

On Sorting by Translocations

ANNE BERGERON,¹ JULIA MIXTACKI,² and JENS STOYE³

ABSTRACT

The study of genome rearrangements is an important tool in comparative genomics. This paper revisits the problem of sorting a multichromosomal genome by translocations, i.e., exchanges of chromosome ends. We give an elementary proof of the formula for computing the translocation distance in linear time, and we give a new algorithm for sorting by translocations, correcting an error in a previous algorithm by Hannenhalli.

Key words: comparative genomics, genome rearrangement, translocation distance, sorting by translocations.

1. INTRODUCTION

WE REVISIT THE PROBLEM OF SORTING MULTICHROMOSOMAL GENOMES by translocations that was introduced by Kececioğlu and Ravi (1995) and Hannenhalli (1996): Given two genomes A and B , the goal is to find a shortest sequence of exchanges of nonempty chromosome ends that transforms A into B . The length of such a shortest sequence is the translocation distance between A and B , and the problem of computing this distance is called the translocation distance problem.

The study of genome rearrangements allows one to better understand the processes of evolution and is an important tool in comparative genomics. However, the combinatorial theories that underly rearrangement algorithms are complex and prone to human errors (Ozery-Flato and Shamir, 2003; Tesler, 2002).

Given their prevalence in eukaryotic genomes (Mouse Genome Sequencing Consortium, 2002), a good understanding of translocations is necessary. Using tools developed in the context of sorting two signed genomes by inversions, we establish on solid grounds Hannenhalli's equation for the translocation distance and give a new algorithm for sorting by translocations.

Restricting genome rearrangements to translocations only might look, at first glance, like a severe constraint. However, mastering the combinatorial knowledge of a single operation is always a step towards a better understanding of the global picture. As more and more genomes are decoded, sound mathematical models and correct algorithms will play a crucial role in analyzing them.

The next section introduces the basic background needed in the following. The third section gives a counterexample to Hannenhalli's algorithm. Section 4 presents a new proof and formula for the translocation distance, and Section 5 discusses the algorithms.

¹Département d'informatique, Université du Québec à Montréal, Canada.

²International NRW Graduate School in Bioinformatics and Genome Research, Center of Biotechnology, Universität Bielefeld, Germany.

³Technische Fakultät, Universität Bielefeld, Germany.

2. DEFINITIONS AND EXAMPLES

2.1. Genes, chromosomes, and genomes

As usual, we represent a *gene* by a signed integer where the sign represents its orientation. A *chromosome* is a sequence of genes and does not have an orientation. A *genome* is a set of chromosomes. We assume that each gene appears exactly once in a genome. If the k -th chromosome in a genome A of N chromosomes contains m_k genes, then the genes in A are represented by the integers $\{1, \dots, n\}$ where $n = \sum_{k=1}^N m_k$:

$$A = \{(a_{11} \ a_{12} \ \dots \ a_{1m_1}), (a_{21} \ a_{22} \ \dots \ a_{2m_2}), \dots, (a_{N1} \ a_{N2} \ \dots \ a_{Nm_N})\}.$$

For example, the following genome consists of three chromosomes and nine genes:

$$A_1 = \{(4 \ 3), \ (1 \ 2 \ -7 \ 5), \ (6 \ -8 \ 9)\}.$$

For an interval $I = (a_i \ \dots \ a_j)$ of elements inside a chromosome, we denote by $-I$ the reversed interval where the sign of each element is changed, i.e., $-I = (-a_j \ \dots \ -a_i)$. Since a chromosome does not have an orientation, we can *flip* the chromosome $X = (x_1 \ x_2 \ \dots \ x_k)$ into $-X = (-x_k \ \dots \ -x_2 \ -x_1)$ and still have the same chromosome. More precisely, let us consider two chromosomes X and Y . We say that a chromosome X is *identical* to a chromosome Y if either $X = Y$ or $X = -Y$. Genomes A and B are *identical* if for each chromosome contained in A there is an identical chromosome in B and vice versa.

A *translocation* transforms the chromosomes $X = (x_1 \ \dots \ x_i \ x_{i+1} \ \dots \ x_k)$ and $Y = (y_1 \ \dots \ y_j \ y_{j+1} \ \dots \ y_l)$ into new chromosomes $(x_1 \ \dots \ x_i \ y_{j+1} \ \dots \ y_l)$ and $(y_1 \ \dots \ y_j \ x_{i+1} \ \dots \ x_k)$. It is called *internal* if all exchanged chromosome ends are nonempty, i.e., $1 \leq i < k$ and $1 \leq j < l$.

Given a chromosome $X = (x_1 \ x_2 \ \dots \ x_k)$, the elements x_1 and $-x_k$ are called its *tails*. Two genomes are *co-tailed* if their sets of tails are equal. Note that an internal translocation does not change the set of tails of a genome.

In the following, we assume that the elements of each chromosome of the target genome B are positive and in increasing order. For example, we have that

$$\begin{aligned} A_1 &= \{(4 \ 3), \ (1 \ 2 \ -7 \ 5), \ (6 \ -8 \ 9)\}, \\ B_1 &= \{(1 \ 2 \ 3), \ (4 \ 5), \ (6 \ 7 \ 8 \ 9)\}. \end{aligned}$$

The *sorting by translocations problem* is to find a shortest sequence of translocations that transforms one given genome A into the genome B . We call the length of such a shortest sequence the *translocation distance* of A and denote this number by $d(A)$. The problem of computing $d(A)$ is called the *translocation distance problem*.

In the following, we will always assume that translocations are internal. Therefore, in the sorting by translocations problem, genomes A and B must be co-tailed.

Translocations on a genome can be simulated by inversions of intervals of signed permutations (see Hannenhalli and Pevzner [1995], Tesler [2002], and Ozery-Flato and Shamir [2003]). For a genome A with N chromosomes, there are $2^N N!$ possible ways to chain the N chromosomes; each of these is called a *concatenation*. Given a concatenation, we extend it by adding a first element 0 and a last element $n + 1$. This results in a signed permutation P_A on the set $\{0, \dots, n + 1\}$:

$$P_A = (0 \ a_{11} \ a_{12} \ \dots \ a_{1m_1} \ a_{21} \ a_{22} \ \dots \ a_{2m_2} \ \dots \ a_{N1} \ a_{N2} \ \dots \ a_{Nm_N} \ n + 1).$$

An *inversion* of an interval reverses the order of the interval while changing the sign of all its elements. We can model translocations on the genome A by inversions on the signed permutation P_A . Sometimes it is necessary to flip a chromosome. This can also be modeled by the inversion of a chromosome, but does not count as an operation in computing the translocation distance since the represented genomes are identical. See Fig. 1 for an example.

In the following sections, we consider several concepts such as elementary intervals, cycles, and components that are central to the analysis of the sorting by translocation problem. These concepts were originally developed for the analysis of the inversion distance problem. The notation follows Bergeron *et al.* (2005).

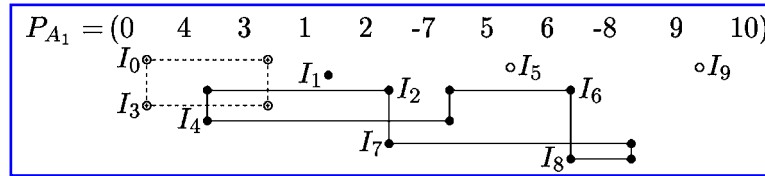


FIG. 2. Elementary intervals and cycles of the signed permutation P_{A_1} .

Definition 3. A black (or white) cycle is a sequence of breakpoints that are linked by black (respectively, white) elementary intervals. Adjacencies define trivial cycles.

The elementary intervals and cycles of our example permutation P_{A_1} are shown in Fig. 2.

The white cycles formed by the $N + 1$ white points depend on the concatenation. Since the order and the orientation of the chromosomes are irrelevant for the sorting by translocations problem, we focus on the black cycles that are formed by the $n - N$ black points. The number of black cycles of P_A is maximized and equals $n - N$, if and only if genome A is sorted.

2.3. Effects of a translocation on elementary intervals and cycles

In the previous section, we have seen that we have to reduce the number of black breakpoints or increase the number of black cycles of P_A in order to sort a genome A by translocations. Thus, we are interested in how a translocation changes the number of breakpoints, as well as the number of cycles.

Lemma 1 (Kececioglu and Ravi [1995]). A translocation in genome A modifies the number of black cycles of P_A by 1, 0, or -1 .

Following the terminology of Hannenhalli (1996), a translocation is called *proper* if it increases the number of black cycles by 1, *improper* if it leaves the number of black cycles unchanged, and *bad* if it decreases the number of black cycles by 1. As a consequence of Lemma 1, we get the lower bound $d(A) \geq n - N - c$, where c is the number of black cycles of genome A .

An elementary interval whose endpoints belong to different chromosomes is called *interchromosomal*; otherwise, it is called *intrachromosomal*. Given an interchromosomal elementary interval I_k of P_A , we can always assume that elements k and $k + 1$ have different signs, since we can always flip a chromosome. This implies that the corresponding translocation creates a new adjacency: either $k \cdot k + 1$ or $-(k + 1) \cdot -k$. Hence we have the following.

Lemma 2. For each interchromosomal elementary interval in P_A , there exists a proper translocation in the genome A .

2.4. Intrachromosomal components

As discussed by Bergeron *et al.* (2004) for the inversion distance problem, elementary intervals and cycles can be grouped into higher structures:

Definition 4. A component of a signed permutation is an interval from i to $i + j$ or from $-(i + j)$ to $-i$, where $j > 0$, whose set of elements is $\{i, \dots, i + j\}$, and that is not the union of smaller such intervals.

We refer to a component by giving its first and last element such as $[i \dots j]$. When the elements of a component belong to the same chromosome, then the component is said to be *intrachromosomal*. An intrachromosomal component is called *minimal* if it does not contain any other intrachromosomal component. An intrachromosomal component that is an adjacency is called *trivial*, otherwise *nontrivial*.

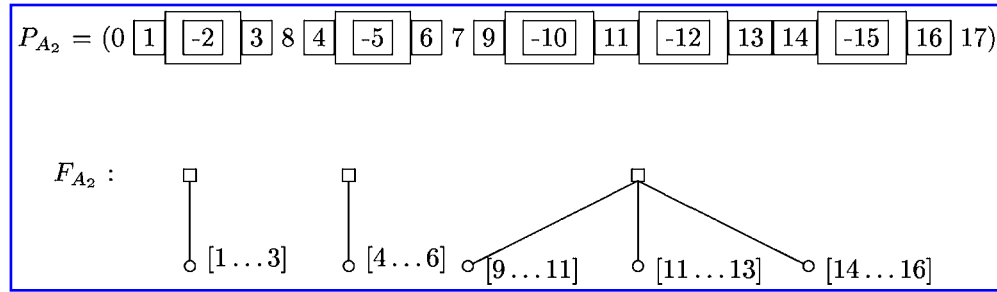


FIG. 3. The intrachromosomal components of the signed permutation P_{A_2} of the genome $A_2 = \{(1 \ -2 \ 3 \ 8 \ 4 \ -5 \ 6), (7 \ 9 \ -10 \ 11 \ -12 \ 13 \ 14 \ -15 \ 16)\}$ and the forest F_{A_2} .

For example, consider the genome

$$A_2 = \{(1 \ -2 \ 3 \ 8 \ 4 \ -5 \ 6), (7 \ 9 \ -10 \ 11 \ -12 \ 13 \ 14 \ -15 \ 16)\}.$$

The signed permutation P_{A_2} has the six intrachromosomal components $[1 \dots 3]$, $[4 \dots 6]$, $[9 \dots 11]$, $[11 \dots 13]$, $[13 \dots 14]$, and $[14 \dots 16]$; all of them are minimal, and all except $[13 \dots 14]$ are nontrivial. They can be represented by a boxed diagram such as in Fig. 3. Note that $[3 \dots 9]$ and $[6 \dots 7]$ are components that are not intrachromosomal.

The relationship between intrachromosomal components plays an important role in the sorting by translocations problem. As shown by Bergeron and Stoye (2003), two different intrachromosomal components of a chromosome are either disjoint, nested with different endpoints, or overlapping on one element.

When two intrachromosomal components overlap on one element, we say that they are *linked*. Successive linked intrachromosomal components form a *chain*. A chain that cannot be extended to the left or right is called *maximal*. We represent the nesting and linking relation of intrachromosomal components of a chromosome in the following way:

Definition 5. Given a chromosome X and its intrachromosomal components, define the forest F_X by the following construction:

1. Each nontrivial intrachromosomal component is represented by a round node.
2. Each maximal chain that contains nontrivial intrachromosomal components is represented by a square node whose (ordered) children are the round nodes that represent the nontrivial intrachromosomal components of this chain.
3. A square node is the child of the smallest intrachromosomal component that contains this chain.

We extend the above definition to a forest of a genome by combining the forests of all chromosomes:

Definition 6. Given a genome A consisting of chromosomes $\{X_1, X_2, \dots, X_N\}$, the forest F_A is the set of forests $\{F_{X_1}, F_{X_2}, \dots, F_{X_N}\}$.

Note that the forest F_A can consist of more than one tree in contrast to the unichromosomal case (Bergeron *et al.*, 2004). Figure 3 shows the forest F_{A_2} that consists of three trees.

2.5. Effects of a translocation on intrachromosomal components

We say that a translocation *destroys* an intrachromosomal component C if C is not an intrachromosomal component in the resulting genome. When a genome is sorted, eventually all its nontrivial intrachromosomal components, and hence all its trees, are destroyed.

The only way to destroy an intrachromosomal component with translocations is to apply a translocation with one endpoint in the component and one endpoint in another chromosome. Such translocations always

merge cycles and thus are always bad. Yet, a translocation may destroy more than one component at the same time. In fact, a translocation that acts on one point of an intrachromosomal component C destroys C and all the intrachromosomal components that contain C . Thus, at most two minimal intrachromosomal components on two different chromosomes, plus all intrachromosomal components containing these two components, can be destroyed by a single translocation.

It is also possible to eventually destroy by a single translocation two intrachromosomal components that initially belong to two different trees of the same chromosome. The next results show how.

Lemma 3. *If a chromosome X of genome A contains more than one tree, then there exists a proper translocation involving chromosome X .*

Proof. Consider the chromosome $X = (x_1 \dots x_m)$. We assume that all elementary intervals involving chromosome X are intrachromosomal. The first step is to show that then the whole chromosome is an intrachromosomal component. We have to show that the first element of the chromosome is the smallest element and the last element is the greatest, if both are positive, and the reverse, if both are negative, and that all elements between the smallest and the greatest are contained in the chromosome.

Let i be the smallest unsigned element contained in chromosome X . Suppose that i has positive sign and $x_1 \neq i$. The left point of i is an endpoint of the elementary interval I_{i-1} . Since i is the smallest element, the unsigned element $i - 1$ belongs to a chromosome different from X . Therefore the elementary interval I_{i-1} is interchromosomal. This contradicts our assumption that all elementary intervals involving the chromosome X are intrachromosomal.

Let j be the greatest unsigned element contained in chromosome X . Suppose that j has positive sign and $x_m \neq j$. Then the right point of j is an endpoint of the elementary interval I_j , and the element $j + 1$ belongs to another chromosome. Thus, the elementary interval I_j is interchromosomal contradicting our assumption.

By a similar argumentation, we can show that $x_1 = -j$, if j is the greatest element and has negative sign, and $x_m = -i$, if i the smallest element and has negative sign. Moreover, all elements between i and j have to be contained in chromosome X because otherwise there would be an interchromosomal elementary interval. Thus, chromosome X itself is an intrachromosomal component and contains a single tree. This leads to a contradiction. Therefore, there must exist an interchromosomal elementary interval with exactly one endpoint in X . By Lemma 2, the corresponding translocation is proper. ■

Hannenhalli showed that if there exists a proper translocation, then there exists a proper translocation that does not create any new minimal intrachromosomal components (see Theorem 10 of Hannenhalli [1996]). However, as we will see in Section 3, Hannenhalli's result is not sufficient to prove his claims, and leads to an incorrect algorithm. The following theorem states a stronger result, which is necessary to prove the distance formula and to develop sound algorithms.

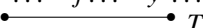
Theorem 1. *If a chromosome X of genome A contains more than one tree, and no other chromosome of A contains any nontrivial intrachromosomal component, then there exists a proper translocation involving chromosome X that does not modify F_A .*

Proof. A proper translocation can modify F_A either by linking two existing nontrivial intrachromosomal components or by creating new ones. In the first case, the two existing components must be in separate chromosomes, contrary to the hypothesis.

By Lemma 3, there exists at least one proper translocation involving chromosome X . Assume that they all create new nontrivial components and consider a proper translocation T that creates a component $[i \dots j]$ of minimal length, where $i < j - 1$. We will show that then there must exist another proper translocation that either creates smaller components or does not create nontrivial components.

Since T creates the component $[i \dots j]$, by flipping chromosomes as necessary, the signed permutation P_A can be written as

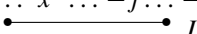
$$P_A = (\dots \ i \ \dots \ x \ \dots \ -j \ \dots \ -y \ \dots \)$$



where i and x are on the same chromosome and j and y on a different chromosome. Translocation T transforms P_A into $P_{A'}$:

$$P_{A'} = (\dots i \dots x y \dots j \dots).$$

Since $i < j - 1$, we have that $i \neq x$ or $j \neq y$ (or both). Suppose that $i \neq x$, then there exists an elementary interval J that has one endpoint between i and x and the other endpoint between x and j . Thus J is an interchromosomal elementary interval of P_A :

$$P_A = (\dots i \dots x \dots -j \dots -y \dots).$$


The diagram shows a horizontal line with dots at the ends, representing an interval J. The line is positioned below the permutation P_A, specifically between the elements x and -j. The label J is placed at the right end of the line.

By flipping chromosomes as necessary, we can assume that J is oriented. Applying the corresponding proper translocation to A yields

$$P_{A''} = (\dots i \dots j \dots -x \dots -y \dots),$$

or

$$P_{A''} = (\dots i \dots -y \dots -x \dots j \dots).$$

In both cases, i and x are on different chromosomes of A'' . A new nontrivial component that contains x does not contain i and thus must contain j in order to be longer than $[i \dots j]$.

A new nontrivial component cannot contain both i and j , since element $x \in \{i, \dots, j\}$ and x is on a different chromosome than i . If it contains i and is longer than $[i \dots j]$, then it must be an interval of $P_{A''}$ of the form $(i' \dots i \dots j')$, where $i' < i < j' < j$. But all the elements at the right of i are greater than i , and all the elements at the left of i are smaller than i , implying that either $i' = i$ or $i = j'$, which is a contradiction. Similar arguments hold if the new nontrivial component contains j and is longer than $[i \dots j]$. The case where $j \neq y$ can be treated similarly. ■

Efficient sorting by translocations will use the fact that trees belonging to different chromosomes can be easily dealt with. When all the trees are in one chromosome, we want to *separate* them; that means move them to different chromosomes. The next result states that such a separation is always possible with translocations that do not modify the topology of the forest.

Corollary 1. *If a chromosome X of genome A contains more than one tree, and no other chromosome of A contains any nontrivial intrachromosomal component, then the trees can be separated by proper translocations without modifying F_A .*

Proof. By Theorem 1, there exists a proper translocation that does not change F_A . Such a proper translocation either separates the trees or does not. If all the trees are still contained in the same chromosome, then, by the same argument, there exists another proper translocation that does not change the number of trees. Thus, there always exists either a separating or a nonseparating proper translocation. Since the number of successive proper translocations is finite, there always exists a sequence of proper translocations that separate the trees. ■

3. A DISCUSSION OF HANNENHALLI'S ALGORITHM

In order to compute the translocation distance, Hannenhalli (1996) introduced the notions of *subpermutations* and *even-isolation*. *Subpermutations* are equivalent to chains containing at least one nontrivial intrachromosomal component. A *minimal* subpermutation is a subpermutation that does not contain any other. A genome A has an *even-isolation* if all the minimal subpermutations of A reside on a single chromosome, the number of minimal subpermutations is even, and all the minimal subpermutations are

contained within a single subpermutation. Hannenhalli showed that

$$d(A) = n - N - c + s + o + 2i$$

where s denotes the number of minimal subpermutations, $o = 1$ if the number of minimal subpermutations is odd and $o = 0$ otherwise, and $i = 1$ if P has an even-isolation and $i = 0$ otherwise.

Based on the above equation, Hannenhalli gave a polynomial time algorithm for the sorting by translocations problem (Algorithm 1) where a translocation is called *valid* if it decreases the translocation distance.

Algorithm 1. Hannenhalli's algorithm (Hannenhalli, 1996)

```

1: while  $A$  is not identical to the target genome do
2:   if there is a proper translocation in  $A$  then
3:     select a valid proper translocation  $\rho$ 
4:   else
5:     select a valid bad translocation  $\rho$ 
6:   end if
7:    $A \leftarrow A\rho$ 
8: end while

```

The main assumption behind the algorithm is that if there exists a proper translocation, then there always exists a valid proper translocation (Theorem 12 of Hannenhalli [1996]). This is based on the argument that there exists a proper translocation that increases the number of cycles by 1 and does not change the number of minimal subpermutations. Hannenhalli wrongly concludes that such a proper translocation cannot create an even-isolation. The following genome shows that, apart from the obvious way to create an even-isolation by creating new subpermutations, there is a second way:

$$A_3 = \{(1 \quad 2 \quad 4 \quad 3 \quad 5 \quad \underline{12}), (11 \quad 6 \quad 8 \quad 7 \quad 9 \quad 10)\}.$$

Genome A_3 has exactly one proper translocation (underlined above), yielding

$$A'_3 = \{(1 \quad 2 \quad 4 \quad 3 \quad 5 \quad 6 \quad 8 \quad 7 \quad 9 \quad 10), (11 \quad 12)\}.$$

This translocation creates an even-isolation by chaining the two existing subpermutations [2...5] and [6...9]. Therefore, the translocation is not valid.

In order to prove the translocation formula, Hannenhalli first shows that if there exists a proper translocation, then there exists an alternative proper translocation that does not create new minimal subpermutations (Theorem 10 of Hannenhalli [1996]). Then Hannenhalli assumes that there is no proper translocation and follows by indicating how to destroy subpermutations (Theorem 13 of Hannenhalli [1996]). These results lead to an algorithm based on the false impression that the subpermutations can be destroyed independently of the sorting procedure.

Sometimes, in an optimal sorting scenario, we first have to destroy the subpermutations as is the case for genome A_3 . But in other cases, we first have to separate the subpermutations before destroying them. For example, consider the following genome:

$$A_4 = \{(-9 \quad 8 \quad -7 \quad 4 \quad -3 \quad 2 \quad -1), (10 \quad 6 \quad 5 \quad 11)\}.$$

In order to sort genome A_4 optimally, we first have to apply a proper translocation separating the subpermutations [-9...-7] and [-3...-1], yielding

$$A'_4 = \{(-9 \quad 8 \quad -7 \quad 4 \quad 5 \quad 11), (10 \quad 6 \quad -3 \quad 2 \quad -1)\}.$$

In the resulting genome A'_4 , the two subpermutations belong to different chromosomes so that we can destroy them by a single bad translocation.

However, in the next section, we will show that Hannenhalli's equation for the translocation distance holds, but that any sorting strategy should deal with destroying intrachromosomal components at each iteration step.

4. COMPUTING THE TRANSLOCATION DISTANCE

Given a genome A and the forest F_A , let L be the number of leaves and T the number of trees of the forest. The following lemma will be central in proving the distance formula and establishing an invariant for the sorting algorithm.

Lemma 4. *Let A be a genome whose forest has L leaves and T trees. If L is even and $T > 1$, then there always exists a sequence of proper translocations, followed by a bad translocation, such that the resulting genome A' has $L' = L - 2$ leaves and $T' \neq 1$ trees.*

Proof. If all the trees are on the same chromosome then, by Corollary 1, we can separate the forest with proper translocations without modifying T or L .

Assume that there exist trees on different chromosomes. In the following, we show how to pair two leaves such that the bad translocation destroying the corresponding intrachromosomal components reduces the number of leaves by two, and such that the number T' of trees in the resulting genome is either 0 or greater than 1.

Let t_1 be a tree with the largest number of leaves and t_2 be a tree with the largest number of leaves not on the same chromosome as t_1 . If t_1 has only one leaf, then all trees have only one leaf, including t_2 , and it is possible to destroy both t_1 and t_2 with a single bad translocation. In this case, $T = L$ and $T' = L'$; thus, $T' \neq 1$ since $L' = L - 2$ is even.

If t_1 has more than one leaf, choose its second leaf from the left and pair it with any leaf of t_2 . Performing the bad translocation that destroys those two leaves will create a new tree with the single leaf that was the leftmost leaf of t_1 . If $T' = 1$, then this single tree is the tree with one leaf; therefore, $L' = 1$, which is impossible since $L' = L - 2$ is even. ■

Lemma 4 implies that when the number of leaves is even and $T > 1$, we can always destroy the forest optimally: we can use proper translocations to separate the forest and then remove two leaves with a bad translocation. Eventually, all trees are destroyed, i.e., $T = 0$. The basic idea is to reduce all other cases to the simple case of Lemma 4.

Theorem 2. *Let A be a genome with c black cycles and F_A be the forest associated to A . Then*

$$d(A) = n - N - c + t$$

where

$$t = \begin{cases} L + 2 & \text{if } L \text{ is even and } T = 1 & (1) \\ L + 1 & \text{if } L \text{ is odd} & (2) \\ L & \text{if } L \text{ is even and } T \neq 1. & (3) \end{cases}$$

Proof. We first show that $d(A) \geq n - N - c + t$. Consider an optimal sorting of length d containing p proper translocations and b bad translocations; thus, $d = p + b$. Since b translocations remove b cycles and p translocations add p cycles, we must have

$$c - b + p = n - N, \text{ implying } d = n - N - c + 2b.$$

We will show that $2b \geq t$, implying $d \geq n - N - c + t$.

Since a bad translocation removes at most two leaves, we have that $b \geq L/2$, if L is even, and $b \geq (L + 1)/2$, if L is odd. Therefore, in cases (2) and (3), it follows that $b \geq t/2$.

If there is only one tree with an even number of leaves, then there must be a bad translocation B in the optimal sorting that has one endpoint in a tree and the other not contained in a tree. If this translocation does not destroy any leaves, then $b \geq 1 + L/2$. If translocation B destroys a minimal component, it destroys exactly one, and the minimal number of bad translocations needed to get rid of the remaining ones is $((L - 1) + 1)/2$, implying again that $b \geq 1 + L/2$. Thus, in case (1), we also have $b \geq t/2$.

In order to show that $d(A) \leq n - N - c + t$, we will exhibit a sequence of proper and bad translocations that achieve the bound $n - N - c + t$.

In case (2), if L is odd and $T = 1$, we destroy the middle leaf of the tree. Then $L - 1$ is even, and $T > 1$ or $T = 0$. If $T > 1$, then the preconditions of Lemma 4 apply, and the total number of bad translocations will be $1 + (L - 1)/2$.

If L is odd and $T > 1$, we destroy a single leaf of some tree with more than one leaf, if such a tree exists. Otherwise, we must have $T > 2$, since the number of leaves is odd, and we destroy any leaf. In both cases, we have $T' > 1$. Again, the total number of bad translocations will be $1 + (L - 1)/2$.

In case (3), if L is even and $T \neq 1$, then the preconditions of Lemma 4 apply, and the total number of bad translocations will be $L/2$.

In case (1), if L is even and $T = 1$, we destroy any leaf and apply case (2). The total number of bad translocations will be $1 + L/2$. ■

For example, the genome

$$A_2 = \{(1 \ -2 \ 3 \ 8 \ 4 \ -5 \ 6), (7 \ 9 \ -10 \ 11 \ -12 \ 13 \ 14 \ -15 \ 16)\}$$

of Section 2.4 consists of two chromosomes and 16 elements. The signed permutation P_{A_2} has seven black cycles. The forest F_{A_2} has three trees and five leaves (see Fig. 3). Therefore, we have

$$d(A_2) = n - N - c + t = 16 - 2 - 7 + 6 = 13.$$

5. ALGORITHMS

In this section, we present two algorithms. The first algorithm allows one to compute the translocation distance between two genomes in linear time. The second algorithm is the first correct polynomial time algorithm for sorting a genome by translocations.

5.1. Translocation Distance Algorithm

The algorithm to compute the translocation distance is similar to the one to compute the reversal distance presented by Bergeron *et al.* (2004). We only sketch the algorithm here and discuss those parts that need to be modified.

Assume that a genome A and an extended signed permutation P_A are given. The algorithm consists of three parts. In the first part, the cycles of P_A are computed by a left-to-right scan of P_A without taking into account the points between tails. The second part is the computation of the intrachromosomal components. We apply to each chromosome the linear-time algorithm of Bergeron *et al.* (2004) to compute intrachromosomal components. Finally, in the third part of the algorithm, the forest F_A is constructed by a single pass over the intrachromosomal components, and the distance can then easily be computed using the formula of Theorem 2.

Altogether, we can state the following theorem, previously announced by Li *et al.* (2004).

Theorem 3. *The translocation distance $d(A)$ of a genome A can be computed in linear time.*

5.2. Sorting by Translocations Algorithm

We now turn to the sorting by translocations problem. An algorithm that sorts a genome optimally is shown in Algorithm 2. Assume that the forest F_A of the genome A is given. We denote by L the number of leaves and by T the number of trees of the forest.

Initially, we apply up to two translocations in order to arrive at the preconditions of Lemma 4. If the forest consists of a single tree with an even number of leaves (line 2), we destroy any leaf. In the resulting genome, if the number of leaves is odd and in a single tree, we destroy its middle leaf; if there is more than one tree, we apply a translocation that destroys one leaf of the greatest tree. In all cases, we get a genome A' with $T' = 0$, or $T' > 1$ and L' even.

Then, as long as there exist intrachromosomal components (i.e., $T > 1$ and L is even), we can destroy the forest optimally as described in Lemma 4: we use proper translocations to separate the forest, and remove two leaves with each bad translocation. Once all intrachromosomal components are destroyed (i.e., $T = 0$),

Algorithm 2. Sorting by translocations algorithm

```

1:  $L$  is the number of leaves, and  $T$  the number of trees in the forest  $F_A$  associated to the genome  $A$ 
2: if  $L$  is even and  $T = 1$  then
3:   destroy one leaf such that  $L' = L - 1$ 
4: end if
5: if  $L$  is odd then
6:   perform a bad translocation such that  $T' = 0$ , or  $T' > 1$  and  $L' = L - 1$ 
7: end if
8: while  $A$  is not sorted do
9:   if there exist intrachromosomal components on different chromosomes then
10:    perform a bad translocation such that  $T' = 0$ , or  $T' > 1$  and  $L'$  is even
11:   else
12:    perform a proper translocation such that  $T$  and  $L$  remain unchanged
13:   end if
14: end while

```

we can sort the genome using proper translocations that do not create new nontrivial intrachromosomal components. Such proper translocations exist as we have shown in the proof of Theorem 1. Thus, there always exists either a proper translocation that does not modify the topology of the forest or a bad translocation that maintains the preconditions of Lemma 4. This establishes the correctness of the algorithm and we have the following theorem.

Theorem 4. *Algorithm 2 solves the sorting by translocations problem in $O(n^3)$ time.*

Initially, the forest F_A associated to a genome A is constructed. This can be done in $O(n)$ time as discussed above. The algorithm requires at most $O(n)$ iterations. The bad translocations of line 9 can be found in constant time as described in the proof of Lemma 4. Since there are $O(n)$ proper translocations and each translocation requires the construction of the forest to verify the condition $T' = T$ and $L' = L$, the search for a proper translocation in line 11 takes $O(n^2)$ time. Hence, the total time complexity of Algorithm 2 is $O(n^3)$.

6. CONCLUSION

The real challenge in developing genome rearrangement algorithms is to propose algorithms whose validity can be checked, both mathematically and biologically. The most useful set of rearrangement operations currently includes translocations, fusions, fissions, and inversions (Tesler, 2002). Unfortunately, we think that few people are able to assess the mathematical validity of the current algorithms. The work we have done in this paper opens the way to simpler description and implementation of such algorithms.

ACKNOWLEDGMENTS

We wish to thank Michal Ozery-Flato for a very careful reading and for sharing with us a simple and elegant proof of Lemma 4.

REFERENCES

- Bergeron, A., Mixtacki, J., and Stoye, J. 2004. Reversal distance without hurdles and fortresses. *Proc. 15th Ann. Symp. on Combinatorial Pattern Matching, CPM 2004, LNCS 3109*, 388–399.
- Bergeron, A., Mixtacki, J., and Stoye, J. 2005. The inversion distance problem, in Gascuel, O., ed., *Mathematics of Evolution and Phylogeny*, chap. 10, 262–290, Oxford University Press, Oxford, UK.
- Bergeron, A., and Stoye, J. 2003. On the similarity of sets of permutations and its applications to genome comparison. *Proc. 9th Ann. Int. Conf. on Computing and Combinatorics, COCOON 2003, LNCS 2697*, 68–79.

- Hannenhalli, S. 1996. Polynomial-time algorithm for computing translocation distance between genomes. *Disc. Appl. Math.* 71(1–3), 137–151.
- Hannenhalli, S., and Pevzner, P.A. 1995. Transforming men into mice (polynomial algorithm for genomic distance problem). *Proc. 36th Ann. Symp. on Foundation of Computer Science, FOCS 1995*, 581–592.
- Kececioğlu, J.D., and Ravi, R. 1995. Of mice and men: Algorithms for evolutionary distances between genomes with translocation. *Proc. 6th Ann. ACM-SIAM Symp. on Discrete Algorithms, SODA 1995*, 604–613.
- Li, G., Qi, X., Wang, X., and Zhu, B. 2004. A linear-time algorithm for computing translocation distance between signed genomes. *Proc. 15th Ann. Symp. on Combinatorial Pattern Matching, CPM 2004, LNCS 3109*, 323–332.
- Mouse Genome Sequencing Consortium. 2002. Initial sequencing and comparative analysis of the mouse genome. *Nature* 420, 520–562.
- Ozery-Flato, M., and Shamir, R. 2003. Two notes on genome rearrangements. *J. Bioinf. Comp. Biol.* 1(1), 71–94.
- Tesler, G. 2002. Efficient algorithms for multichromosomal genome rearrangements. *J. Comput. Syst. Sci.* 65(3), 587–609.

Address correspondence to:

Jens Stoye
Universität Bielefeld
Technische Fakultät
AG Genominformatik
D-33594 Bielefeld, Germany

E-mail: stoye@TechFak.Uni-Bielefeld.DE

This article has been cited by:

1. Michal Ozery-Flato , Ron Shamir . 2008. Sorting Genomes with Centromeres by Translocations. *Journal of Computational Biology* **15**:7, 793-812. [[Abstract](#)] [[PDF](#)] [[PDF Plus](#)]
2. Yun Cui, Lusheng Wang, Daming Zhu, Xiaowen Liu. 2008. $(1.5 + \sqrt{5})$ -Approximation Algorithm for Unsigned Translocation Distance. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* **5**:1, 56. [[CrossRef](#)]
3. Michal Ozery-Flato , Ron Shamir . 2007. Sorting by Reciprocal Translocations via Reversals Theory. *Journal of Computational Biology* **14**:4, 408-422. [[Abstract](#)] [[PDF](#)] [[PDF Plus](#)]