An $O(n^{3/2}\sqrt{\log(n)})$ algorithm for sorting by reciprocal translocations

Michal Ozery-Flato and Ron Shamir

School of Computer Science, Tel-Aviv University, Tel Aviv 69978, Israel
{ozery,rshamir}@post.tau.ac.il

Abstract. We prove that sorting by reciprocal translocations can be done in $O(n^{3/2}\sqrt{\log(n)})$ for an n-gene genome. Our algorithm is an adaptation of the Tannier et. al algorithm for sorting by reversals. This improves over the $O(n^3)$ algorithm for sorting by reciprocal translocations given by Bergeron et al.

1 Introduction

In this paper we study the problem of sorting by reciprocal translocations (abbreviated SRT). *Reciprocal translocations* exchange non-empty tails between two chromosomes. Given two multi-chromosomal genomes A and B, the problem of SRT is to find a shortest sequence of reciprocal translocations that transforms A into B. SRT that was first introduced by Kececioglu and Ravi [7] and was given a polynomial time algorithm by Hannenhalli [3]. Bergeron, Mixtacki and Stoye [2] pointed to an error in Hannenhalli's proof of the reciprocal translocation distance formula and consequently in Hannenhalli's algorithm. They presented a new $O(n^3)$ algorithm, which to the best of our knowledge, is the only extant correct algorithm for SRT[1].

Reversals (or inversions) reverse the order and the direction of transcription of the genes in a segment inside a chromosome. Given two uni-chromosomal genomes π_1 and π_2, the problem of sorting by reversals (abbreviated SBR) is to find a shortest sequence of reversals that transforms π_1 into π_2. Tannier, Bergeron and Sagot [9] presented an elegant algorithm for SBR that can be implemented in $O(n^{3/2}\sqrt{log(n)})$ using a clever data structure by Kaplan and Verbin [6]. This is currently the fastest algorithm for SBR.

In this paper we prove that SRT can be solved in $O(n^{3/2}\sqrt{\log(n)})$ for an n-gene genome. Our algorithm for SRT is similar to the algorithm by Tannier et al [9] for SBR. The paper is organized as follows. The necessary preliminaries are given in Sect. 2. In Sect. 3 we give a linear time reduction from SRT to a simpler restricted subproblem. In Sect. 4 we prove the main theorem and present the algorithm for the restricted subproblem. In Sect. 5 we describe an $O(n^{3/2}\sqrt{\log(n)})$

[1] Li et al. [8] gave a linear time algorithm for computing the reciprocal translocation distance (without producing a shortest sequence). Wang et al. [10] presented an $O(n^2)$ algorithm for SRT. However, the algorithms in [8, 10] rely on an erroneous theorem of Hannenhali and hence provide incorrect results in certain cases.

implementation of the algorithm. Due to space constraints, most proofs are omitted.

2 Preliminaries

This section provides a basic background for the analysis of SRT. It follows to a large extent the nomenclature and notation of [3, 5, 2]. In the model we consider, a *genome* is a set of chromosomes. A *chromosome* is a sequence of genes. A *gene* is identified by a positive integer. All genes in the genome are distinct. When it appears in a genome, a gene is assigned a sign of plus or minus. For example, the following genome consists of 8 genes in two chromosomes: $A = \{(1, -3, -2, 4, -7, 8), (6, 5)\}$. The *reverse* of a sequence of genes $I = (x_1, \ldots, x_l)$ is $-I = (-x_l, \ldots, -x_1)$. A *reversal* reverses a segment of genes inside a chromosome. Two chromosomes, X and Y, are *identical* if either $X = Y$ or $X = -Y$. Therefore, *flipping* chromosome X into $-X$ does not affect the chromosome it represents.

A *signed permutation* $\pi = (\pi_1, \ldots, \pi_n)$ is a permutation on the integers $\{1, \ldots, n\}$, where a sign of plus or minus is assigned to each number. If A is a genome with the set of genes $\{1, \ldots, n\}$ then any concatenation π_A of the chromosomes of A is a signed permutation of size n.

Let $X = (X_1, X_2)$ and $Y = (Y_1, Y_2)$ be two chromosomes, where X_1, X_2, Y_1, Y_2 are sequences of genes. A *translocation* cuts X into X_1 and X_2 and Y into Y_1 and Y_2 and exchanges segments between the chromosomes. It is called *reciprocal* if X_1,X_2, Y_1 and Y_2 are all non-empty. There are two ways to perform a translocation on X and Y. A *prefix-suffix* translocation switches X_1 with Y_2 resulting in: $(\underline{X_1}, X_2), (Y_1, \underline{Y_2}) \Rightarrow (-Y_2, X_2), (Y_1, -X_1)$. A *prefix-prefix* translocation switches X_1 with Y_1 resulting in: $(\underline{X_1}, X_2), (\underline{Y_1}, Y_2) \Rightarrow (Y_1, X_2), (X_1, Y_2)$. Note that we can mimic a prefix-prefix (respectively, prefix-suffix) translocation by a flip of one of the chromosomes followed by a prefix-suffix (respectively, prefix-prefix) translocation. As was demonstrated by Hannenhalli and Pevzner [4], a translocation on A can be simulated by a reversal on π_A in the following way: $(\ldots, X_1, \underline{X_2, \ldots, Y_1}, Y_2, \ldots) \Rightarrow (\ldots, X_1, \underline{-Y_1, \ldots, -X_2}, Y_2, \ldots)$. The type of translocation depends on the relative orientation of X and Y in π_A (and not on their order): if the orientation is the same, then the translocation is prefix-suffix, otherwise it is prefix-prefix. The segment between X_2 and Y_1 may contain additional chromosomes that are flipped and thus unaffected.

For a chromosome $X = (x_1, \ldots, x_k)$ define $Tails(X) = \{x_1, -x_k\}$. Note that flipping X does not change $Tails(X)$. For a genome A_1 define $Tails(A_1) = \bigcup_{X \in A_1} Tails(X)$. For example: $Tails(\{(1, -3, -2, 4, -7, 8), (6, 5)\}) = \{1, -8, 6, -5\}$. Two genomes A_1 and A_2 are *co-tailed* if $Tails(A_1) = Tails(A_2)$. In particular, two co-tailed genomes have the same number of chromosomes. Note that if A_2 was obtained from A_1 by performing a reciprocal translocation then $Tails(A_2) = Tails(A_1)$. Therefore, SRT is defined only for genomes that are co-tailed. For the rest of this paper the word "translocation" refers to a reciprocal translocation and we assume that the given genomes, A and B, are co-tailed.

2.1 The Cycle Graph

Let N be the number of chromosomes in A (equivalently, B). We shall always assume that both A and B contain genes $\{1, \ldots, n\}$. The *cycle graph* of A and B, denoted $G(A, B)$, is defined as follows. The set of vertices is $\bigcup_{i=1}^{n} \{i^0, i^1\}$. For every two genes, i and j, where j immediately follows i in some chromosome of A (respectively, B) add a black (respectively, grey) edge $(i, j) \equiv (out(i), in(j))$, where $out(i) = i^1$ if i has a positive sign in A (respectively, B) and otherwise $out(i) = i^0$, and $in(j) = j^0$ if j has a positive sign in A (respectively, B) and otherwise $in(j) = j^1$. There are $n - N$ black edges and $n - N$ grey edges in $G(A, B)$. A grey edge (i, j) is *external* if the genes i and j belong to different chromosomes of A, otherwise it is *internal*.

Every vertex in $G(A, B)$ has degree 2 or 0, where vertices of degree 0 (isolated vertices) belong to $Tails(A)$ (equivalently, $Tails(B)$). Therefore, $G(A, B)$ is uniquely decomposed into cycles with alternating grey and black edges. An *adjacency* is a cycle with two edges.

2.2 The Overlap Graph

Place the vertices of $G(A, B)$ along a straight line according to their order in π_A. Now, every grey edge can be associated with an interval of vertices of $G(A, B)$. Two grey edges *overlap* if the intersection of their intervals is not empty but none contains the other. The *overlap graph* of A and B w.r.t. π_A, denoted $OV(A, B, \pi_A)$, is defined as follows. The set of vertices is $\{(i_1, i_2) : (i_1, i_2)$ is a grey edge in $G(A, B)\}$. Two vertices are connected if their corresponding grey edges overlap. We shall use the word "component" for a connected component of the overlap graph. The set of components of $OV(A, B, \pi_A)$ can be computed in linear time using an algorithm by Bader, Moret and Yan [1].

A vertex in an overlap graph is *external* if its corresponding edge is external, otherwise it is *internal*. Note that the internal/external state of a vertex in $OV(A, B, \pi_A)$ does not depend on π_A (the partition of the chromosomes is known from A). A component of $OV(A, B, \pi_A)$ is *external* if at least one of the vertices in it is external, otherwise it is *internal*. A component is *trivial* if it corresponds to an adjacency and hence always internal. A vertex in the overlap graph is *oriented* if its corresponding edge connects two genes with different signs in π_A, otherwise it is *unoriented*.

The *span* of a component M is an interval of genes $I(M) = [i, j] \subset \pi_A$, where $i = arg\min\{\pi_A^{-1}(i_1), \pi_A^{-1}(i_2) \mid (i_1, i_2) \in M\}$ and $j = arg\max\{\pi_A^{-1}(j_1), \pi_A^{-1}(j_2) \mid (j_1, j_2) \in M\}$. Clearly, $I(M)$ is independent of π_A iff M is internal. Therefore, the set of internal components in $OV(A, B, \pi_A)$ is independent of π_A.

2.3 The Forest of Internal Components

$(M_1, \ldots, M_t)$ is a *chain* of components if $I(M_j)$ and $I(M_{j+1})$ overlap in exactly one gene for $j = 1, .., t - 1$. For a chain of components $C = (M_1, \ldots, M_t)$ define $I(C) = \bigcup_{j=1}^{t} I(M_j)$. The *forest of internal components*, denoted $F(A, B)$, is

defined as follows. The vertices of $F(A,B)$ are *(i)* the non-trivial internal components and *(ii)* every maximal chain of internal components that contains at least one non-trivial component. Let M and C be two vertices in $F(A,B)$ where M corresponds to a component and C to a chain. $M \to C$ is an edge of $F(A,B)$ if $M \in C$. $C \to M$ is an edge of $F(A,B)$ if $I(C) \subset I(M)$ and $I(M)$ is minimal. We will refer to a component that is a leaf in $F(A,B)$ as simply a *leaf*.

2.4 The Reciprocal Translocation Distance

Let $c(A,B)$ denote the number of cycles in $G(A,B)$. Let $T(A,B)$ and $L(A,B)$ denote the number of trees and leaves in $F(A,B)$ respectively. Obviously $T(A,B) \leq L(A,B)$. Define $f_{\mathrm{r}}(A,B) = \begin{cases} 2 & \text{if } T(A,B)=1 \text{ and } L(A,B) \text{ is even} \\ 1 & \text{if } L(A,B) \text{ is odd} \\ 0 & \text{otherwise } (T(A,B) \neq 1 \text{ and } L(A,B) \text{ is even}) \end{cases}$

Theorem 1 *[2, 3] The reciprocal translocation distance between A and B is $d_{\mathrm{r}}(A,B) = n - N - c(A,B) + L(A,B) + f_{\mathrm{r}}(A,B)$*

Let Δc denote the change in the number of cycles after performing a translocation on A. Then $\Delta c \in \{-1,0,1\}$ [3]. A translocation is *proper* if $\Delta c = 1$ and *bad* if $\Delta c = -1$. A translocation ρ is *valid* if $d_{\mathrm{r}}(A \cdot \rho, B) = d_{\mathrm{r}}(A,B) - 1$. A translocation is *safe* if it does not create any new non-trivial internal component. As was demonstrated by Bergeron et al. [2] a safe translocation might be invalid if the set of leaves is not empty. However, if there are no leaves, then a safe proper translocation is necessarily valid. We define SRTNL as a special case of SRT when there are no leaves (i.e. $T(A,B) = L(A,B) = 0$).

3 A Linear Reduction of SRT to SRTNL

A translocation is bad iff it cuts two black edges, b_1 and b_2, that belong to different cycles [3]. Note that there are two bad translocations, either prefix-prefix or suffix-prefix, cutting the black edges b_1 and b_2. A leaf M is *eliminated* by performing a (bad) translocation that cuts one black edge incident to a grey edge in M and one black edge in another chromosome of A. Observe that in this case all the ancestors of M in $F(A,B)$ are eliminated as well. Let $L(X)$ denote the number of leaves in chromosome X. Let $N^{\mathrm{L}}(A,B)$ denote the number of chromosomes of A containing at least one leaf. A translocation ρ is *separating* if $N^{\mathrm{L}}(A,B) = 1$ but $N^{\mathrm{L}}(A \cdot \rho, B) > 1$. It is easy to see that a translocation is separating only if it cuts a black edge between two leaves.

Lemma 1 *[2] There is a sequence of safe proper translocations that sorts all external components (i.e., after performing the sequence, every edge in an external component becomes an adjacency).*

Lemma 2 *[2] Let $S = (\rho_1, \ldots, \rho_k)$ be a sequence of safe proper translocations that sorts all external components. If $N^{\mathrm{L}}(A,B) = 1$ but $T(A,B) > 1$ then S contains a separating translocation ρ_l. Moreover, $S' = \rho_1, \ldots, \rho_l$ is a sequence of valid translocations and $N^{\mathrm{L}}(A \cdot \rho_1 \cdots \rho_l, B) > 1$.*

Lemma 3 *[3] Suppose that the following conditions are satisfied: (i)* $N^{\mathrm{L}}(A,B) = 1$*, (ii)* $L(A,B) \geq 2$*, and (iii) either* $L(A,B)$ *is odd or* $T(A,B) = 1$*. Let* ρ *be a (prefix-prefix) translocation that eliminates the second leaf from the left in* A*. Then* ρ *is valid and if* $L(A \cdot \rho, B) \geq 2$ *then* $N^{\mathrm{L}}(A \cdot \rho, B) \geq 2$*.*

Lemma 4 *All the bad translocations in the algorithm in Fig. 1 are valid.*

(1) **if** $N^{\mathrm{L}} = 1$ **and** $L \geq 2$:
  (a) **if** $T > 1$ **and** L is even:
    (i) Solve SRTNL on the set of external components **until** $N^{\mathrm{L}} \neq 1$.
  (b) **else**: eliminate the second leaf from the left by a prefix-prefix translocation.
(2) Let Q_1 be a queue of the chromosomes containing exactly one leaf.
  Let Q_2 be a queue of the chromosomes containing more than one leaf.
(3) **while** $L > 0$ *(Invariant: L=1* **or** $N^{\mathrm{L}} \geq 2$*)*
  (a) **if** $L = 1$: eliminate the single leaf by a prefix-prefix translocation.
  (b) **else**:
    (i) For $i = 1, 2$
      1. **if** $Q_2 \neq \emptyset$ then $X_i \leftarrow pop(Q_2)$, otherwise $X_i \leftarrow pop(Q_1)$.
      2. **if** $L(X_i) = 2$ then $l_i \leftarrow$ the second leaf from the left in X_i, otherwise $l_i \leftarrow$ the single leaf in X_i.
    (ii) Eliminate l_1 and l_2 by a prefix-prefix translocation.
    (iii) For $i = 1, 2$: **if** $L(X_i) > 1$ then $push(X_i, Q_2)$. **if** $L(X_i) = 1$ then $push(X_i, Q_1)$.
(4) Solve SRTNL on A.

Fig. 1. A generic algorithm for solving SRT using an algorithm for SRTNL.

The generic algorithm in Fig. 1 and the preceding lemmas imply:

Theorem 2 *SRT is linearly reducible to SRTNL.*

4 An Algorithm for SRTNL

In this section we present an algorithm for SRTNL. We first define an extension of the overlap graph and then prove the algorithm's correctness. Fig. 3 provides examples of the graphs used.

4.1 The Overlap Graph with Chromosomes

A chromosome X and an edge e *overlap* if X contains exactly one of the two endpoints of e. Hence, if edge e overlaps chromosome X of A then e must be an external grey edge. We define the *overlap graph with chromosomes*, $OVCH(A, B, \pi_A)$ based on $OV(A, B, \pi_A)$ as follows. We add to $OV(A, B, \pi_A)$ a vertex for each chromosome of A. In order to prevent confusion, we will refer to the new vertices as "chromosomes" and reserve the word "vertex" for the original vertices of $OV(A, B, \pi_A)$ (that correspond to edges). A vertex and a chromosome are connected if the corresponding grey edge overlaps the chromosome. There are no edges between chromosomes.

Let $H = OVCH(A, B, \pi_A)$ and let v be any vertex in H. Denote by $N(v) \equiv N(v, H)$ the set of vertices that are neighbors of v, including v itself (but not including chromosome neighbors). Denote by $CH(v) \equiv CH(v, H)$ the set of chromosomes that are neighbors of v in H. Hence if v is external then $|CH(v)| = 2$, otherwise $CH(v) = \emptyset$.

Every external grey edge e defines one proper translocation that cuts the black edges incident to e. (Out of the two possibilities of prefix-prefix or prefix-suffix translocations, exactly one would be proper). For an external vertex v denote by $\rho(v)$ the proper translocation that the corresponding grey edge defines on A. Two external vertices v_1 and v_2 in H are *equivalent* if they define the same translocation, i.e. $\rho(v_1) \equiv \rho(v_2)$. Let $H \cdot \rho(v) = OVCH(A \cdot \rho(v), B, \pi_A)$. Given two sets S_1 and S_2 define $S_1 \bigoplus S_2 = (S_1 \bigcup S_2) \setminus (S_1 \bigcap S_2)$.

Lemma 5 *Let v be an oriented external vertex in H. Then $H \cdot \rho(v)$ is obtained from H by the following operations. (i) Complement the subgraph induced by $N(v)$ and flip the orientation of every vertex in $N(v)$. (ii) For every vertex $u \in N(v)$ such that the endpoints of u and v share at least one common chromosome, update the edges between u and $CH(u) \bigcup CH(v)$ such that $CH(u) = CH(u) \bigoplus CH(v)$.*

Two overlap graphs with chromosomes are *equivalent* if one can be obtained from the other by a sequence of chromosome flips. For a chromosome X let $\rho(X)$ denote a flip of chromosome X in π_A. Let $H \cdot \rho(X) = OVCH(A, B, \pi_A \cdot \rho(X))$.

Lemma 6 *$H \cdot \rho(X)$ is obtained from H by complementing the subgraph induced by the set $\{u : X \in CH(u)\}$ and flipping the orientation of every vertex in it.*

4.2 The Main Theorem and Algorithm

In this section we give the main theorem and algorithm. Our algorithm is formally very similar to the algorithm for SBR presented in [9]. Instead of performing reversals on oriented edges in [9], we perform translocations on external edges. Despite of the great similarity between the algorithms our validity proof is completely new. We analyze an overlap graph with chromosomes of a multi-chromosomal genome, while [9] analyze the overlap graph of a uni-chromosomal genome. Like [9], we perform operations defined by oriented vertices (i.e. translocations). However, in our case these vertices must also be external. If an external vertex is unoriented, we can turn it into an oriented vertex by a flip of a chromosome. Hence, we consider two types of operations in our analysis.

A sequence of vertices $S = (v_1, \ldots, v_k)$ from H is *legal* if v_j is external in $H \cdot \rho(v_1) \cdots \rho(v_{j-1})$ for $j = 1, .., k$. For a legal sequence S define $\rho(S) = \rho(v_1) \cdots \rho(v_k)$. A legal sequence S is *total* if $H \cdot \rho(S)$ contains only trivial components. For H_1, an overlap graph with chromosomes, let $IN(H_1)$ and $EXT(H_1)$ denote the sets of vertices that are in non-trivial internal components and external components respectively. If S is a maximal legal sequence of vertices in H then $EXT(H \cdot \rho(S)) = \emptyset$. If in addition S is not total then $IN(H \cdot \rho(S)) \neq \emptyset$.

Theorem 3 *Let $S = (v_1, \ldots, v_k)$ be a maximal legal but not total sequence of vertices in H. Let $IN = IN(H \cdot \rho(S))$. Let v_l be the first vertex in S satisfying $IN(H \cdot \rho(v_1, \ldots, v_l)) = IN$, i.e. $\rho(v_l)$ is the last unsafe translocation in $\rho(S)$. Let $S_1 = (v_1, \ldots, v_{l-1})$ and $S_2 = (v_l, \ldots, v_k)$. Then every maximal sequence of vertices $S' = (w_1, \ldots, w_m)$ in IN that satisfies (i) (S_1, S') is legal and (ii) v_l is not an adjacency in $H \cdot \rho(S_1, S')$ also satisfies: (iii) S' is not empty and (iv) (S_1, S', S_2) is a maximal legal sequence. Moreover, all the translocations in $\rho(S_2)$ are safe.*

Proof. Let $v = v_l$, $H_0 = H \cdot \rho(S_1)$ and $IN_0 = EXT(H_0) \cap IN$. Then $IN_0 \neq \emptyset$ and none of the vertices in IN_0 is equivalent to v in H_0 (otherwise it would be an adjacency in $H \cdot \rho(S)$ and hence not in IN). Hence S' is not empty. Let $A_0 = A \cdot \rho(S_1)$ and $CH(v) = \{X, Y\}$. We choose π_0 to be a concatenation of the chromosomes in A_0 in which X and Y are the first two chromosomes. We can assume w.l.o.g. that $H = OVCH(A, B, \pi_0)$, hence $H_0 = OVCH(A_0, B, \pi_0)$. For $j = 1, .., m$ let $H_j = H_0 \cdot \rho(w_1, \ldots, w_j)$. Let $IN_j = EXT(H_j) \bigcap IN$. Then for $j = 1, \ldots, m$: *(i)* $w_j \in IN_{j-1}$ and *(ii)* w_j is not equivalent to v in H_{j-1}. Let $EXT = EXT(H_0 \cdot \rho(v))$. The following conditions hold for H_j when $j = 0$ (see Fig. 4-(a)):

(1) The subgraphs of $H_j \cdot \rho(v)$ and $H_0 \cdot \rho(v)$ that are induced by EXT are equivalent.
(2) Every $w \in IN_j$ satisfies: $CH(w) = CH(v) = \{X, Y\}$.
(3) If v is oriented then $N(v) \bigcap IN = IN_j$.
(4) All the possible edges exist between $N(v) \bigcap EXT$ and IN_j.
(5) There are no edges between $IN \setminus IN_j$ and vertices outside IN.
(6) There are no edges between $EXT \setminus N(v)$ and vertices outside EXT.

We shall prove below that in H_m v is external and that all the above conditions are satisfied. The first condition ensures that (S_1, S', S_2) is legal. The rest of the conditions ensure that $H_m \cdot \rho(v)$ satisfies: *(i)* there are no external vertices in IN and *(ii)*there are no edges between EXT and vertices outside EXT. Hence (S_1, S', S_2) is maximal and every translocation in $\rho(v_{l+1}, \ldots, v_k)$ is safe. $\rho(v_l)$ is safe in H_m since S' is maximal. Therefore, all the translocations in $\rho(S_2)$ are safe.

Assume that v is external in H_j and that the all above conditions hold for a certain j. Since these conditions are true for every graph that is equivalent to H_j we can assume that v is oriented. We now prove, using an induction on j, that these conditions are satisfied for every H_i, $i \in \{1, \ldots, m\}$ in which v is external, and that v is external in H_m.

Case 1: w_{j+1} is oriented in H_j. Let $H_{j+1} = H_j \cdot \rho(w_{j+1})$ (see Fig. 4-(b)). Then $IN_{j+1} = N(v, H_j) \bigoplus N(w_{j+1}, H_j)$. $IN_{j+1} \neq \emptyset$, otherwise v is an isolated internal vertex in H_{j+1} and hence equivalent to w_{j+1} in H_j. Hence $m \geq j + 2$.

Case 1.*a*: w_{j+2} is oriented in H_{j+1}. Let $H_{j+2} = H_{j+1} \cdot \rho(w_{j+2})$ (see Fig. 4-(c)). Clearly, v is external in H_{j+2}. Let $M = N(v, H_j) \bigcap EXT$. Then $N(w_{j+2}, H_{j+1}) \bigcap EXT = N(w_{j+1}, H_j) \bigcap EXT = M$. Hence the subgraphs of H_{j+2} and H_j that are induced by M are identical and the first condition is satisfied in H_{j+2}.

Case 1.*b*: w_{j+2} is unoriented in H_{j+1}. Let $H'_{j+1} = H_{j+1} \cdot \rho(X)$ (H'_{j+1} and H_{j+1} are equivalent) (see Fig. 4-(d)). Hence w_{j+2} is oriented in H'_{j+1}. Note that v is an internal vertex in H'_j. Let $M' = N(w_{j+1}, H'_{j+1}) \bigcap EXT$. Let $H_{j+2} = H'_{j+1} \cdot \rho(w_{j+2})$ (see Fig. 4-(e)). v is an oriented external vertex in H_{j+2} and $N(v, H_{j+2}) \bigcap EXT = M'$. Therefore, the two subgraphs of $H_{j+2} \cdot \rho(v)$ (see Fig. 4-(f)) and H'_{j+1} (see Fig. 4-(d)) that are induced by EXT are identical. The subgraphs of H_{j+1} and $H_j \cdot \rho(v)$ that are induced by EXT are also identical. Hence, the first condition is satisfied.

Looking at Figs. 4-(c) and 4-(e) it is easy to verify that the rest of the conditions are also satisfied for H_{j+2}.

Case 2: w_{j+1} is unoriented in H_j. We define the three subsets of vertices $M_1, M_2, M_3 \subset EXT$ in H_j as follows:

(1) M_1 is the set of neighbors of w_{j+1} (equivalently, v) that are either internal or external but does not overlap chromosome X.
(2) M_2 is the set of neighbors of w_{j+1} (equivalently, v) that overlap chromosome X. Hence $M_1 \bigcup M_2 = N(v, H_j) \bigcap EXT$.
(3) M_3 is the set of vertices that overlap chromosome X but are not neighbors of w_{j+1} (equivalently, v).

For an illustration of H_j see Fig. 4-(g). Let $H'_j = H_j \cdot \rho(X)$ (see Fig. 4-(h)). In H'_j: w_{j+1} is an oriented external vertex and is not a neighbor of v. Let $H_{j+1} = H'_j \cdot \rho(w_{j+1})$ (see Fig. 4-(i)). Obviously, v remains intact in H_{j+1}. Let $H'_{j+1} = H_{j+1} \cdot \rho(X)$ (see Fig. 4-(j)). Then, the subgraphs of $H'_{j+1} \cdot \rho(v)$ (see Fig. 4-(k)) and $H_j \cdot \rho(v)$ that are induced by M_1, M_2 and M_3 are equivalent (Compare the subgraph induced by EXT in H_j in Fig. 4 (g) with the subgraph induced by EXT in $H'_{j+1} \cdot \rho(v) \cdot \rho(X)$ in Fig. 4 (l)). Hence the first condition is satisfied. Looking at Fig. 4-(i), it is easy to verify that conditions (2)-(6) hold for H_{j+1}. □

The algorithm in Fig. 2 builds a sequence of translocations by a repeated application of Theorem 3. It greedily removes external edges from an allowed subset and performs the corresponding translocations (step (2).(a)). When the allowed subset contains only internal edges, the algorithm repeats the last translocations in a reverse order (thereby cancelling them) until another edge in the allowed subset becomes external (step (2).(b)). Every translocation in the algorithm is applied at most twice and so the algorithm performs at most $2n$ translocations.

5 An $O(n^{3/2}\sqrt{\log(n)})$ Time Implementation of the Algorithm

The algorithm in Fig. 2 can be implemented in $O(n^2)$ time in a relatively simple manner. We provide below an $O(n^{3/2}\sqrt{\log(n)})$ algorithm. The implementation follows closely the ideas of [6] and [9].

Assume w.l.o.g. that π_B is the identity permutation. Then every grey edge is of the form $(i, i+1)$. We identify a grey edge $(i, i+1)$ by i and refer to $(i+1)$ as the *remote end* of i. The data structure we use for maintaining the genome A is as follows.

(1) Let V be the set of edges in $G(A, B)$ that are in non-trivial components.
Set $S_1 = S_2 = \emptyset$
(2) **while** $V \neq \emptyset$:
  (a) **while** there exists an external edge $v \in V$ in $G(A, B)$:
    (i) Remove v from V.
    (ii) **if** v is not equivalent to the first element in S_2:
      1. Append $\rho(v)$ to S_1
      2. $A \leftarrow A \cdot \rho(v)$
  (b) **while** all the edges in V are internal:
    (i) Let ρ be the last translocation in S_1
    (ii) Remove ρ from S_1
    (iii) Prepend ρ to S_2
    (iv) $A \leftarrow A \cdot \rho$
(3) return (S_1, S_2)

Fig. 2. An algorithm for SRTNL.

1. A doubly linked list of $O(\sqrt{\frac{n}{\log(n)}})$ blocks. We partition π_A into continuous blocks such that the size of every block is at least $\frac{1}{2}\sqrt{n\log(n)}$ and at most $2\sqrt{n\log(n)}$.
2. A balanced search tree for every block. The tree contains the edges in the block ordered by the positions of their remote ends. We use balanced trees that support split and concatenate operations in logarithmic time, such as red-black trees or 2-4 trees. We use $T[v]$ to denote the subtree rooted at v and containing all its descendants.
3. An n-array of block pointers. The i^{th} entry in the array points to the block containing i.

We add the following fields to the above data structure.

1. For each edge we keep an external-bit. If the external-bit is *on* then the edge is external, otherwise it is internal.
2. For each block we keep the following fields: *(i)* a counter of external edges in V, *(ii)* a counter of chromosomes' left tails, and *(iii)* a reverse-flag. If the reverse-flag of a block is *on* then the order and signs of the elements in the block are reversed.
3. For every subtree $T[v]$ of each block's search tree we keep the following fields in its root v: *(i)* counters of external and internal edges in V, *(ii)* a direction-flip-flag and *(iii)* an external-flip-flag. If the external-flip-flag of a vertex v is *on* then in $T[v]$ the external-bits of all the elements are flipped and the counters of internal and external elements from V exchange their values. If the direction-flip-flag of a vertex v is *on* then in $T[v]$ the order of the elements is reversed.

We can clear the direction-flip-flag of a node by reversing the order of its children and flipping the direction-flip-flag in each of them. We can clear the external-flip-flag in a node by exchanging the values of the counters of external and internal

edges in V, flipping the external-flip-flag in each of its children and flipping the external-bit of the element residing at the node. One can view this procedure as "pushing down" the flags. An direction-flip-flag and an external-flip-flag that are *on* are "pushed down" whenever $T[v]$ is searched.

We implement the algorithm using the above data structures. A search for an external edge in V is done as follows. We traverse the list of blocks until we reach a block that contains external edges from V. We then search the tree of the block for an external edge i. We locate element $i+1$ (the remote end of edge i) using the n-array and a search of its block.

Let ρ be a translocation on A operating on the chromosomes $X = (X_1, X_2)$ and $Y = (Y_1, Y_2)$. Then ρ is performed in $O(\sqrt{n \log(n)})$ time as follows:

(1) Split at most six blocks so that each of the four segments X_1, X_2, Y_1 and Y_2 corresponds to a union of blocks. If ρ is a prefix-prefix translocation exchange the blocks of X_1 and Y_1. Otherwise, reverse the order and flip the reverse-flags of the blocks of X_2 and Y_1 and then exchange the blocks of X_2 and Y_1.
(2) We now have to modify the trees of each block to reflect the order and direction changes. This is done as follows. Traverse all the blocks and for each block:
   (a) Let T be the balanced search tree of the block. If ρ is a translocation on an edge i in V and i is contained in the block: decrease by 1 the counters of external edges in V of the block and of every node in T that contains i in its subtree.
   (b) Split T into at most seven subtrees such that each of the segments X_1, X_2, Y_1 and Y_2 has a corresponding subtree.
   (c) If the block corresponds to a segment of X_1, X_2, Y_1 and Y_2 flip the external-flip-flag at the roots of two subtrees according to Table 1.
   (d) If ρ is a prefix-prefix translocation, exchange the subtrees of X_1 and Y_1. Otherwise, exchange the subtrees of X_2 and Y_1 and flip the direction-flip-flags of both.
   (e) Concatenate the seven subtrees into T.
(3) If necessary, concatenate small blocks and split large blocks such that the size of each block is at least $\frac{1}{2}\sqrt{n \log(n)}$ and at most $2\sqrt{n \log(n)}$.

Theorem 4 *SRTNL can be solved in* $O(n^{3/2}\sqrt{\log(n)})$. □

References

1. D.A. Bader, B. M.E. Moret, and M. Yan. A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. *Journal of Computational Biology*, 8(5):483–491, 2001.
2. A. Bergeron, J. Mixtacki, and J. Stoye. On sorting by translocations. *Journal of Computational Biology*, 13(2):567–578, 2006.
3. S. Hannenhalli. Polynomial algorithm for computing translocation distance between genomes. *Discrete Applied Mathematics*, 71:137–151, 1996.

4. S. Hannenhalli and P. Pevzner. Transforming men into mice (polynomial algorithm for genomic distance problems). In *Proc. FOCS'95*, pages 581–592, Los Alamitos, 1995. IEEE Computer Society Press.
5. H. Kaplan, R. Shamir, and R. E. Tarjan. Faster and simpler algorithm for sorting signed permutations by reversals. *SIAM Journal of Computing*, 29(3):880–892, 2000.
6. H. Kaplan and E. Verbin. Sorting signed permutations by reversals, revisited. *Journal of Computer and System Sciences*, 70(3):321–341, 2005.
7. J. D. Kececioglu and R. Ravi. Of mice and men: Algorithms for evolutionary distances between genomes with translocation. In *Proceedings of the 6th Annual Symposium on Discrete Algorithms*, pages 604–613, New York, NY, USA, January 1995. ACM Press.
8. G. Li, X. Qi, X. Wang, and B. Zhu. A linear-time algorithm for computing translocation distance between signed genomes. In *Proc. CPM*, pages 323–332, 2004.
9. E. Tannier, A. Bergeron, and M. Sagot. Advances on sorting by reversals. *to appear in Discrete Applied Mathematics*.
10. L. Wang, D. Zhu, X. Liu, and S. Ma. An O(n^2) algorithm for signed translocation problem. In *Proc. APBC*, pages 349–358, 2005.

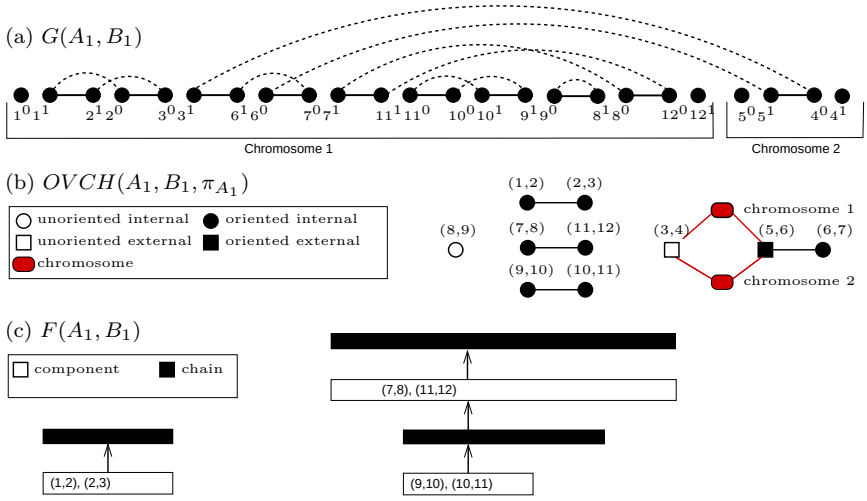

Fig. 3. Auxiliary graphs for $A_1 = \{(1, -2, 3, -6, 7, -11, 10, -9, -8, 12), (5, 4)\}$, $B_1 = \{(1, \ldots, 4), (5, \ldots, 12)\}$ $(\pi_{A_1} = (1, -2, 3, -6, 7, -11, 10, -9, -8, 12, 5, 4))$.

Table 1. The subtrees for which the external-flip-flag is flipped as a function of translocation type and block type.

Block	X_1	X_2	Y_1	Y_2
prefix-prefix	X_2, Y_2	X_1, Y_1	X_2, Y_2	X_1, Y_1
prefix-suffix	X_2, Y_1	X_1, Y_2	X_1, Y_2	X_2, Y_1

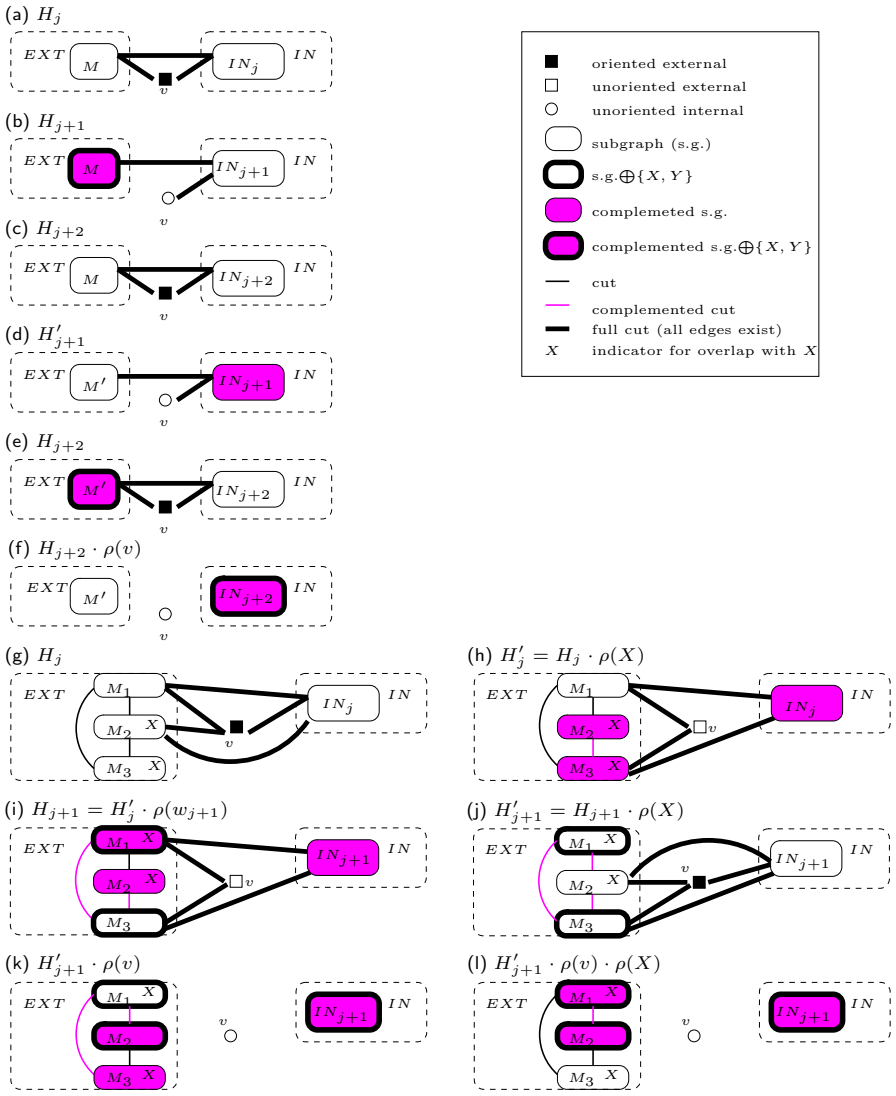

Fig. 4. Illustrations for the proof of Theorem 3.