BMC Bioinformatics

Research

Genome rearrangements with duplications

Martin Bader

Address: Institute of Theoretical Computer Science, Ulm University, 89069 Ulm, Germany

E-mail: martin.bader@uni-ulm.de

from The Eighth Asia Pacific Bioinformatics Conference (APBC 2010)
Bangalore, India 18-21 January 2010

Published: 18 January 2010

BMC Bioinformatics 2010, **11**(Suppl 1):S27 doi: 10.1186/1471-2105-11-S1-S27

This article is available from: http://www.biomedcentral.com/1471-2105/11/S1/S27

Abstract

Background: Finding sequences of evolutionary operations that transform one genome into another is a classical problem in comparative genomics. While most of the genome rearrangement algorithms assume that there is exactly one copy of each gene in both genomes, this does not reflect the biological reality very well - most of the studied genomes contain duplicated gene content, which has to be removed before applying those algorithms. However, dealing with unequal gene content is a very challenging task, and only few algorithms allow operations like duplications and deletions, especially if the duplicated or deleted segments are of arbitrary size.

Results: We recently proposed a heuristic algorithm for sorting unichromosomal genomes by reversals, block interchanges, tandem duplications, and deletions. In this paper, we extend this approach to multichromosomal genomes. We are now able to sort a multichromosomal ancestral genome into a genome of a descendant by a large set of different operations, including tandem duplications and deletions of segments of arbitrary size.

Conclusion: Experimental results show that our algorithm finds sorting sequences that have a weight close to the true evolutionary distance.

Background

During evolution, genomes are subject to genome rearrangements, which are large scale mutations that can alter the ordering and orientation (strandedness) of the genes on the chromosomes or even change the genome content by inserting, deleting, or duplicating genes. Because these events are rare compared to point mutations, they can give us valuable information about ancient events in the evolutionary history of organisms. For this reason, one is interested in the most "plausible" genome rearrangement scenario between two genomes. More precisely, given two genomes, one wants to find an optimal sequence of rearrangements that transforms this genome into the other. In the classical approach, each gene has exactly one copy in each genome, and only operations that do not change the genome content are considered. A breakthrough in the research of these "classical operations" was Hannenhalli and Pevzner's algorithm for *sorting by reversals* [1], and due to further research, we are now able to sort multichromosomal

genomes by *reversals, translocations, fusions, and fissions* in polynomial time [2,3]. If one also considers *transpositions*, the problem gets more complicated, and there are only approximation algorithms known [4-6]. To simplify the existing algorithms, Yancopoulos et al. invented the *double cut and join* operator (DCJ), which can simulate reversals, translocations, fusions, fissions, and block interchanges (a more generalized form of transpositions), resulting in a simple and efficient algorithm [7].

However, restricting the genes to be unique in each genome does not reflect the biological reality very well, as in most genomes that have been studied, there are some genes that are present in two or more copies. This holds especially for the genomes of plants, and one of the most prominent genomes is the one of the flowering plant *Arabidopsis thaliana*, where large segments of the genome have been duplicated (see e.g. [8]). There are various evolutionary events that can change the content of the genome, like duplications of single genes, horizontal gene transfer, or tandem duplications. For a nice overview in the context of comparative genomics, see [9]. In a pioneering work, Sankoff tackled the challenge of genomes with duplicated genes with his "exemplar model" [10], where the following problem was examined. Given two genomes with duplicated genes, identify in both genomes the "true exemplars" of each gene and remove all other genes, such that the rearrangement distance between these modified genomes is minimized. This approach was later extended to the "matching model", where one searches for a maximum matching between the copies of each gene such that the genome rearrangement distance according to this matching is minimized [11]. However, both approaches have been proven to be NP-hard for the breakpoint distance and the reversal distance [11-13]. Note that these approaches do not construct the evolutionary events that changed the genome contents, i.e. the set of operations is still restricted to the classical set of operations. The first approach that explicitly constructed duplication events was done by El-Mabrouk [14], where one searches for a hypothetical ancestor with unique gene content, such that the reversal and duplication distance from this ancestor to a given descendant (with duplicated genes) is minimized. This work has been further extended during the last years (see e.g. [13,15]). Still, the duplications were technically limited to have the length of one element, and therefore the algorithms can only be applied if no large segmental duplication happened during evolution. One idea to overcome this problem was to simulate duplications by insertions, as it has been done in [16-18]. Recently, Yancopoulos and Friedberg provided a mathematical model (but without algorithm) of a genome rearrangement distance for genomes with unequal gene content [19], combining the DCJ operator with arbitrary but length-weighted insertions and deletions. To the best of our knowledge, the first work that allowed duplications of arbitrary segments was done by Ozery-Flato and Shamir [20], who demonstrated that a simple greedy algorithm can find biologically realistic sorting scenarios for most karyotypes in the *Mitelman Database of Chromosome Abberations in Cancer* [21]. Further simplifications of the model led to an algorithm with provable approximation ratio of 3 [22] (note that the algorithm performs much better in practice). Recently, we proposed a heuristic algorithm for sorting a unichromosomal genome by DCJs, tandem duplications, and deletions of arbitrary segments [23]. In this work, we extend this approach to multichromosomal genomes. We are now able to sort an ancestral multichromosomal genome by a large set of operations, including duplications and deletions of arbitrary size. As a constraint, two chromosomes in the ancestral genome must be either disjoint or identical. Although this restriction seems to be very limiting, this is often fulfilled in practice. A possible application is to examine the evolution of a cancer karyotype out of a diploid set of healthy chromosomes.

Methods

Preliminaries

A *chromosome* $\pi^i = (\pi^i_1 \dots \pi^i_l)$ is a string over the alphabet $\Sigma = \{1, ..., n\}$, where each element may have a positive or negative orientation (indicated by $\vec{x}$ or $\overleftarrow{x}$). We get the *inverse* of an element π^i_j (indicated by $-\pi^i_j$) by inverting its orientation. The *reflection* of a chromosome $(\pi^i_1 \dots \pi^i_l)$ is the chromosome $(-\pi^i_l \dots -\pi^i_1)$. It is considered to be equivalent to π^i. A *genome* $\pi = \{\pi^i, ..., \pi^m\}$ is a multiset of chromosomes. The *multiplicity* $mult(\pi, x)$ of an element x is the number of its occurrences (with arbitrary orientation) in π. A *segment* is a consecutive sequence of elements of a chromosome. Each element x can also be represented by the ordered set of its *extremities* x_t(the *tail*) and x_h(the *head*), where the ordering of the extremities is $x_t\ x_h$ if x has positive orientation, and $x_h\ x_t$ otherwise. For example, the chromosome $(\vec{1}\ \vec{2})$ can also be written as $(1_t 1_h 2_t 2_h)$. The two extremities belonging to the same element are called *co-elements*. We say the tail/head $x_{t/h}$ of an element x is a *telomere* if there is a chromosome in π beginning or ending with $x_{t/h}$. The value $t(\pi, x_t)$ determines how often x_t is a telomere in π, the value $t(\pi, x_h)$ is defined analogously. Two consecutive extremities in a chromosome π^i which are no co-elements form an *inner adjacency* w.r.t. another genome ρ if they are also consecutive in a chromosome of ρ, otherwise they form an *inner breakpoint*. An extremity which is a telomere in π forms a *telomere adjacency* w.r.t. another genome ρ if it is also a telomere in ρ, otherwise it forms a

telomere breakpoint. For example, if $\rho = \{(1_t 1_h 2_t 2_h 3_t 3_h)\}$ and $\pi = \{(1_t 1_h 2_t 2_h), (1_t 1_h 3_t 3_h)\}$, then 1_t and 3_h form telomere adjacencies, while 2_h forms a telomere breakpoint. Applying an *operation op* to a genome π yields the genome $op(\pi)$. A *genome rearrangement problem* is defined as follows. Given two genomes ρ and π and a set of possible operations, where each operation is assigned a weight, find a sequence of minimum weight (i.e. the sum of the weights of the operations is minimized) that transforms ρ into π. This minimum weight will be denoted by $d(\rho, \pi)$. In our algorithm, we will consider all operations listed in Subsection "Operations". For simplification, we will assume that two chromosomes in ρ are either disjoint (i.e. they have no element in common) or identical. Furthermore, each element in Σ must appear in at least one chromosome of ρ. Note that these restrictions only hold for the genome ρ, not for π.

Operations

The following set of operations will be considered by our algorithm. A *reversal* inverts the order of the elements of a segment. Additionally, the orientation of each element in the segment is inverted. A *transposition* cuts a segment out of a chromosome, and reinserts it at another position in the same chromosome. If we additionally apply a reversal on this segment, we speak of an *inverted transposition*. A *fusion* concatenates two chromosomes. Both chromosomes π^i and π^j can be inverted before the operation, i.e. we can replace π^i or π^j by its reflection. A *fission* splits a chromosome into two chromosomes. A *translocation* splits two chromosomes π^i and π^j into $\pi^{i,t}$, $\pi^{i,h}$ and $\pi^{j,t}$, $\pi^{j,h}$, and then concatenates $\pi^{i,t}$ with $\pi^{j,h}$ and $\pi^{j,t}$ with $\pi^{i,h}$. Again, both chromosomes can be inverted before the operation. A *tandem duplication* inserts an identical copy of a segment immediately after this segment in a chromosome. A *transposition duplication* inserts an identical copy of a segment at an arbitrary position in the genome. A *deletion* cuts a segment out of a chromosome. A *chromosome duplication* creates an identical copy of a chromosome. A *chromosome deletion* deletes a chromosome.

All operations have weight 1, except for (inverted) transpositions and transposition duplications, which have weight 2. These weights are chosen rather by mathematical than biological reasons, but still result in biologically realistic scenarios (see also Section "Discussion"). To simplify the analysis of the effects of the operations in Section "A lower bound", we will there use the *double cut and join operator* (DCJ), which has been introduced in [7]. A DCJ cuts a genome at two positions, and rejoins the cut ends in two new pairs. It is a well-known fact that reversals, translocations, fusions, and fission can be described by one DCJ operation, while transpositions can be described by two DCJ operations. Duplications and deletions cannot be described by DCJ operations and therefore must be examined separately.

The breakpoint graph

Our main tool for developing the algorithm is the breakpoint graph, which is an edge-colored multigraph that visualizes the current adjacencies and breakpoints. It has been introduced by Bafna and Pevzner to solve rearrangement problems on genomes without duplicated genes [24]. We extend this graph such that it can also be used for genomes with duplicated genes. The breakpoint graph of two genomes ρ and π can be constructed as follows. First, we write the set of vertices $\{1_t, 1_h, 2_t, 2_h, \ldots, n_t, n_h\}$ from left to right on a straight line. Second, for each pair $(x_{t/h}, y_{t/h})$ of extremities that are no co-elements but consecutive in a chromosome of ρ, we connect the corresponding vertices by a *gray edge*. Third, we analogously add a *black edge* for each pair of extremities that are consecutive in π. However, if one of the endpoints is not an endpoint of a gray edge (i.e. it corresponds to a telomere in ρ), we do not add the black edge (this is required to obtain a good lower bound in Section "A lower bound"). For an example, see Fig. 1. In contrast to the original breakpoint graph, each vertex can be the endpoint of several gray and black edges. The *multiplicity* of an edge (v, v') is the number of black edges between v and v'. A black edge (v, v) is called a *loop*. Let $L(\rho, \pi)$ denote the number of vertices with a loop. Two vertices v, v' are in the same *component* of the graph if and only if there is a path (consisting of gray and black edges) from v to v'. Let $C(\rho, \pi)$ denote the number of components in the breakpoint graph of π. A black edge is called a *1-bridge* if the removal of this edge increases $C(\rho, \pi)$. A pair of black edges is called a *2-bridge* if none of the edges is a 1-bridge and the removal of both edges increases $C(\rho, \pi)$.

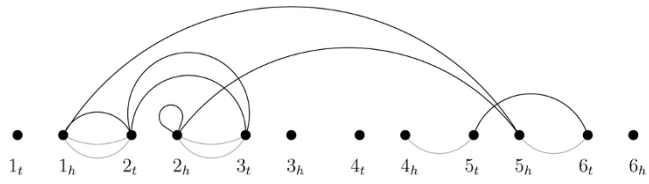

Figure 1
The breakpoint graph. The breakpoint graph of $\rho = \{(\vec{1}\ \vec{2}\ \vec{3}), (\vec{1}\ \vec{2}\ \vec{3}), (\vec{4}\ \vec{5}\ \vec{6})\}$ and $\pi = \{(\vec{1}\ \vec{2}\ \vec{2}\ \vec{3}), (\vec{4}\ \vec{3}\ \vec{2}\ \vec{5}\ \vec{6}), (\vec{5}\ \vec{1})\}$. Note that there is no black edge $(4_h, 3_h)$, as 3_h is a telomere in ρ. The edge $(2_t, 3_t)$ has a multiplicity of 2, all other black edges have a multiplicity of 1. The edge $(2_h, 2_h)$ is a loop. The breakpoint graph consists of 5 components, the black edge $(5_t, 6_t)$ is a 1-bridge, the pair of black edges $(1_h, 5_h)$, $(2_h, 5_h)$ is a 2-bridge.

A lower bound

Instead of searching for a sequence of operations op_1, ..., op_kthat sorts ρ into π, one can also search for the inverse sequence $op_k^{-1}, \ldots, op_1^{-1}$ that sorts π into ρ (we will call such a sequence a *sorting sequence*). This is more convenient, because then the breakpoint graph has some nice properties due to the limitations in ρ. Note that simply restricting π instead of ρ would not work, because the operations are directed from the restricted ancestral genome to the unrestricted descendant, i.e. we would nevertheless have to invert the operations. In the following, we will examine what effects the inverse operations have on the breakpoint graph. In the unichromosomal case, we were mainly interested in the effects on components and loops (see [23]). As the breakpoint graph does not contain edges for telomeres, we additionally have to consider the amount of *incorrect telomeres* (denoted by $T(\rho, \pi)$), which is defined as follows.

$$\begin{aligned} T(\rho,\pi) = &\sum_{x_t \mid t(\rho,x_t)>0} \max\{t(\rho,x_t) - t(\pi,x_t), 0\} \\ &+ \sum_{x_h \mid t(\rho,x_h)>0} \max\{t(\rho,x_h) - t(\pi,x_h), 0\} \\ &+ \sum_{x_t \mid t(\rho,x_t)>0} t(\pi,x_t) + \sum_{x_h \mid t(\rho,x_h)>0} t(\pi,x_h) \end{aligned}$$

In other words, for each telomere x_t(or x_h) in ρ, we count how often we must create this telomere in π such that $t(\rho, x_t) \leq t(\pi, x_t)$ (or $t(\rho, x_h) \leq t(\pi, x_h)$). Additionally, we add the amount of telomeres x_t and x_h in π that have to be removed, i.e. they are not telomeres in ρ.

Lemma 1. *If $\pi = \rho$, then $C(\rho, \pi)$ is maximized, and $L(\rho, \pi)$ and $T(\rho, \pi)$ are minimized.*

Proof. If $\pi = \rho$, the set of gray edges and the set of black edges in the breakpoint graph are equal. Thus, removing black edges does not increase the number of components, and adding black edges can only decrease the number of components. Therefore, changing π cannot increase $C(\rho, \pi)$. $L(\rho, \pi)$ and $T(, \pi)$ are both positive numbers, and if $\pi = \rho$, they are equal to 0 and therefore minimized. □

In [23], we have shown that all operations can either change the number of components by 1, or change the number of loops by 2. These observations still hold for our slightly modified breakpoint graph. We will now examine how an operation can change $T(\rho, \pi)$.

Inverse reversal, translocation, transposition

These operations can be simulated by one or two inverse DCJs (which is equivalent to a normal DCJ), thus it is sufficient to examine the effects of a DCJ (note that transpositions, which require two DCJs, have weight 2). A DCJ can only change $T(\rho, \pi)$ if one of its edges is the end of a chromosome. Then, a telomere $x_{t/h}$ is removed and a new telomere $y_{t/h}$ is created. This decreases $T(\rho, \pi)$ by 1 or 2 if $x_{t/h}$ is not a telomere in π and $y_{t/h}$ is a telomere in π, otherwise the operation does not decrease $T(\rho, \pi)$. However, in the first case, the DCJ did not cut any black edge, as we neither draw black edges for telomeres in ρ nor for telomeres in π.

Inverse fusion (fission)

Splitting a chromosome will only decrease $T(\rho, \pi)$ if both new telomeres are also telomeres in ρ. In this case, no black edge in the breakpoint graph is removed, i.e. $C(\rho, \pi)$ and $L(\rho, \pi)$ remain unchanged. $T(\rho, \pi)$ is decreased by at most 2.

Inverse fission (fusion)

Concatenating two chromosomes can decrease $T(\rho, \pi)$ by at most 2. This operation never removes a black edge, thus $C(\rho, \pi)$ cannot be increased and $L(\rho, \pi)$ cannot be decreased.

Inverse tandem duplication

This operation does never change the set of telomeres in π, and therefore cannot change $T(\rho, \pi)$.

Inverse transposition duplication

This operation can decrease $T(\rho, \pi)$ only if the duplicated segment is at a chromosome end, and the new chromosome end (after deleting the segment) is a telomere in ρ. In this case, no black edge with multiplicity 1 is removed, therefore $C(\rho, \pi)$ and $L(\rho, \pi)$ remain unchanged. The decrement of $T(\rho, \pi)$ is ≤ 2.

Inverse deletion (insertion)

This operation can only change $T(\rho, \pi)$ if we insert a segment at a chromosome end. In this case, no black edge is removed, i.e. $C(\rho, \pi)$ cannot be increased and $L(\rho, \pi)$ cannot be decreased. $T(\rho, \pi)$ is decreased by at most 2.

Inverse chromosome duplication

This operation can decrease $T(\rho, \pi)$ by at most 2 (if the telomeres of the removed chromosome are not telomeres in ρ). Only black edges with multiplicity ≥ 2 are removed, thus $C(\rho, \pi)$ and $L(\rho, \pi)$ remain unchanged.

Inverse chromosome deletion (chromosome insertion)

This operation can decrease $T(\rho, \pi)$ by at most 2 (if the telomeres of the new chromosome are also telomeres in ρ). In the breakpoint graph, no black edges are removed,

i.e. $C(\rho, \pi)$ cannot be increased and $L(\rho, \pi)$ cannot be decreased.

Theorem 1. *A lower bound* $lb(\rho, \pi)$ *of the distance* $d(\rho, \pi)$ *is*

$$d(\rho,\pi) \geq lb(\rho,\pi) = \left\lceil \frac{T(\rho,\pi)}{2} \right\rceil + c(\rho) - C(\rho,\pi) + \sum_{Components C_i} \left\lceil \frac{L_i(\rho,\pi)}{2} \right\rceil$$

where $c(\rho) = n$ + (*number of different chromosomes in* ρ), *and* $L_i(\rho, \pi)$ *is the number of vertices with a loop in component* C_i *in the breakpoint graph of* ρ *and* π.

Proof. An operation that decreases $T(\rho, \pi)$ will neither increase $C(\rho, \pi)$ nor decrease $L(\rho, \pi)$, therefore we can separate every sequence into operations that decrease $T(\rho, \pi)$ and operations that decrease $CL(\rho,\pi) = c(\rho) - C(\rho,\pi) + \sum_{Components C_i} \left\lceil \frac{L_i(\rho,\pi)}{2} \right\rceil$. Each operation decreases $T(\rho, \pi)$ by at most 2, so we need at least $\frac{T(\rho,\pi)}{2}$ operations of the first kind. Furthermore, if $\rho = \pi$, then $C(\rho, \pi) = c(\rho)$ and therefore $CL(\rho, \pi) = 0$. As each operation decreases $CL(\rho, \pi)$ by at most one (the proof in [23] still holds), we need at least $CL(\rho, \pi)$ operations of the second kind. Therefore, any sorting sequence must have at least $lb(\rho, \pi)$ operations. □

Corollary 1. $lb(\rho, \rho) = 0$.

Unfortunately, there are genomes $\pi \neq \rho$ with $lb(\rho, \pi) = 0$, i.e. it is not sufficient to sort until the lower bound reaches 0. We therefore have to introduce another distance measure $\tau(\rho, \pi)$. We will use the following definitions.

$$m(\rho,\pi) = \sum_{\text{elements } x} \left| mult(\rho,x) - mult(\pi,x) \right|$$

$$ia(\rho,\pi) = 2 \cdot \#\text{ inner adjacencies in } \pi \text{ w.r.t } \rho$$

$$ta(\rho,\pi) = \#\text{ telomere adjacencies in } \pi \text{ w.r.t } \rho$$

$$\tau(\rho,\pi) = 4 \cdot m(\rho,\pi) + ia(\rho,\rho) + ta(\rho,\rho) - ia(\rho,\pi) - ta(\rho,\pi)$$

For example, if $\rho = \{(\vec{1}\ \vec{2}\ \vec{3}\ \vec{4})\}$ and $\pi = \{(\vec{1}\ \vec{2})(\vec{3}\ \vec{4})\}$, then $lb(\rho, \pi) = 0$ and $\tau(\rho, \pi) = 2$.

Lemma 2. *If* $\rho = \pi$, *then* $\tau(\rho, \pi) = 0$. *Otherwise*, $\tau(\rho, \pi) > 0$.

Proof. It is clear that $\tau(\rho, \pi) = 0$ if $\rho = \pi$. In order to minimize $\tau(\rho, \pi)$, it is necessary to minimize $m(\rho, \pi)$ and to maximize $ia(\rho, \pi)$ and $ta(\rho, \pi)$. $ia(\rho, \rho)$ and $ta(\rho, \rho)$ are independent of π and therefore fixed. Each inner adjacency is weighted by 2. We can interpret this as if each of the adjacent extremities is weighted by 1. Therefore, we can say that each element in π can account at most 2 to $ia(\rho, \pi) + ta(\rho, \pi)$, and this value is reached if there is an adjacency on both sides of the element. Thus, the contribution to $\tau(\rho, \pi)$ of all occurrences of an element x in π is minimized if $mult(\rho, x) = mult(\pi, x)$ and no extremity of x is part of any breakpoint. Every additional occurrence of x may increase $ia(\rho, \pi) + ta(\rho, \pi)$ by 2, but also increases $m(\rho, \pi)$ by 4 and therefore increases $\tau(\rho, \pi)$ by at least 2. This means that $\tau(\rho, \pi)$ is minimized if each element has the same multiplicity in ρ and π, and the breakpoint graph contains no breakpoints. This is the case if and only if ρ and π are identical. □

The algorithm

The algorithm uses a greedy strategy to sort π into ρ by inverse operations. For better readability, we will simply write "operation" instead of "inverse operation" in this section. First, we define $\Delta lb(op) = lb(\rho, \pi) - lb(\rho, op(\pi))$ and $\Delta\tau(op) = \tau(\rho, \pi) - \tau(\rho, op(\pi))$. The *score* σ of an operation op is defined as the tuple $\sigma(op) = (\Delta lb(op), \Delta\tau(op))$. The comparison operator between two scores is defined by $\sigma(op_1) > \sigma(op_2)$ if $\Delta lb(op_1) > \Delta lb(op_2) \vee \Delta(lb(op_1) = \Delta lb(op_2) \wedge \Delta\tau(op_1) > \Delta\tau(op_2))$. In each step, we search for operations that decrease the lower bound, and apply the one with the greatest score. If no such operation exists, we use additional heuristics to find operations that do not change the lower bound but have a positive score (i.e. $\sigma(op) > (0, 0)$). There is still the possibility that we even do not find such an operation. In this case, we use a fallback algorithm that is guaranteed to terminate.

Operations that decrease the lower bound

Finding operations that increase $C(\rho, \pi)$ can be done by finding 1-bridges and 2-bridges in the breakpoint graph and verifying additional preconditions, as shown in [23]. The only difference is that now a DCJ can cut only one black edge. This is the case when the other cutting point contains a telomere in ρ or π. Thus, we also have to consider DCJs that act on a 1-bridge and a telomere. Such a DCJ can be interpreted as inverse reversal, translocation, fission, or transposition. In the last case, we have to find a third cutting point in the same chromosome such that the resulting inverse transposition still increases $C(\rho, \pi)$. Also finding operations that decrease $L(\rho, \pi)$ is straightforward and can be done as in [23]. The remaining task is to find operations that decrease $T(\rho, \pi)$. For this, we create a list of telomeres in π that are not telomeres in ρ, and another list of inner breakpoints in π where at least one of the adjacent elements is a telomere in ρ. Operations that decrease $T(\rho, \pi)$ must act on one or two points of these lists, depending on the operation type. Creating the lists can be done by a linear scan over π, therefore all operations that decrease $T(\rho, \pi)$ can be found in quadratic time. The

only exceptions are inverse deletions and inverse chromosome deletions, which may add segments of arbitrary content. Practical tests have shown that it is better to only allow the insertion of segments without any breakpoints. This does not only lead to better sorting sequences, but also keeps the time complexity of finding the operations in $O(n^2)$.

Additional operations

If there is no operation that decreases $lb(\rho, \pi)$, we may still find operations that do not change the lower bound but decrease $\tau(\rho, \pi)$. Searching for all these operations would exceed our computing capacity, so we just search for the following subset of these operations that can be found easily.

- There are inverse tandem duplications and transposition duplications that do not change $\sigma(\rho, \pi)$, but decrement $\tau(\rho, \pi)$. We therefore search for identical consecutive segments that are maximal, i.e. they cannot be extended in any direction, and check the effect on $\sigma(\rho, \pi)$ and $\tau(\rho, \pi)$ if we remove one of them. This operation corresponds either to an inverse tandem duplication, or to an inverse transposition duplication.
- Depending on the telomeres of a chromosome, the lower bound can remain unchanged during an inverse chromosome duplication, but $\tau(\rho, \pi)$ can decrease. We therefore search for identical chromosomes and check the score of removing one of them.
- Inserting a segment of consecutive elements x with $mult(\rho, x) > mult(\pi, x)$ decreases $\tau(\rho, \pi)$. We search for such segments of maximal length and try to insert them by an inverse deletion. Note that this is not always possible as this operation can increase the lower bound by merging two components.
- Creating inner or telomere adjacencies never increases the lower bound, but decreases $\tau(\rho, \pi)$. We therefore search for DCJs and inverse fissions that create new adjacencies without splitting old ones.

The fallback algorithm

It is possible that there is neither an operation that decreases $lb(\rho, \pi)$, nor an operation that decreases $\tau(\rho, \pi)$, so the main algorithm gets stuck. However, this case cannot occur if all elements have the same multiplicity in ρ and in π.

Lemma 3. *If $\rho \neq \pi$ and $mult(\rho, x) = mult(\pi, x)$ holds for all elements x, then there is an operation with positive score.*

Proof. When the preconditions are fulfilled, there must be at least one breakpoint in π. We have to distinguish three cases. (1) This is a telomere breakpoint. W.l.o.g. a chromosome in π ends with x_h, but x_h is not a telomere in ρ. Then, $mult(\rho, x) = mult(\rho, x + 1)$ (as they are in the same chromosome), and therefore there must be another breakpoint including $(x + 1)_t$. An operation that creates an adjacency between x_h and $(x + 1)_t$ will not decrease the lower bound, but decrease $\tau(\rho, \pi)$ by at least 2. (2) The breakpoint is an inner breakpoint between two extremities that are telomeres in ρ. In this case, the score of cutting the chromosome at this breakpoint is (1, 2), because both extremities become telomeres (i.e. $T(\rho, \pi)$ increases by 2), and we create two telomere adjacencies. (3) The breakpoint is an inner breakpoint, and at least one of the adjacent extremities is not a telomere in ρ. W.l.o.g., the breakpoint is of the form (x_h, y_h), and x_h is not a telomere in ρ. Then, $mult(\rho, x) = mult(\rho, x + 1)$, thus there must be another breakpoint including $(x + 1)_t$. An operation that creates an adjacency between x_h and $(x + 1)_t$ will not increase the lower bound, but decrease $\tau(\rho, \pi)$ by at least 2. □

The fallback algorithm will first ensure that the precondition of the lemma holds. For each chromosome ρ^i in ρ, we determine the element x with the most occurrences in π. We then create maximal segments of consecutive elements $\vec{\gamma}\,\overrightarrow{\gamma+1}$ ... such that each element z in the segment belongs to ρ^i and $mult(\pi, z) < mult(\pi, x)$, and add this segment by an inverse deletion to π. Note that this can be done without creating new breakpoints. This step is repeated until all elements belonging to the same component in ρ have the same multiplicity in π. We then transform ρ into ρ' by applying chromosome duplications and chromosome deletions on ρ such that for each element x, $mult(\rho', x) = mult(\pi, x)$. Now, we apply our normal algorithm to sort π into ρ'. To ensure that the precondition of Lemma 3 is always fulfilled, we forbid operations that create or delete elements, i.e. any kind of duplication or deletion. Due to Lemma 3, the algorithm will find a sorting sequence that transforms π into ρ'. As last step, we have to undo the operations that transformed ρ into ρ'.

Results

We tested our algorithm on artificial data and on cancer karyotypes from the "Mitelman Database of Chromosome Aberrations in Cancer" [21].

Artificial data

We used simulated data to assess the performance of our algorithm. First, we created genomes ρ with n different elements and c different chromosomes. Each chromosome has the same size, the ploidy (i.e. the number of identical copies) of the chromosomes is 1 or 2. Then, we generated the genome π by applying random sequences of operations of weight $w = \alpha n$ on ρ (with α varying from

0.1 to 1.0 in steps of 0.1). The operations of the sequences are independently distributed, with all operations having the same probability. Although these probabilities do not match the biological reality, this is still convenient to assess the performance of the algorithm. Once the type of an operation was determined, the operation was drawn from a uniform distribution of all operations of this type. The genomes ρ and π were now used as input to our algorithm. The parameters n and c were chosen such that they reflect the properties of biologically meaningful datasets. To understand what "biologically meaningful" means, let us have a brief look on biological datasets. In most of them, elements do not represent single genes but *synteny blocks*, i.e. regions of a chromosome that are highly conserved and do not contain breakpoints. These synteny blocks normally contain several genes. The amount n of different synteny blocks depends on the allowed dissimilarity between the synteny blocks as well as on the evolutionary distance between the genomes. For example, El-Mabrouk et al. [25] tested their algorithm on yeast genomes with 55 synteny blocks, Zheng et al. [26] identified 34 synteny blocks between rice, sorghum, and maize. Salse et al. [27] used 60 synteny blocks to compare *Arabidopsis thaliana* and rice. A recent comprehensive study of *Drosophila* genomes [28] identified between 112 and 1406 synteny blocks, depending on the evolutionary distance of the species. Our datasets reflect those parameters. Dataset 1 contains 16 chromosomes of ploidy 2 with a total of 64 elements, this approximately matches the yeast genome. Dataset 2 contains 12 chromosomes of ploidy 2 with a total of 36 elements, Dataset 3 contains 5 chromosomes of ploidy 2 with a total of 60 elements. These are realistic values for plant genomes. Dataset 4 contains 5 different chromosomes with a total of 200 elements, two of them with ploidy 1 and three of them with ploidy 2 (corresponding to 2 sex chromosomes and 3 diploid chromosomes). This reflects the values for closely related *Drosophila* species. Each dataset contains 100 different test cases for each generated distance w. Together with the use of different distances w, this allows us to get a much more robust result than just testing on a few biological datasets. The results of our experiments are depicted in Fig. 2. The

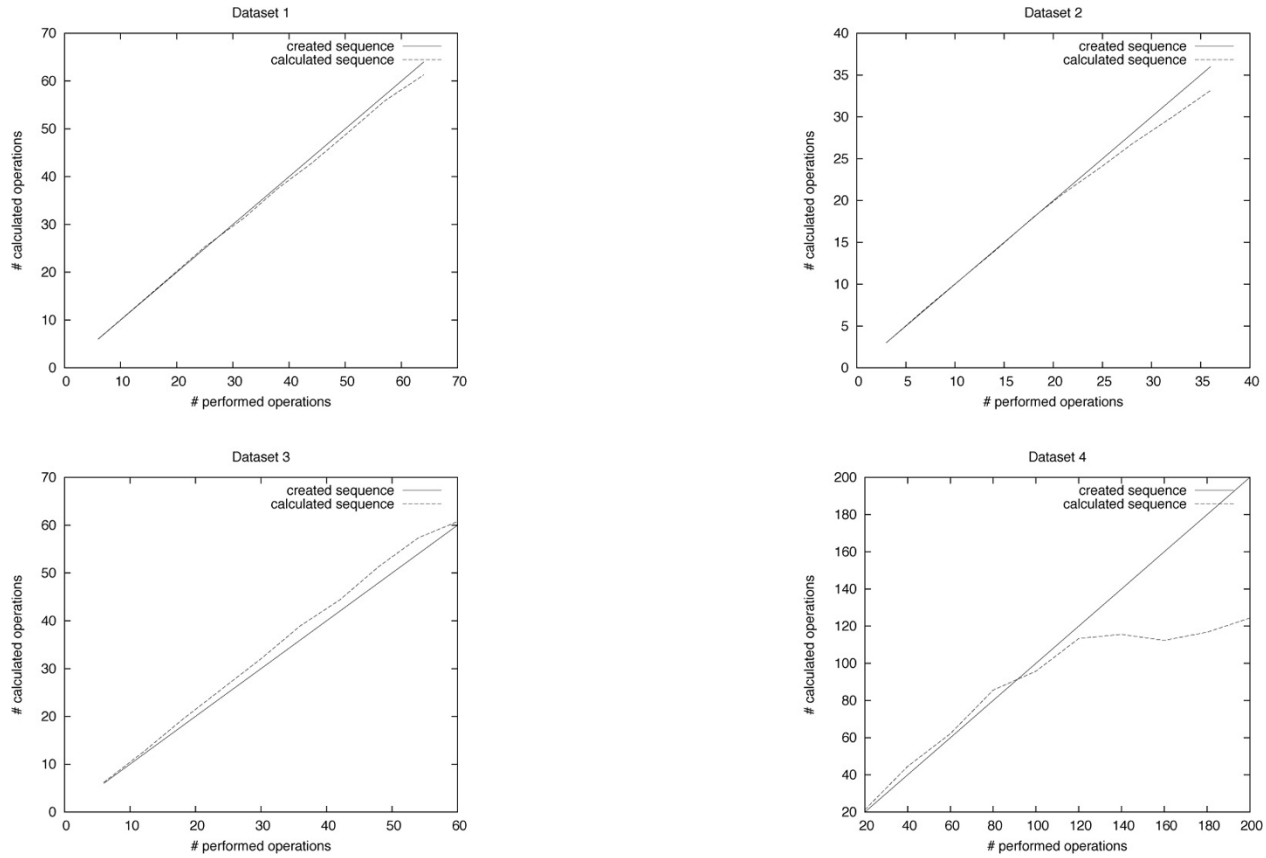

Figure 2
Results on artificial data. The relation of the created distance, the calculated distance, and the lower bound for the different artificial datasets.

diagrams show that, on average, the calculated distance and the true evolutionary distance w lie close together. In many cases, the calculated scenarios were even slightly shorter than the true distance. In the fourth diagram, an additional saturation effect can be observed, i.e. we can find a sorting sequence with weight $\lesssim 130$ for most genomes π, independent of the true distance w.

Cancer karyotypes

The "Mitelman Database of Chromosome Aberrations in Cancer" [21] contains the descriptions of cancer karyotypes which have been manually collected from publications over the last twenty years. For our experiments, we used the version of May 14, 2009, which contains 56428 datasets. The data is represented in the ISCN format, which can be parsed by the software tool CyDAS [29]. From all datasets which could be parsed by CyDAS without error (44064 datasets), we removed all unknown elements and compressed all segments without breakpoint, i.e. if a set of consecutive elements contains no breakpoint in any chromosome, it can be represented as one element. The resulting datasets were used as input to our algorithm. Most of the scenarios are rather easy to reconstruct, the average lower bound is 2.72 and the average calculated weight is 4.08. However, there are some more complicated karyotypes, with rearrangement scenarios of over 50 operations. Exemplarily, the reconstructed scenario for one karyotype is shown in Fig. 3. This karyotype was reported in [30], and can be described by the ISCN formula (for details about the ISCN format, see [31]).

47, *XY*, *t*(3;7)(*q*23;*q*22), *t*(3;7;9)(*q*23;*q*32;*q*22), +*i*(7)(*q*10), *t*(14;18)(*q*32;*q*21), *del*(17)(*p*11)

In principle, our algorithm correctly identified all operations. The triple translocation *t*(3; 7; 9)(*q*23; *q*32; *q*22) and the new chromosome +*i*(7)(*q*10) are not allowed operations in our model. Our algorithm replaced the triple translocation by two translocations, and the new chromosome by a tandem duplication with a subsequent fission, which are the best possible explanations within our model.

Discussion

In the last sections, we have shown that our algorithm will terminate in any case, and finds rearrangement scenarios of reasonable quality. However, the weights of the operations are chosen due to a mathematical model and do not reflect the biological reality. This leaves room for further investigations. For example, the algorithm could be improved by giving unwanted operations a larger weight or completely omit them. While adapting the theoretical model to other weights seems to be the obvious way to improve the algorithm, it might also be worth to examine how robust the results are w.r.t. the chosen weights. In other words, does the optimal rearrangement scenario change when we use other weights? Some observations in the genome can be explained at best by a specific operation (e.g. a duplicated chromosome is most likely caused by a single chromosomal duplication), no matter how this operation is weighted. Such observations are predominant in closely related genomes, and the corresponding operations can be reconstructed even with a wrong weighting scheme. In more diverged genomes, there are often different possible rearrangement scenarios, and the

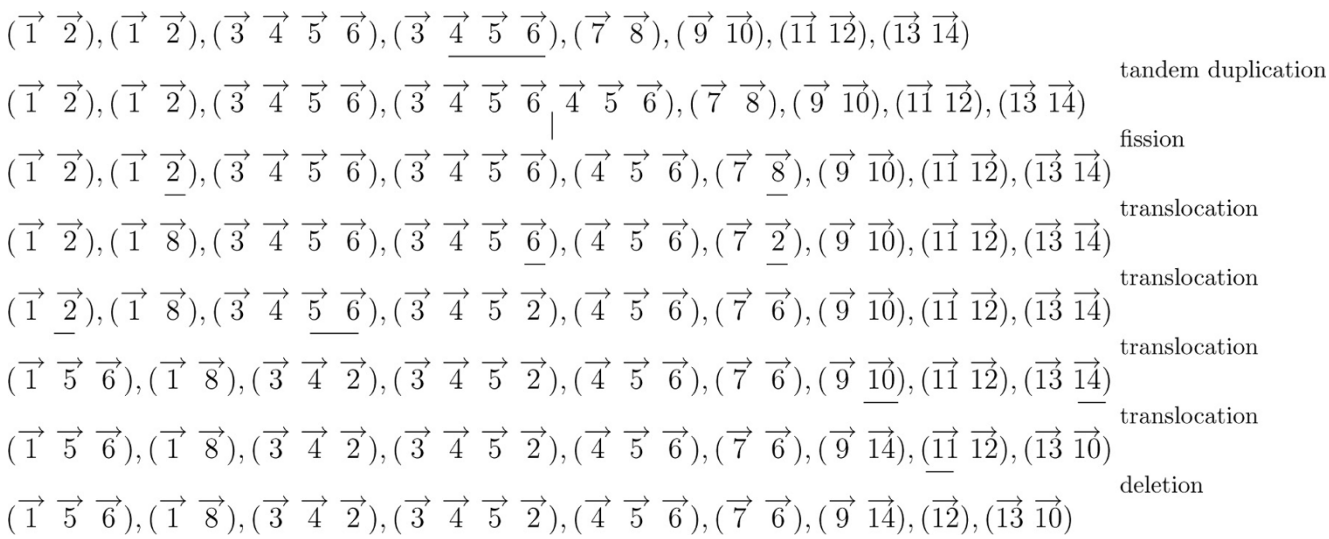

Figure 3

An example sorting scenario. Sorting scenario of a cancer karyotype that was reported in [30]. For better readability, all chromosomes that are identical in ρ and π are removed.

weighting scheme matters. Thus, further investigations should examine what the "critical distance" between two genomes is, i.e. up to which distance the optimal rearrangement scenario is mostly robust w.r.t. the weighting scheme.

Conclusion

We presented an algorithm to sort multichromosomal genomes by a wide range of different operations. Although our results are promising, this algorithm should be seen as a single step towards an algorithm that produces biologically reliable results. While one direction of further research should investigate the chosen weighting scheme (see Section "Discussion"), other possible improvements are closer lower bounds or better heuristics.

Competing interests

The authors declare that they have no competing interests.

Authors' contributions

MB designed the algorithm, implemented it, performed the tests, and drafted this manuscript.

Acknowledgements

MB wants to thank Enno Ohlebusch for valuable comments.

This article has been published as part of *BMC Bioinformatics* Volume 11 Supplement 1, 2010: Selected articles from the Eighth Asia-Pacific Bioinformatics Conference (APBC 2010). The full contents of the supplement are available online at http://www.biomedcentral.com/1471-2105/11?issue=S1.

References

1. Hannenhalli S and Pevzner P: **Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals.** *Journal of the ACM* 1999, **46:**1–27.
2. Hannenhalli S and Pevzner P: **Transforming Men into Mice (polynomial algorithm for genetic distance problem).** *Proc. 36th Annual IEEE Symposium on Foundations of Computer Science* 1995, 581–592.
3. Ozery-Flato M and Shamir R: **Two Notes on Genome Rearrangements.** *Journal of Bioinformatics and Computational Biology* 2003, **1:**71–94.
4. Hartman T and Shamir R: **A Simpler and Faster 1.5-Approximation Algorithm for Sorting by Transpositions.** *Information and Computation* 2006, **204(2):**275–290.
5. Hartman T and Sharan R: **A 1.5-Approximation Algorithm for Sorting by Transpositions and Transreversals.** *Journal of Computer and System Sciences* 2005, **70(3):**300–320.
6. Gog S, Bader M and Ohlebusch E: **GENESIS: genome evolution scenarios.** *Bioinformatics* 2008, **24(5):**711–712.
7. Yancopoulos S, Attie O and Friedberg R: **Efficient sorting of genomic permutations by translocation, inversion and block interchange.** *Bioinformatics* 2005, **21(16):**3340–3346.
8. Blanc G, Barakat A, Guyot R, Cooke R and Delseny M: **Extensive Duplication and Reshufling in the Arabidopsis Genome.** *The Plant Cell* 2000, **12:**1093–1101.
9. Sankoff D: **Gene and genome duplication.** *Current Opinion in Genetics and Development* 2001, **11:**681–684.
10. Sankoff D: **Genome Rearrangement with Gene Families.** *Bioinformatics* 1999, **15:**909–917.
11. Blin G, Chauve C and Fertin G: **The breakpoint distance for signed sequences.** *Proc. 1st International Conference on Algorithms and Computational Methods for Biochemical and Evolutionary Networks* 2004, 3–16.
12. Bryant D: **The complexity of calculating exemplar distances.** *Comparative Genomics* Kluwer Academic Publishers: Sankoff D, Nadeau J 2000, 207–211.
13. Chen X, Zheng J, Fu Z, Nan P, Zhong Y, Lonardi S and Jiang T: **The assignment of orthologous genes via genome rearrangement.** *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 2005, **2(4):**302–315.
14. El-Mabrouk N: **Reconstructing an ancestral genome using minimum segments duplications and reversals.** *Journal of Computer and System Sciences* 2002, **65:**442–464.
15. Bertrand D, Lajoie M, El-Mabrouk N and Gascuel O: **Evolution of Tandemly Repeated Sequences Through Duplication and Inversion.** *Proc. 4th RECOMB Comparative Genomics Satellite Workshop* Lecture Notes in Computer Science, Springer-Verlag; 2006, **4205:**129–140.
16. El-Mabrouk N: **Sorting Signed Permutations by Reversals and Insertions/Deletions of Contiguous Segments.** *Journal of Discrete Algorithms* 2001, **1:**105–122.
17. Marron M, Swenson K and Moret B: **Genomic Distances under Deletions and Insertions.** *Theoretical Computer Science* 2004, **325 (3):**347–360.
18. Swenson K, Marron M, Earnest-DeYoung J and Moret B: **Approximating the True Evolutionary Distance Between Two Genomes.** *ACM Journal of Experimental Algorithmics* 2008, **12:** 3.5:1–3.5:17.
19. Yancopoulos S and Friedberg R: **Sorting genomes with insertions, deletions and duplications by DCJ.** *Proc. 6th Annual RECOMB Satellite Workshop on Comparative Genomics* Lecture Notes in Bioinformatics, Springer-Verlag; 2008, **5267:**170–183.
20. Ozery-Flato M and Shamir R: **On the frequency of genome rearrangement events in cancer karyotypes.** *Proc. 1st RECOMB Satellite Workshop in Computational Cancer Biology* 2007, 17.
21. Mitelman F, Johansson B and Mertens F: *Mitelman Database of Chromosome Aberrations in Cancer* 2008 http://cgap.nci.nih.gov/Chromosomes/Mitelman.
22. Ozery-Flato M and Shamir R: **Sorting Cancer Karyotypes by Elementary Operations.** *Proc. 6th Annual RECOMB Satellite Workshop on Comparative Genomics* Lecture Notes in Bioinformatics, Springer-Verlag; 2008, **5267:**211–225.
23. Bader M: **Sorting by reversals, block interchanges, tandem duplications, and deletions.** *BMC Bioinformatics* 2009, **10 (Suppl 1):**S9.
24. Bafna V and Pevzner P: **Genome Rearrangements and Sorting by Reversals.** *SIAM Journal on Computing* 1996, **25(2):**272–289.
25. El-Mabrouk N, Nadeau J and Sankoff D: **Genome Halving.** *Proc. 9th Annual Symposium on Combinatorial Pattern Matching* Lecture Notes in Bioinformatics, Springer-Verlag; 1998, **1448:**235–250.
26. Zheng C, Zhu Q and Sankoff D: **Genome Halving with an Outgroup.** *Evolutionary Bioinformatics* 2006, **2:**319–326.
27. Salse J, Piégu B, Cooke R and Delseny M: **Synteny between Arabidopsis thaliana and rice at the genome level: a tool to identify conservation in the ongoing rice genome sequencing project.** *Nucleic Acids Research* 2002, **30(11):**2316–2328.
28. Drosophila 12 Genomes Consortium: **Evolution of genes and genomes on the Drosophila phylogeny.** *Nature* 2007, **450:**203–218.
29. Hiller B, Bradtke J, Balz H and Rieder H: **CyDAS: a cytogenetic data analysis system.** *BioInformatics* 2005, **21(7):**1282–1283 http://www.cydas.org.
30. Cabanillas F, Pathak S, Trujillo J, Manning J, Katz R, McLaughlin P, Welasquez W, Hagemeister F, Goodacre A, Cork A, Butler J and Freireich E: **Frequent Nonrandom Chromosome Abnormalities in 27 Patients with Untreated Large Cell Lymphoma and Immunoblastic Lymphoma.** *Cancer Research* 1988, **48:**5557–5564.
31. Mitelman F: *Iscn 1995: An International System For Human Cytogenetic Nomenclature. S. Karger AG* 1995.