notation of an edge, the mention of the exponent $+$ or $-$ may be omitted. For instance, $\pi_i\pi_{i+1}$ is a reality edge, indicating nothing as concerns the signs of π_i and π_{i+1}.

It is easy to check that every vertex of $BG(\pi)$ has degree two (it has one incident edge in R and one in D), so the breakpoint graph is a set of disjoint cycles. By the cycles of a permutation π, we mean the cycles of $BG(\pi)$. The number of cycles of π is denoted by $c(\pi)$.

We now state the theorem of Hannenhalli and Pevzner. Recall that $d(\pi)$ is the minimum number of reversals needed to sort a permutation, $p(\pi)$ is the number of points of the permutation (adjacencies and breakpoints), $c(\pi)$ is the number of cycles of the breakpoint graph, and $u(\pi)$ is the number of unoriented components.

Theorem 1: [9] Let π be a permutation but not a fool. Then $d(\pi) = p(\pi) - c(\pi) + u(\pi)$.

This means that any reversal in a perfect parsimonious scenario increases by one the number of cycles of the permutation ($p(\pi) = n + 1$ does not change after a reversal), except one (the first one) for each unoriented component. Each component is sorted separately, and independently from the rest of the permutation. A detailed description of this is given in Section IV.

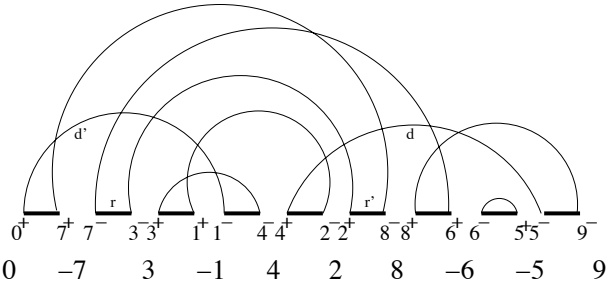

Fig. 1. The breakpoint graph of the permutation $(0\ -7\ 3\ -1\ 4\ 2\ 8\ -6\ -5\ 9)$. Reality edges are represented in bold, and desire edges are represented by thin lines.

III. The structure of common intervals

As mentioned in the previous section, two framed intervals may only overlap on one of their extremities, so it is easy to identify an inclusion-wise minimal one and treat it separately. This is not immediately the case in general for common intervals. However, common intervals of a permutation have a nice structure, which will allow us to consider intervals one by one, starting from inclusion-wise minimal ones. We recall basic facts about the structure of common intervals that are useful for our purpose. The reader may refer to [7] for a general presentation on modular structures.

A. Strong common intervals

A common interval is called *strong* if it does not overlap any other common interval. By definition, the family of strong common intervals is nested. We then define the *tree of strong common intervals* of a permutation, in which each node is a strong common interval, and its children are the strong common intervals that it contains. Note that in this tree, the leaves are the numbers in the permutation, and the root is the whole permutation. An example of such a tree for the permutation $(0\ -7\ 3\ -1\ 4\ 2\ 8\ -6\ -5\ 9)$ is given in Figure 2.

Given any strong interval, it is possible to assign an order to its children. Indeed, given two of its children A and B, since they are common intervals themselves, either all the numbers of A are bigger

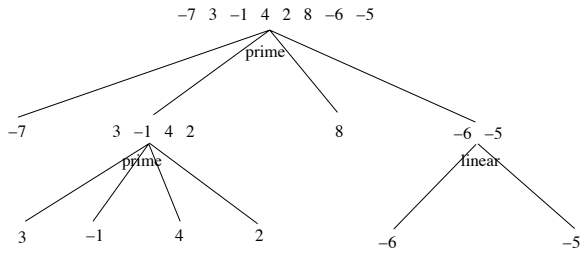

Fig. 2. The tree of strong common intervals of the permutation $(0\ -7\ 3\ -1\ 4\ 2\ 8\ -6\ -5\ 9)$.

(in absolute value) than all the numbers of B, either all the numbers of A are smaller (in absolute value) than all the numbers of B. So it is possible to give an order to the set of children of a strong interval, and to say $A < B$, or $B < A$. For a strong interval I, with children $C_1, \ldots, C_k$, a *quotient* of I is a permutation of k positive numbers $(c_1\ \ldots\ c_k)$, such that $c_i < c_j$ if and only if $C_i < C_j$, and c_i is placed before c_j if and only if the elements of C_i are placed before the elements of C_j.

For example, if the permutation is $(-7\ \ 3\ \ -1\ \ 4\ \ 2\ \ 8\ \ -6\ \ -5)$, suppose I is the whole permutation, the children nodes of I are $\{7\}$, $\{3,1,4,2\}$, $\{8\}$, $\{6,5\}$. A quotient permutation of I is therefore $(3\ \ 1\ \ 4\ \ 2)$.

Strong intervals can be of two types. If all the intervals of a quotient permutation are common intervals (a quotient is Id or $Id \cdot \rho_{1,n}$), then I is called *linear*. If the only common intervals are the singletons, then I is called *prime*. A singleton is both linear and prime. There are examples of linear and prime nodes on the permutation of Figure 2. The following lemma is where lies the great interest of this structure, and what makes dealing with strong intervals sufficient for obtaining properties on all intervals.

Lemma 2: [7] If I is a strong interval, then it is either linear, or prime.

We shall sort each strong interval independently, assuming that all the strong intervals strictly included in it are already sorted (we start by the inclusion-wise minimal ones), and end with the last strong common interval, which is the permutation itself, when all the strong sub-intervals have been sorted. We therefore always deal with quotient permutations. The problem will be to know in which direction to sort an interval, when all sub-intervals have been sorted. This is what we study now.

B. Perfect intervals

A trivial necessary condition for the existence of a perfect parsimonious scenario is that every common interval is perfect. It is easy to determine if one strong interval is perfect or not, thanks to the following fundamental lemma presented in [8].

Lemma 3: [8] If a sequence of reversals sorts a permutation and does not break an interval I, then there exists a sorting sequence of same size (with the same reversals), in which all the reversals contained in I (they sort I) are before all other reversals (they sort outside I).

If we want to know if an interval is perfect or not, it is sufficient to know if there is a sequence of reversals that sorts the interval, and preserves optimality of the remaining permutation. The difficulty remains to choose whether to sort the interval positively or negatively. For this purpose, we introduce two new notations. Let $[a,b] \subseteq [1,n]$ be such that $I = \{|\pi_a|, \ldots, |\pi_b|\}$ is a strong common interval. Let $M = \max_{i\in[a,b]} |\pi_i|$, and $m = \min_{i\in[a,b]} |\pi_i|$. Denote by ι^+ the permutation of the numbers $[m-1, M+1]$, which consists in the numbers of I framed by $m-1$ and $M+1$. Let ι^- be the permutation which consists in the numbers of I framed by $-(M+1)$ and ends with $-(m-1)$.

We have that $d(\iota^+)$ is the minimum number of reversals needed to sort a common interval I positively, and $d(\iota^-)$ the minimum number of reversals needed to sort it negatively. Of course, $|d(\iota^+) - d(\iota^-)| \leq 1$, because if it is sorted in one direction, then one reversal is sufficient to have it sorted in the other. If $d(\iota^+) = d(\iota^-)$, the interval I is called *neutral*.

Call $\pi \setminus I^+$ the obtained permutation when I has been sorted positively, and $\pi \setminus I^-$ the obtained permutation when I has been sorted negatively. An example of permutations ι^+, ι^-, $\pi \setminus I^+$ and $\pi \setminus I^-$ is given in Figure 3.

By Lemma 3, we have this characterisation of perfect intervals.

Lemma 4: An interval I of π is perfect if and only if $d(\pi) = d(\iota^+) + d(\pi \setminus I^+)$ or $d(\pi) = d(\iota^-) + d(\pi \setminus I^-)$.

As a consequence, if ι^+ or $\pi \setminus I^+$ contain more unoriented components than π, then because of

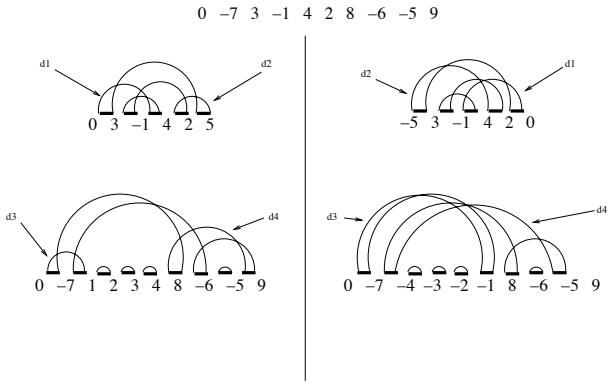

Fig. 3. Permutations ι^+, ι^-, $\pi \setminus I^+$ and $\pi \setminus I^-$ for the interval $I = \{3, 1, 4, 2\}$ in the permutation $(0\ -7\ 3\ -1\ 4\ 2\ 8\ -6\ -5\ 9)$, and their breakpoint graphs. Here, $c(\iota^+) = c(\iota^-) = 1$, $c(\pi) = 1$, $c(\pi \setminus I^+) = 5$ and $c(\pi \setminus I^-) = 6$, so the interval I is neutral and $d(\pi \setminus I^+) \neq d(\pi \setminus I^-)$.

Theorem 1, the interval I cannot be perfect, and it is the same for the negative case.

The following key result helps to choose between the positive and negative direction. It is the result that makes the polynomial algorithm possible. We treat the case of unoriented components separately in the algorithm, so this lemma will be useful only in the case of oriented components.

Lemma 5: If I is a perfect neutral interval included in an oriented component of a permutation π, then $d(\pi \setminus I^+) \neq d(\pi \setminus I^-)$.

Proof: Let $[a, b] \subseteq [1, n]$ be such that $I = \{|\pi_a|, \ldots, |\pi_b|\}$ is the perfect neutral interval of π. Then $d(\iota^+) = d(\iota^-)$, and suppose that $d(\pi) = d(\iota^+) + d(\pi \setminus I^+)$ (the other case is symmetric). Let $M = \max_{i\in[a,b]} |\pi_i|$, and $m = \min_{i\in[a,b]} |\pi_i|$. As mentioned before, from Theorem 1 and Lemma 4, since I is perfect, there cannot be unoriented components in $\pi \setminus I^+$ that are not in π, so saying that $d(\pi \setminus I^+) \neq d(\pi \setminus I^-)$ is equivalent to saying that $c(\pi \setminus I^+) \neq c(\pi \setminus I^-)$ (again by Theorem 1). We therefore prove the latter.

In the breakpoint graph of π, given an interval I, call a *cut-edge* an edge that has one extremity in I and one outside I. If I is a common interval, there are four cut-edges (two desire and two reality edges), with extremities π_a, π_b, m and M. In the permutation of Figure 1 with $I = \{1, 2, 3, 4\}$, they are the edges denoted by d, d', r and r'. Let $\mathcal{C}$ be the set of cycles of π containing at least one cut-edge. Since cut-edges are exactly the ones separating the inside from the outside of the interval, if a cycle contains one of those edges, it has to contain another one (if it goes outside, it has to come back). Then either $\mathcal{C}$ contains only one cycle, or there are two cycles, each containing two cut-edges. In Figure 1, $\mathcal{C}$ contains only one cycle.

All the cycles outside $\mathcal{C}$ have an exact correspondent in $\pi \setminus I^+$ or ι^+. The cycles of $\mathcal{C}$ in π are replaced by disjoint cycles in $\pi \setminus I^+$ and ι^+. Let $\mathcal{C}^+$ be the set of those cycles. Note that the cycles of $\mathcal{C}^+$ are obtained from the cycles of $\mathcal{C}$ by removing the four cut-edges: this gives a set of paths, with extremities that are then joined by new edges. This gives in particular $|\mathcal{C}^+| \leq |\mathcal{C}| + 2$.

Let us prove that in any case, $|\mathcal{C}^+| = |\mathcal{C}| + 2$. Suppose $|\mathcal{C}^+| < |\mathcal{C}| + 2$. Note that in $\pi \setminus I^+$, there are at least $p(\iota^+) - 2$ adjacencies due to the sorted interval I. So adding to $|\mathcal{C}^+| < |\mathcal{C}| + 2$ the cycles which are common to π and ι^+ or $\pi \setminus I^+$, we get $c(\iota^+) + c(\pi \setminus I^+) - (p(\iota^+) - 2) < c(\pi) + 2$. With $p(\pi) = p(\pi \setminus I^+)$, we get $p(\pi) - c(\pi) < p(\iota^+) - c(\iota^+) + p(\pi \setminus I^+) - c(\pi \setminus I^+)$, and then $d(\pi) < d(\iota^+) + d(\pi \setminus I^+)$, which contradicts the hypothesis.

There are then at least three cycles in $\mathcal{C}^+$, two of them being either in $\pi \setminus I^+$ or in ι^+. Suppose first that two are in ι^+. Let d_1 and d_2 be the two desire edges in these two cycles of $\mathcal{C}^+$ in ι^+. For an illustration see Figure 3. Then in ι^-, these two desire edges exchange one of their extremities, and they belong to the same cycle (the remaining of the cycles is unchanged). This makes $c(\iota^+) > c(\iota^-)$ which is in contradiction with the hypothesis that I should be neutral. There is therefore only one cycle of $\mathcal{C}^+$ in ι^+ (by the way $|\mathcal{C}| = 1$ and $|\mathcal{C}^+| = 3$).

In consequence, $\pi \setminus I^+$ has two cycles in $\mathcal{C}^+$. Let d_3 and d_4 be the two desire edges in the two cycles of $\mathcal{C}^+$. Then in $\pi \setminus I^-$, these two desire edges exchange one of their extremities, so they belong to the same cycle (the remaining of the cycles is unchanged). This makes $c(\pi \setminus I^+) > c(\pi \setminus I^-)$, which ends the proof. ■

IV. Perfect parsimonious sequences of reversals

As noticed in [8], the main difficulty in finding perfect sequences of reversals of minimum length (among all perfect sequences) is that it is sometimes impossible to decide whether to sort a particular interval positively or negatively. The last lemma of the previous section shows that in the case of parsimonious scenarios, this choice is constrained by the data.

Here we describe the general method to decide the existence of a perfect parsimonious sequence of reversals sorting a permutation π, and study the time complexity of the method.

We apply as a black box the usual techniques to sort permutations by reversals when there is no common intervals constraint, or to compute the reversal distance. One can see for instance [1], [5], [12] for fast methods to do so.

We divide the algorithm into two parts, one is the procedure to sort an oriented component (this is the main algorithmic issue), described in Algorithm 1, while the main procedure, which treats the general case, is described in Algorithm 2.

Algorithm 1: Sorting an oriented component
Input: *An oriented component C of a permutation π, with no proper sub-component which is not sorted.*
Output: *A perfect parsimonious scenario for C, or a certificate that proves it does not exist.*

While *C is not sorted*

For all *strong common intervals I included in C, in bottom-up order (such that I is not sorted but has only sorted strong sub-intervals)*

1) *Compute $d(\iota^+)$, $d(\iota^-)$, $d(\pi \setminus I^+)$, and $d(\pi \setminus I^-)$.*
2) **If** $d(\pi) = d(\iota^+)+d(\pi \setminus I^+)$, **then** *assign "+" to I;*
   If $d(\pi) = d(\iota^-)+d(\pi \setminus I^-)$, **then** *assign "-" to I;*
   If *I has no sign,* **then** *there is no perfect parsimonious scenario;*
   If *I has both signs,* **then** *it is not neutral:* **if** $d(\iota^+) < d(\iota^-)$, **then** *keep the "+" sign,* **else** *keep the "-" sign.*
3) **If** *I is prime,* **then** *sort the permutation ι^+ (or ι^- if I has a "-" sign) with the method of [12], and apply the same reversals to π.*
4) **If** *I is linear,* **then** *reverse all the children that are sorted negatively (or positively if I has a "-" sign).*

Algorithm 2: Parsimonious and perfect sorting by reversals
Input: *A permutation π.*
Output: *A perfect parsimonious scenario, or a certificate that proves it does not exist.*

Check that π is not a fool (see Lemma 1). If it is, there is no perfect parsimonious sorting sequence.
For all *components C of the permutation, in bottom-up order (such that every component included in C has already been sorted)*

1) **If** *C is oriented,* **then** *sort it with Algorithm 1.*
2) **If** *C is unoriented* **then**
   For all *reversals ρ that make C oriented, do not break any common interval, nor decrease $c(\pi)$,*
   Apply ρ to π
   Sort C (now oriented) with Algorithm 1.

Theorem 2: Algorithm 2 constructs a perfect parsimonious sequence of reversals sorting π if one exists, and runs in $O(n^4)$.

Proof: First, we prove that Algorithm 1 is correct, that is, it sorts an oriented component whenever possible. Observe that it requires that every proper sub-component has already been sorted, which is guaranteed by a bottom-up search over the components of π in Algorithm 2.

From Lemma 4, every time a strong interval is not given a sign, this means the interval is not perfect, so it is a sufficient condition for the inexistence of a perfect parsimonious scenario.

If an interval is given only one sign, then the direction in which it should be sorted is necessary, and any perfect parsimonious scenario would sort it this way, so if there is such a scenario, the algorithm finds it.

Now the only difficulty is the case when the interval I is given two signs. In this case, from Lemma 5, we have either $d(\iota^+) < d(\iota^-)$ or $d(\iota^-) < d(\iota^+)$. Suppose without loss of generality that $d(\iota^+) < d(\iota^-)$. We prove that if there is a perfect parsimonious scenario, then there is one sorting I in the positive direction. Indeed, suppose a perfect parsimonious scenario sorts I in the negative direction. It thus sorts I with $d(\iota^-)$ reversals, and then $\pi \setminus I^-$ with $d(\pi \setminus I^-)$ reversals. Then the scenario sorting I in the positive direction, reversing it, and then sorting $\pi \setminus I^-$ with the same reversals is also parsimonious, because its length is the same as for the optimal scenario, and perfectness is not changed by the reversal of a whole strong segment. There is therefore also a solution sorting I posi-

tively, and the algorithm finds it.

Lastly, in the case of a linear interval, the only allowed reversals are the reversals of a single child, because any other reversal would break a common interval. The described method, reversing all children with negative or positive signs according to the direction of the interval, is the method of [3] to sort linear intervals with a perfect scenario of minimum length. Thus it is a parsimonious one if a parsimonious one exists. This method may be applied to sort all linear intervals while checking that all operations are optimal for parsimony.

Now let us examine the theoretical time complexity of Algorithm 1. We prove it runs in time $O(k^2)$, where k is the number of breakpoints of the component C.

Step 1 is achieved in $O(n)$ time by reversal distance computations, with the methods of [1], [5]. Step 2 is done in constant time. Step 3 is the most costly, because it requires an algorithm to sort I. The most efficient ones can run in less than $O(k'^2)$ time [12], where k' is the number of breakpoints of I. The last Step 5 is linear in k', as mentioned in [3].

Overall, the four steps may be achieved in time $O(n) + O(k'^2)$, and the number of breakpoints of C is decreased by k' at the end of these steps. The total time complexity is therefore $O(n + k^2)$.

Now the correctness of Algorithm 2 remains to be stated. It sorts a permutation by examining every component one by one. As the permutation is not a fool, it can be decomposed into components, and each component can be treated separately. If a component is oriented, then the algorithm calls Algorithm 1, and if it is unoriented, it tries all reasonable possibilities to orient it. This is done only at the first step, so it still has a polynomial running time. The complete exploration guarantees the correctness of the method.

Algorithm 2 runs therefore in time $O(n^4)$, where n is the size of the permutation. Indeed, in the case of an unoriented component, every reversal in Step 2 of the algorithm has to be tried. There are at most $O(k^2)$ of them, where k is the number of breakpoints of the component C, and they are applied only in the first step, so in the case of unoriented components $O(k^2)$ calls to Algorithm 1 may be necessary. At the end, the number of breakpoints of π has decreased by k, because C is sorted. Thus the whole yields an $O(n^4)$ time complexity algorithm. ■

V. Perfect Sorting by Reversals

The algorithm of the previous section decides if there exists a perfect parsimonious scenario sorting a permutation. It is a particular case of the problem mentioned in [8] and proved NP-hard. It is a general case of the problem mentioned in [3] and proved polynomial. Those studies, together with [4], treated the problem of relaxing parsimony, and of designing perfect scenarios of reversals of minimum cardinality among all perfect scenarios. In [4], Bérard *et al.* also generalised the result in [3], and constructed a large class of permutations for which there exists a polynomial method to design perfect scenarios of minimum length. The question therefore arised of the relation between the study presented in the present paper and Bérard *et al.*'s class. In this section, we prove that our algorithm widens this latter class by adding to it examples that are treated by our method, but neither by [3] nor by [4].

In the tree of strong common intervals of a permutation, we call a node *ambiguous* if it is neutral and prime, and its parent is prime. Let $a(\pi)$ be the number of ambiguous nodes in a permutation. The following is an extrapolation from the results in [4].

Theorem 3: [4] There is a polynomial time algorithm for designing perfect scenarios of reversals of minimum length for all permutations where $a(\pi)$ is bounded by a poly-logarithmic function of the size of the permutation.

We show a class of permutations such that $a(\pi)$ is a linear function of their size, and that admit a perfect parsimonious scenario. This means our algorithm computes in polynomial time a perfect scenario of reversals, while the one of [4] is exponential. This widens the class of permutations for which there exists a polynomial algorithm that finds a minimal length perfect scenario of reversals.

Let K be any positive integer. We define for any K a permutation π_K of size $n = 4 \times K + 1$.

- $\pi_K(1) = -(K + 1)$.
- For all $i = 1 \dots K$, $\pi_K(4i - 2) = 3K + i + 1$,
- For all $i = 1 \dots K$, $\pi_K(4i - 1) = -(K + 2i)$,
- For all $i = 1 \dots K$, $\pi_K(4i) = i$,
- For all $i = 1 \dots K$, $\pi_K(4i{+}1) = -(K{+}2i{+}1)$.

For instance, $\pi_1 = (-2\ \ 5\ \ -3\ \ 1\ \ -4)$, $\pi_2 = (-3\ \ 8\ \ -4\ \ 1\ \ -5\ \ 9\ \ -6\ \ 2\ \ -7)$ and $\pi_3 = (-4\ \ 11\ -5\ \ 1\ -6\ \ 12\ -7\ \ 2\ -8\ \ 13\ -9\ \ 3\ -10)$.

We let the reader check that a permutation is

indeed defined for every K, and that $d(\pi_K) = 2K+1$ for all K. Indeed, the breakpoint graph has $2K+1$ cycles, each of size 4. There is no common interval in this permutation other than the singletons and the set of all numbers.

Now take any negative number $-i$ of π_K, and replace it in the permutation by $2+i \quad -i \quad 3+i \quad 1+i$, which is an ambiguous common interval. Shift the remaining of the permutation by adding 3 (in absolute value) to every number greater than i. For instance, if $K=1$, $i=2$, the result is $(4 \ -2 \ 5 \ 3 \ 8 \ -6 \ 1 \ -7)$. Note that the number of cycles has decreased by one through this operation.

Do the same for all the negative numbers of π_K except one. Let Π_K be the resulting permutation. It has size $10K+1$. For instance, $\Pi_1 = (4 \ -2 \ 5 \ 3 \ 11 \ 8 \ -6 \ 9 \ 7 \ 1 \ -10)$. It has $2K$ ambiguous intervals, which are all the intervals added from π_K. It has a perfect parsimonious scenario, because all the intervals $2+i \quad -i \quad 3+i \quad 1+i$ can be sorted negatively in 4 reversals, which transforms Π_K into π_K in $8K$ reversals. There is then a perfect scenario for sorting Π_K in $10K+1$ reversals. Furthermore, the breakpoint graph of Π_K has only one cycle: indeed, every replacement of a negative number by the sequence of four numbers has the effect of decreasing the number of cycles by one, so doing this for every negative number except one decreases the number of cycles up to one. So by Theorem 1, $d(\Pi_K) = 10K+1$.

Those permutations have therefore a linear number of ambiguous intervals, and are sorted in polynomial time by our algorithm. This proves that together with [4], we construct a bigger class of permutations for which perfect sorting by reversals results in a polynomial problem.

VI. Experimental results

We implemented the algorithm described in Section IV and tested it on gene order data from several mammalian species. The implementation[1] was realised in Java, since simplicity was our concern rather than speed. The high theoretical complexity ($O(n^4)$) is not a constraint given the size of the tested data. The implementation thus provides the result nearly immediately for all tests.

The algorithm decides whether there exists a perfect parsimonious scenario of reversals; if yes, the sequence is given, in the other case, the algorithm stops when a contradiction is found.

We then extended the algorithm presented in Section IV by enabling it to continue even if a perfect parsimonious scenario does not exist. The principle of the extension is simple: every time the algorithm stops because the only choice to continue would be to break a common interval, this

[1] `http://biomserv.univ-lyon1.fr/~tannier/PSbR`

TABLE I

Experimental results: columns 1 and 2 give the compared chromosomes; column 3 indicates whether there exists a perfect parsimonious scenario or not (the result of the algorithm of Section IV); column 4 shows the upper bound to the number of strong common intervals that have to be broken in a parsimonious scenario (given by the extension): it is 0 if the previous answer is "yes"; column 5 indicates the reversal distance between the two chromosomes, and column 6, the minimum number of reversals in a perfect scenario (given by our implementation of the algorithm of [4]).

Chromosome 1	Chromosome 2	perf. ?	#segs brok.	d	#rev. perf.
Human 1	Chimp 1	no	3	28	30
Human 2	Chimp 12	yes	0	11	11
Human 2	Chimp 13	no	1	8	11
Human 3	Chimp 2	yes	0	3	3
Human 4	Chimp 3	yes	0	29	29
Human 5	Chimp 4	no	1	36	37
Human 6	Chimp 5	yes	0	6	6
Human 7	Chimp 6	no	1	23	24
Human 8	Chimp 7	yes	0	10	10
Human 9	Chimp 11	yes	0	20	20
Human 10	Chimp 8	yes	0	16	16
Human 11	Chimp 9	no	1	15	16
Human 12	Chimp 10	yes	0	12	12
Human 13	Chimp 14	yes	0	3	3
Human 14	Chimp 15	yes	0	3	3
Human 15	Chimp 16	yes	0	9	9
Human 16	Chimp 18	no	2	15	16
Human 17	Chimp 19	yes	0	15	15
Human 18	Chimp 17	yes	0	5	5
Human 19	Chimp 20	no	1	34	35
Human 20	Chimp 21	yes	0	12	12
Human 21	Chimp 22	yes	0	3	3
Human 22	Chimp 23	no	2	19	20
Human X	Chimp X	no	1	24	25
Human Y	Chimp Y	yes	0	3	3
Human X	Dog X	yes	0	48	48
Human X	Mouse X	no	5	62	69
Human X	Rat X	no	4	52	56
Mouse X	Rat X	no	6	65	71
Mouse X	Chimp X	yes	0	31	31
Mouse X	Dog X	yes	0	51	51
Rat X	Chimp X	yes	0	38	38
Rat X	Dog X	no	2	54	59
Chimp X	Dog X	yes	0	29	29

operation is performed and the algorithm is then allowed to run on. Since intervals are considered in a bottom-up inclusion-wise order, this tends to break preferentially small intervals, which may be considered less significant than bigger ones.

Regarding the tree of strong common intervals, the extended algorithm deletes a node and replaces it by joining its children to the parental node while preserving their order, wherein the elements are sorted ignoring the former interval.

A second modification we introduced is linked to the occurrence of *fool* permutations. Since common intervals model clusters of genes that must stay together, the reversals associated to a *hurdle merging* do not seem very probable from a biological point of view. We therefore never break framed intervals, and sort them always the same way, even if in this case the resulting sequence is not optimal in the sense of parsimony.

The data we used for the tests consisted of reciprocal-best-hit (RBH) ortholog assignments taken from the GeM knowledge database[2], which contains gene order information on the genomes of human, mouse, rat, chimp, dog (in draft form) and other vertebrate species.

The RBH method is well adapted to our model, because it provides a one to one correspondence between genes, and data are immediately transcribed into permutations. However, it often provides false positives because of duplication events. There are also genes with a position that very unlikely is the result of a reversal. We then applied a pre-processing to smooth the permutations by deleting putative false orthologs or genes with positions reflecting other rearrangement events than reversals. The pre-processing consists in the following: if a single gene is found between two blocks of contiguous genes in one permutation while these blocks are direct neighbours in the other permutation, the "intruder" gene is deleted in both permutations. This pre-processing should be in the future replaced, for a cleaner process, by better methods of identification of orthologs, and detection of false positive in the RBHs. This is one of the possibilities for future work.

The tests focussed on genomic data from man and chimp, since interchromosomal rearrangements are rare between these two species, and on the gonosomes of man, chimp, mouse, rat and dog, since they do not rearrange much with autosomes. The data are therefore coherent with our single chromosome model. We compared our method and the extension of it to the one of [4], that we re-implemented for our purpose. When a perfect parsimonious scenario exists, the results of both algorithms are identical: in contrast to the method of [4], our method simply guarantees a polynomial running time. In the other case (there is no perfect parsimonious scenario), the algorithm in [4] increases the number of reversals, while our method may break some intervals but maintains a parsimonious solution. The results are shown in Table I. Column 3 indicates whether a perfect parsimonious scenario exists. Column 4 shows the number of intervals that have been broken by our heuristic in order to have a parsimonious scenario. Column 5 is the number of reversals in a parsimonious scenario, and Column 6 is the minimum number of reversals in a perfect scenario, computed with our implementation of the algorithm of [4]. Observe that two lines are not necessarily comparable, since the number of extracted orthologous genes is different for all the couples. Permutations are of diverse sizes, all the genes are not common to all the species, so the triangle inequality is not verified and these distances may not be used to infer phylogenies.

For the closely related species human and chimp, the existence of perfect parsimonious scenarios is observed for most chromosomes. For the chromosomes for which such scenarios do not exist, most of the time only one small interval has to be broken. In this case, we find exactly the minimum number of strong intervals that have to be broken, so the extension of the algorithm gives an exact solution. Sometimes it is necessary to add several inversions in order to preserve only one small interval (see human 2 against chimp 13). In these cases, the parsimonious solution may be more likely than the perfect one. In other cases, like in the comparison of the mouse and rat X chromosomes, some big intervals have to be broken in order to obtain a parsimonious scenario, and maybe some biologically relevant conservations are not preserved.

The existence of a perfect parsimonious scenario is correlated with the number of reversals in a scenario, which is expected. In some cases, even distant species like mouse and chimp have such a scenario. Then the algorithm discriminates among

[2] `http://pbil.univ-lyon1.fr/gem/gem_home.php`

all possible parsimonious scenarios, and proposes a perfect one, which may be seen as a more likely candidate than a non-perfect scenario of evolution. The most problematic cases are those which involve murine species, for which there are many branch-specific rearrangements. In this case, it has to be decided if one trusts the parsimony principle in relation to the number of events, or the preservation of common intervals, because obviously both are not compatible.

VII. Conclusion

We designed a polynomial algorithm to decide the existence of a perfect parsimonious sequence of reversals sorting a permutation, thus answering an open question mentioned in the literature. We showed that this widens the known class of permutations for which perfect sorting by reversals is solvable in polynomial time. We also extended the algorithm to a heuristic that sorts a permutation if no such perfect sorting sequence exists.

We implemented our method and tested it on several chromosomes of mammalian species by comparing what we obtain to the results of another similar method in the literature. In certain cases, the space of optimal solutions is reduced by this additional constraint. In other cases, we observe an incompatibility between the principle of reconstructing evolutionary events with parsimony methods, and preserving co-localised clusters of genes.

One future direction would be to extend our model to be able to handle multi-chromosomal genomes and duplicated genes. The latter might lead to better methods than reciprocal best hit to identify orthologies, and thus enable to address a wider range of data.

Acknowledgment

We would like to thank Vincent Navratil for helping us to retrieve the data from the GeM database, Cédric Chauve for putting to us the question about the existence of the family of permutations constructed in Section V, and the anonymous referees for their comments that have given to the paper its final form.

References

[1] Bader, D.A., Moret, B.M.E., and Yan, M., "A linear-time algorithm for computing inversion distances between signed permutations with an experimental study", *J. Comput. Biol.* 8, 5 (2001), 483-491.

[2] Beal M.-P., Bergeron A., Corteel S. and Raffinot, M., "An Algorithmic View of Gene Teams", *Theor. Comput. Sci.* 320(2-3):395-418, 2004.

[3] Bérard S., Bergeron A. and Chauve C., "Common structures in evolution scenarios", 2nd RECOMB Comparative Genomics Satellite Workshop, *Lecture Notes in BioInformatics*, 3388:1-15, 2004.

[4] Bérard S., Bergeron A. and Chauve C., Paul C., "Perfect Sorting by Reversals is not always difficult", proceedings of WABI 2005, *Lecture Notes in Computer Science* 3692:228-237, 2005.

[5] Bergeron A., Mixtacki J. and Stoye J., "The inversion distance problem", in *Mathematics of evolution and phylogeny* (O. Gascuel Ed.) Oxford University Press, 2005.

[6] Bourque G., Pevzner P. and Tesler G., "Reconstructing the genomic architecture of ancestral Mammals: Lessons from human, mouse and rat genomes", *Genome Research* 14:507-516, 2004.

[7] Bui Xuan B.-M., Habib M. and Paul C., "Revisiting T. Uno and M. Yagiura's Algorithm", Proceedings ISAAC 2005, *Lecture Notes in Computer Science* 3827:146-155, 2005.

[8] Figeac M. and Varré J.-S., "Sorting by reversals with common intervals", proceedings of WABI 2004, *Lecture Notes in Computer Science* 3240, 26-37, 2004.

[9] Hannenhalli S. and Pevzner P. , "Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals", Journal of the ACM, 46:1– 27, 1999.

[10] Heber S. and Stoye J., "Finding all Common Intervals of k Permutations", Proceedings of CPM 2001, *Lecture Notes in Computer Science*, vol. 2089, 207-218, 2001.

[11] Sagot M.-F. and Tannier E., "Perfect Sorting by Reversals", proceedings of COCOON 2005, *Lecture Notes in Computer Science* 3595, 42-51, 2005.

[12] Tannier E., Bergeron A. and Sagot M.-F., "Advances on Sorting by Reversals", to appear in *Discrete Applied Mathematics*, 2006.

Yoan Diekmann is studying "Naturwischenschaftliche Informatik" (computer science) at the Technische Fakultät of the University of Bielefeld, Germany. He achieved this work when staying for an internship at the University of Marne-La-Vallée. His research interest are in computational biology.

Marie-France Sagot received the BSc degree in computer science from the university of Sao Paulo, Brazil, in 1991, the PhD degree in theoretical computer science and applications from the University of Marne-la-Vallée, France, in 1996, and the Habilitation from the same university in 2000. From 1997 to 2001, she worked as a research associate at the Pasteur Institute in Paris, France. In 2001, she moved to Lyon, France, as a research associate at the INRIA. She is now 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, in Villeurbanne. Her research interests are in computational biology, algorithmics and combinatorics.

Eric Tannier got his PhD in discrete mathematics from the University of Grenoble 1. He is now 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, in Villeurbanne. His research interests are in comparative genomics and combinatorics.