The solution space of sorting by reversals

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

The Solution Space of Sorting by Reversals

Marília D.V. Braga[1], Marie-France Sagot[1], Celine Scornavacca[2], and Eric Tannier[1]

[1] INRIA Rhône-Alpes, Laboratoire de Biométrie et Biologie Évolutive (UMR 5558), CNRS, Univ. Lyon 1
43 bd 11 Nov, 69622, Villeurbanne Cedex, France
marilia@biomserv.univ-lyon1.fr,
{Marie-France.Sagot,Eric.Tannier}@inrialpes.fr
[2] Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier, 34392 Montpellier Cedex 5 - France
Celine.Scornavacca@lirmm.fr

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.* [4] 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, no algorithm exists so far to compute this set of representatives except through the enumeration of all solutions, which takes too much time even for small permutations. Bergeron *et al.* [4] state as an open problem the design of such an algorithm. 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 several biological examples where the result is more relevant than a unique optimal solution or the list of all solutions[1].

1 Introduction

The combinatorics of genome rearrangements is a very prolific domain of computational biology. It consists in, given a set of actual genomes, inferring the large-scale evolutionary mutations that explain the differences in the organisation of those genomes. For a general survey of the algorithmic aspects of genome rearrangements, see [13].

One of the most used mathematical models for representing and manipulating such genome rearrangements is given by signed permutations, where the elements

[1] An implementation of the algorithm is available online, as part of the BaobabLuna package, at www.geocities.com/mdvbraga/baobabLuna.html

I. Măndoiu and A. Zelikovsky (Eds.): ISBRA 2007, LNBI 4463, pp. 293–304, 2007.

are unique homologous markers, and reversals as the main events that may alter the order of the markers along the genomes. The combinatorial problem consists then in giving a 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, [2,5,17,12,8,11]), 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 [14] 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 can not 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 favouring small inversions [1], or inversions that do not cut some clusters of co-localised genes [8,3]. The number of solutions is decreased, but the whole set of solutions is never handled.

Bergeron *et al.* [4] 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.

The paper is organised 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, and on an analysis of the performances of our implementation.

2 Sorting by Reversals and Its Solution Space

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, by a permutation of the set $\{1, \ldots, n\}$, where each number is, in addition, given a sign '+' or '-' (the sign '+' is usually omitted). 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* 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.

Sorting by reversals. Given a permutation π and an interval ρ, 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 $\rho = \{|\pi_i|, \ldots, |\pi_j|\}$,

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

Due to this, an interval ρ can also be used to denote a reversal. We may always represent an interval or reversal ρ as the sorted set of its values.

If $\rho_1, \ldots, \rho_k$ is a sequence of intervals or reversals, we say that it *sorts* a permutation π if $\pi \cdot \rho_1 \cdots \rho_k = Id$. The length of a shortest sequence of reversals sorting a permutation π is called the *reversal distance* of π, and is denoted by $d(\pi)$. A 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 appeared in [12], while the fastest algorithm to compute the distance was given in [2], and the one to find an optimal sequence can be retrieved from a compilation of [5,11,17].

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 8278540, 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 [14], that, given a permutation, computes all the reversals that are the first step of an optimal sequence, but in the aforementioned example, listing the 8278540 sequences is almost as useless as giving only one of them.

Traces. More interesting for our study is the representation of the set of solutions that is given in [4]. Recall a reversal is written as a subset of $\{1, \ldots, n\}$, where the elements are ordered increasingly, so that they can be compared by a lexicographic order. Identifying a sequence of reversals with a word on the alphabet $\mathcal{A}$ of reversals, the authors of [4] define an equivalence relation on these words: if ρ and θ are reversals (intervals) and do not overlap, then the words $\rho\theta$ and $\theta\rho$ are equivalent. We say that ρ and θ *commute*. Under this relation, any two words containing $\rho\theta$ as a subword are equivalent to the same word, replacing the subword $\rho\theta$ by $\theta\rho$.

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 $\{1, 2\}$ commute, so $\{1\}\{1, 2\}\{4\}\{1, 2, 3, 4\}$ is equivalent to $\{1\}\{4\}\{1, 2\}\{1, 2, 3, 4\}$, and as every pair of reversals also commute, every permutation of these four reversals is a solution.

Now if the solution $\{1, 3, 4\}\{2, 4\}\{2, 3\}\{3\}$ is considered, then it is equivalent to $\{1, 3, 4\}\{2, 4\}\{3\}\{2, 3\}$, $\{1, 3, 4\}\{3\}\{2, 4\}\{2, 3\}$ and $\{3\}\{1, 3, 4\}\{2, 4\}\{2, 3\}$ by commutation of $\{3\}$ with all the other reversals, which do not commute among themselves.

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 [7]. It is particularly relevant in our study because of the following result proven in [4].

Proposition 1. *[4] Let π be a signed permutation. The set of all optimal sequences of reversals sorting π 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.

Normal form of a trace. A trace T is thus a set of equivalent words over an alphabet $\mathcal{A}$. An element s of T is said to be in *normal form* if it can be decomposed into subwords $s = u_1|\dots|u_m$ such that:

- every pair of elements of a subword u_i commute;
- for every element ρ of a subword u_i $(i > 1)$, there is at least one element θ of the subword u_{i-1} such that ρ and θ do not commute;
- every subword u_i is a nonempty increasing word under the lexicographic order induced by $\mathcal{A}$

A theorem by Cartier and Foata (cited in [4]) states that, for any trace, there is a unique word 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.

The algorithmics of traces. Bergeron *et al.* [4] 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.

We give in this paper 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

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. We then present our algorithm, and make a comparison between the two.

3.1 The Enumeration of the Solutions

A sequence of reversals $s = \rho_1\rho_2\dots\rho_i$ 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 [14]. It has time complexity $O(n^3)$, and the number of possible optimal 1-sequences is bounded by $\frac{n(n+1)}{2} \leq n^2$.

The set of all optimal i-sequences can then be computed from the set of $(i-1)$-sequences by iterating the same algorithm for finding all 1-sequences. The set of i-sequences has therefore size at most $O(n^{2i})$, and the algorithm has time complexity 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 construct the normal form of the trace for each sorting sequence, and then to group the sorting sequences by 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 $s = \rho_1\rho_2 \dots \rho_d$ for a permutation π with reversal distance d, the normal form of the trace T that contains s is constructed by iterating an integer i from 1 to d and, at each step i, adding the element ρ_i, represented as the sorted set of its values, to the normal form of the $(i-1)$-trace containing $\rho_1 \dots \rho_{i-1}$ (the initial 0-trace is an empty trace). This procedure is described by Algorithm 1.

Algorithm 1. Adding an element to a normal form of a trace

Require: An $(i-1)$-trace $u_1|u_2|\dots|u_k$ and the next element ρ_i
Ensure: The normal form of the i-trace containing the element $u_1u_2\dots u_k\rho_i$

Let j be the maximum index such that u_j contains an element that does not commute with ρ_i, or 0 if such a u_j does not exist
if $j = k$ **then**
  Add a new subword $u_{k+1} \leftarrow \rho_i$
else
  Add ρ_i to the subword u_{j+1}, according to the lexicographic order
end if

As the reversal distance and the interval size are bounded by n, the procedure has complexity $O(n^2 \log n)$, considering that each reversal or interval has 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 $O(n \log N)$ operations, where N is the number of represented traces. As N is bounded by the number of solutions, we have $n \log N \leq n \log(n^{2n}) = 2n^2 \log n$.

Eventually, the total time complexity for enumerating all the normal forms of the traces is bounded by $O(n^{2n+3}) + O(n^{2n}(n^2 \log n + 2n^2 \log n)) = O(n^{2n+3})$.

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 is implemented, for example, in the GRAPPA software[2], 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 The Enumeration of the Traces

A k-trace T' is a *prefix* of an i-trace T ($k \leq i$) if T' contains a k-sequence which is a prefix of an i-sequence of T. It is equivalent [7] to saying that each k-sequence of T' 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 remark 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 $(i+1)$-trace from the one of an i-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.

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 the 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 by the algorithm of Siepel [14] 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 the algorithm of Siepel [14].

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 are computed. Let $\rho_1, \ldots, \rho_k$ be the reversals that are in the last position of at least one element in T. Let x_j be the number of elements of T which have ρ_j as their last position. Then the number of elements 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. So by the induction hypothesis, the size of the trace T is the sum of the sizes

[2] `http://www.cs.unm.edu/~moret/GRAPPA/`. We re-implemented 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.

Algorithm 2. Enumerating all the traces of a signed permutation

Require: A signed permutation π

Ensure: The normal form and size $(norm(T), size(T))$ of each trace T of optimal sequences of reversals for sorting π

```
d ← reversal distance of π
S ← {ρ | ρ is an optimal 1-sequence for π}/* Algorithm of Siepel [14] */
𝒯 ← ∅
for each reversal ρ ∈ S do
  norm(T) ← ρ /* T is a 1-trace */
  size(T) ← 1
  Insert {(norm(T), size(T))} in 𝒯
end for
for each integer i from 2 to d do
  𝒯_next ← ∅ /* contains the normal forms of all the i-traces */
  for each (norm(T), size(T)) in 𝒯 /* T is a (i − 1)-trace */ do
    Let π_T be the resulting permutation after applying the (i−1)-sequence norm(T)
    to π
    S ← {ρ | ρ is an optimal 1-sequence for π_T}/* Algorithm of Siepel [14] */
    for each reversal ρ ∈ S do
      norm(T_+ρ) ← norm(T) + ρ /* Algorithm 1 */
      size(T_+ρ) ← size(T)
      if there is (norm(T_+), size(T_+)) ∈ 𝒯_next such that norm(T_+) = norm(T_+ρ)
      then
        size(T_+) ← size(T_+ρ) + size(T_+)
      else
        Insert (norm(T_+ρ), size(t_+ρ)) in 𝒯_next
      end if
    end for
  end for
  𝒯 ← 𝒯_next
end for
return 𝒯 /* 𝒯 is the final set of d-traces */
```

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 $\rho_1 \dots \rho_n$ by a partial ordering of the set $\mathcal{P}_T = \{(\rho_i, k_i)\}_i$, where ρ_i is an element of $\mathcal{A}$ appearing in

$\rho_1 \dots \rho_n$ and k_i is the number of occurrences of ρ_i in the subword $\rho_1 \dots \rho_i$. The relation $<_T$ is defined as the transitive closure of the relation $\lhd$ itself defined by $(\rho_i, k_i) \lhd (\rho_j, k_j)$ if and only if $i < j$ and ρ_i and ρ_j do not commute.

In other words, $(\rho_i, k_i) <_T (\rho_j, k_j)$ if and only if the k_i^{th} ρ_i is always before the k_j^{th} ρ_j in the elements of T (see [7]).

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

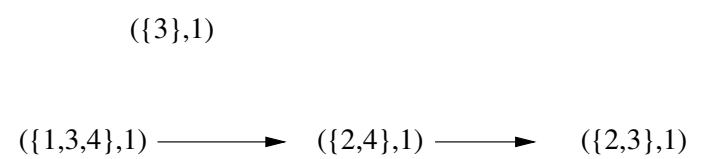

Fig. 1. The poset constructed from the normal form $T = \{1,3,4\}\{3\}|\{2,4\}|\{2,3\}$. All the linear extensions of this poset are the optimal sequences of reversals belonging to the trace represented by T.

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 verifies $\rho <_T \theta \Rightarrow \rho <_{tot} \theta$. The set of all linear extensions of $\mathcal{P}_T, <_T$ is exactly the set of elements of the trace T (see [7]). 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 subwords 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 subword 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 [10] to a bipartite matching problem.

The representation of a trace as a poset allows 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 organisation 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 [15].

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.

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 following $(i+1)$-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 [6], and the best known algorithms run in $O(n^k)$, where n is the size of the poset and k is its width [16]. 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 Figure 2) which show that in practice, this parameter is often lower, which explains the speed-up of our algorithm compared to a total enumeration procedure.

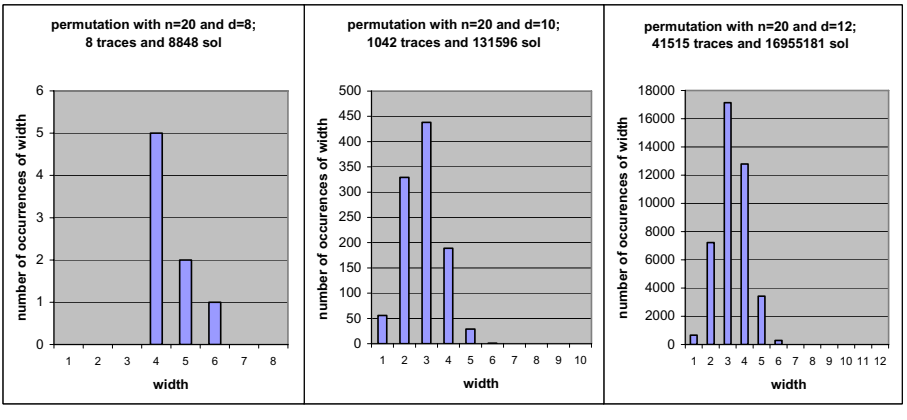

Fig. 2. Distribution of the width of posets on random permutations

4 Experimental Results and Applications

We implemented our algorithm and tested it on randomly simulated permutations, as well as on biological data[3]. Some results are recorded in Figure 3. 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 permutations that are 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.

5 Conclusions, Limitations and Perspectives

We 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.

PERMUTATION	N_s	N_t	Enum. sol.	Enum. + traces	BaobabLUNA
rat/human chr X n =16 d=10	2419750	418	$\simeq$ 10 min	$\simeq$ 13 min	$\simeq$ 10 s
mouse/human chr X n =16 d=10	3362310	218	$\simeq$ 12 min	$\simeq$ 18 min	$\simeq$ 10 s
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
Chr X/Chr Y human n =30 d=12	207600628	115512	$>$ 24 h	$>$ 48 h	$\simeq$ 28 min

Fig. 3. Computation results. 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; 6- the execution time of the enumeration of the traces, according to Algorithm 2.

[3] The first two permutations, that model the organisation of the X chromosomes of human, mouse and rat, are taken from [3]; the permutation `Chr X/Chr Y human`, modelling the comparison of the human X and Y chromosomes, is a simplified version of a set of markers coming from an ongoing study at the LBBE laboratory, University of Lyon 1, France.

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 $d(\pi)$ inferior to 20, on a personal computer that has 1Gb random access memory[4]. It is sufficient for some biological applications, as we show it here on the data from X chromosomes of mammalian species, or on the comparison of the human X and Y chromosomes (see Figure 3). For many datasets 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. Probably another structure remains to be invented in order to solve a similar problem for large permutations.

References

1. Ajana Y., Lefebvre J.F., Tillier E., El-Mabrouk N., "Exploring the set of all minimal sequences of reversals - An application to test the replication-directed reversal hypothesis". Second International Workshop, Algorithms in Bioinformatics (WABI'02), LNCS 2452, R. Guigo and D. Gusfield eds., pp. 300-315, September 2002.
2. 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.
3. Berard S., Bergeron A., Chauve C. and Paul C. "Perfect sorting by reversals is not always difficult", to appear in *IEEE transactions on cioinformatics and computational biology*, 2006.
4. Bergeron A., Chauve C., Hartmann T., St-Onge K., "On the properties of sequences of reversals that sort a signed permutation". JOBIM 2002, 99-108.
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. Brightwell G. and Winkler P., "Counting linear extensions is #P-complete", *STOC '91: Proceedings of the twenty-third annual ACM symposium on Theory of computing*, 1991, ACM Press.
7. Diekert V. Rozenberg G. (eds) *The book of traces*, World Scientific, 1995.
8. Diekmann Y., Sagot M.F. and Tannier E., "Evolution under reversals: parsimony and preservation of common intervals", to appear in *IEEE/ACM transactions in computational biology and bioinformatics*, 2006 (A preliminary version appeared in COCOON 2005, *Lecture Notes in Computer Science* 3595, 42-51, 2005).
9. Dilworth R.P., "A Decomposition Theorem for Partially Ordered Sets", Annuals of Mathematics 51 (1950) pp. 161-166.
10. Fulkerson D.R., "Note on Dilworth's decomposition theorem for partially ordered sets", Proc. Amer. Math. Soc. 7 (1956), 701–702

[4] 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.

11. Han Y, "Improving the Efficiency of Sorting by Reversals", Proceedings of The 2006 International Conference on Bioinformatics and Computational Biology, CSREA Press, Las Vegas, Nevada, USA, 2006.
12. 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.
13. Li Z., Wang L. and Zhang K., "Algorithmic approaches for genome rearrangement: a review", *IEEE transactions on systems, man and cybernetics*, 36:636–648, 2006.
14. Siepel A. "An algorithm to enumerate sorting reversals for signed permutations". J Comput Biol 10:575-597, 2003.
15. Steiner G., "An algorithm to generate the ideals of a partial order" Operations Research Letters, 5(6):317 – 320, 1986.
16. Steiner G., "Polynomial algorithms to count linear extensions in certain posets". Congressus Numerantium, 75, 71-90, 1990
17. Tannier E., Bergeron A. and Sagot M.-F., "Advances on Sorting by Reversals", to appear in *Discrete Applied Mathematics*, 2006 (A preliminary version appeared in CPM 2004, *Lecture Notes in Computer Science* 3595, 42-51).