The Problem of Chromosome Reincorporation in DCJ Sorting and Halving

Jakub Kováč[1,2], Marília D. V. Braga[2], and Jens Stoye[2]

[1] Department of Computer Science, Comenius University, e-mail: kuko@ksp.sk
[2] AG Genominformatik, Technische Fakultät, Universität Bielefeld
e-mail: mbraga@cebitec.uni-bielefeld.de, stoye@techfak.uni-bielefeld.de

Abstract. We study two problems in the double cut and join (DCJ) model: sorting – transforming one multilinear genome into another and halving – transforming a duplicated genome into a perfectly duplicated one. The DCJ model includes rearrangement operations such as reversals, translocations, fusions and fissions. We can also mimic transpositions or block interchanges by two operations: we extract an appropriate segment of a chromosome, creating a temporary circular chromosome, and in the next step we reinsert it in its proper place. Existing linear-time algorithms solving both problems ignore the constraint of reincorporating the temporary circular chromosomes immediately after their creation. For the restricted sorting problem only a quadratic algorithm was known, whereas the restricted halving problem was stated as open by Tannier, Zheng, and Sankoff. In this paper we address this constraint and show how to solve the problem of sorting in $O(n \log n)$ time and halving in $O(n^{3/2})$ time.

1 Introduction

During evolution, genomes undergo large-scale mutations: a segment of DNA can get reversed, or moved to another position. In genome rearrangement problems we try to find a shortest sequence of operations transforming one genome into another. Such a sequence explains the differences between the genomes and its length can be used to estimate the evolutionary distance.

The *double cut and join* (DCJ) operation, introduced by Yancopoulos et al. [1], models most of the large-scale mutation events, such as reversals, translocations, fusions, and fissions in a unified way. Furthermore, transpositions and block interchanges can be simulated by two operations: an appropriate segment of a chromosome is extracted, creating a temporary circular chromosome, which is then reinserted at the proper place in the next step.

The sorting algorithm given by Yancopoulos et al. [1] running in quadratic time guarantees that each new circular chromosome is immediately reincorporated, thus mimicking transpositions and block interchanges.

Bergeron et al. [2] restated the model and gave a simple linear-time algorithm for DCJ sorting ignoring the reincorporation constraint. However, the algorithm finds a sequence of DCJ operations without any explicit mention of the underlying operations (reversals, translocations, block interchanges, etc.) and many circular chromosomes may coexist in intermediate stages of the sorting process. Such sorting sequences are not biologically plausible e.g. in eukaryotic organisms that typically have only linear chromosomes.

In this work we revisit the original study of Yancopoulos et al. [1]. We borrow techniques from other studies on sorting by reversals and block interchanges [3–5] and propose a new algorithm that sorts multichromosomal linear genomes in the DCJ model, reincorporates circular chromosomes and runs in $O(n \log n)$ time.

Furthermore, we present a new result on the *halving* problem. In the halving problem we imagine a genome that underwent a whole genome duplication and then evolved by large-scale rearrangements. Given a present genome in which all markers are in two copies (paralogs), we try to reconstruct the genome right after the duplication, where each chromosome has its perfectly duplicated copy.

If no restriction on the linearity of chromosomes is imposed and no guarantee concerning circular reintegration is required, we can use linear-time algorithms proposed by Warren and Sankoff [6] and Mixtacki [7]. However, given a multilinear genome, these algorithms may predict some circular chromosomes in the ancestral genome. In the worst case, these algorithms may even produce $\Omega(n)$ circular chromosomes given a single linear chromosome of length n. Again, this is not biologically plausible, when organisms with linear genomes are considered.

The restricted halving problem has not been studied previously and is stated as open in [8]. In this paper we propose an algorithm to solve the halving problem for multichromosomal linear genomes with circular reincorporation in $O(n^{3/2})$ time.

The paper is organized as follows: in Section 2 we introduce the DCJ model and review the previous results, and in Section 3 we describe efficient data structures representing multilinear genomes. We solve the restricted versions of sorting and halving problems in Sections 4 and 5, respectively, and conclude in Section 6.

2 Preliminaries

Genome model. In the DCJ model, genomes Π and Γ consist of the same set of markers (genes, synteny blocks). Every marker g has two ends, called extremities, which we denote g^- and g^+.

Each extremity p is either adjacent to some other extremity q (two consecutive markers on a chromosome), or it is a telomere – the end of a linear chromosome. In the first case we say the extremities form an *adjacency* pq, in the second case we have a *telomeric adjacency* $p\circ$. Thus a genome is a set of adjacencies such that every extremity is in exactly one (possibly telomeric) adjacency.

For genome Π we can draw a genome graph G_Π where vertices are extremities and edges connect either adjacent extremities or two extremities of the same marker. Every vertex in this graph has degree 1 or 2, so the components of G_Π

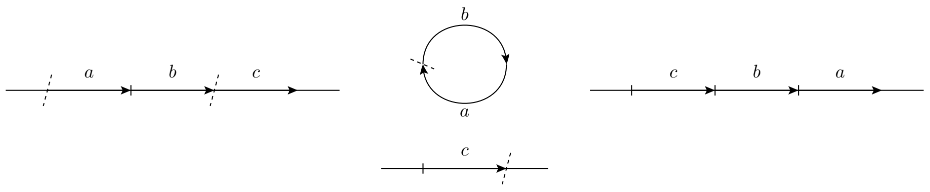

Fig. 1. To interchange blocks a and c (left) in the DCJ model we cut before a and after b and create a temporary circular chromosome (middle). The next operation cuts between a and b and after c and reincorporates the blocks in the correct order (right).

are paths and cycles. These components represent chromosomes – linear and circular, respectively.

DCJ operation. A *double cut and join operation acting on adjacencies* pq *and* rs replaces them by either adjacencies pr, qs, or ps, qr (the adjacencies pq and rs can be telomeric or even an empty chromosome $\circ\circ$). We say the operation *cuts* pq and rs and *joins* either pr, qs, or ps, qr. By these operations we can mimic every common rearrangement operation in genomes: To invert a segment, we cut at its ends and join reversed. By cutting and joining adjacencies on different linear chromosomes, we get a translocation. By cutting two telomeric adjacencies $p\circ$ and $q\circ$ and joining pq we can fuse two chromosomes into one or create a circular chromosome from a linear one (as a byproduct we get an empty chromosome $\circ\circ$). By two DCJ operations we can mimic transpositions and block interchanges: We first cut out an appropriate segment and by joining its ends create a temporary circular chromosome. In the next step we reincorporate it into the original chromosome (see Fig. 1).

DCJ distance and scenarios. A sequence of k DCJ operations transforming a given genome Π into Γ is called a *DCJ scenario of length* k. A scenario of minimum length is called *optimal* and its length is the *DCJ distance between* Π *and* Γ, denoted $d(\Pi, \Gamma)$. A sequence of k DCJ operations transforming Π into Π' is *optimal* (with respect to Γ), if $d(\Pi, \Gamma) = d(\Pi', \Gamma) + k$.

The distance and a sorting scenario can be calculated using an adjacency graph $AG(\Pi, \Gamma)$. This is a bipartite graph where vertices are adjacencies of Π and Γ; an adjacency in Π is connected with an adjacency in Γ, if they share an extremity. Since every adjacency is connected with one (telomeric) or two other adjacencies, this graph consists of paths and cycles only. If Π and Γ share a *common adjacency*, this corresponds to a cycle of length 2 or path of length 1 (common telomere) in the adjacency graph. Note that when Π and Γ are equal, their adjacency graph consists of 2-cycles and 1-paths only.

The following theorem gives the DCJ distance between two genomes:

Theorem 1 (Bergeron et al. [2]). *Given two genomes* Π *and* Γ *on* n *markers, let* c *be the number of cycles and* p_o *the number of odd length paths in the adjacency graph* $AG(\Pi, \Gamma)$*. Then the distance between* Π *and* Γ *is*

$$d(\Pi, \Gamma) = n - (c + p_o/2) \ .$$

Halving problem. In the halving problem we imagine a genome that underwent a whole genome duplication and then evolved into genome Γ. We are given Γ and our goal is to reconstruct the genome before the whole genome duplication. More formally: In a duplicated genome, every marker g has two copies – $g_1^- g_1^+$ and $g_2^- g_2^+$. If p is an extremity, we will denote by $\bar{p}$ the other copy – the paralogous extremity. Similarly, if $x = pq$ is an adjacency (possibly telomeric), then $\bar{x} = \bar{p}\bar{q}$ is the paralogous adjacency, and if C is a chromosome (set of adjacencies), then $\overline{C}$ is the set of paralogous adjacencies.

We say that genome Θ is *perfectly duplicated*, if for each adjacency pq in Θ, adjacency $\bar{p}\bar{q}$ is also in Θ and $p \neq \bar{q}$. This is the same as saying that if we ignore the subscripts (1's and 2's), every linear chromosome has an identical copy and every circular chromosome has either an identical copy, or is itself composed of two successive identical copies.

The genome halving problem can be stated as follows: Given a duplicated genome Γ, find a perfectly duplicated genome Θ such that $d(\Theta, \Gamma)$ is minimal.

The halving distance and scenario can be calculated using an analogy of an adjacency graph – a natural graph $NG(\Gamma)$ introduced by El-Mabrouk and Sankoff [9]. Vertices of this graph are adjacencies of Γ, and two adjacencies are connected by an edge, if they share a paralogous extremity. The natural graph consists of paths and cycles only, and Θ is perfectly duplicated if and only if $NG(\Theta)$ consists of 2-cycles and 1-paths only.

The following theorem gives the DCJ halving distance:

Theorem 2 (Mixtacki [7]). *Let Γ be a duplicated genome with $2n$ markers. The minimal distance between Γ and any perfectly duplicated genome Θ is*

$$d(\Gamma, \Theta) = n - (c_e + \lfloor p_o/2 \rfloor),$$

where c_e is the number of even cycles and p_o the number of odd paths in the natural graph $NG(\Gamma, \Theta)$.

Linear chromosomes. From now on we will be interested in genomes with linear chromosomes only. These *multilinear* genomes are more comfortably written as signed permutations: Choose a direction of a linear chromosome, and list the markers from left to right; write $\overrightarrow{g}$, if extremity g^- is before g^+ and $\overleftarrow{g}$ otherwise. Thus chromosome ($\overrightarrow{1}$, $\overrightarrow{3}$, $\overleftarrow{2}$) (which is the same as ($\overrightarrow{2}$, $\overleftarrow{3}$, $\overleftarrow{1}$)) corresponds to the set of adjacencies $\{ \circ 1^-, 1^+3^-, 3^+2^+, 2^-\circ \}$. We will write $-g$ for the reversed marker g, i.e. $-\overleftarrow{g} = \overrightarrow{g}$ and $-\overrightarrow{g} = \overleftarrow{g}$.

Restricted sorting and halving. Given multilinear genomes, we call a sorting or halving DCJ scenario *restricted*, if every DCJ operation that creates a circular chromosome is immediately followed by another operation that reintegrates it into the original chromosome. Such scenarios can be viewed as sequences of reversals, translocations, fusions, fissions (with weight 1) and block interchanges, which have weight 2, i.e. count as two operations. In the restricted sorting and halving problems, we are searching for restricted scenarios of minimal length. Note that in both cases the distance remains the same as in their unrestricted versions.

3 Data Structures for Handling Permutations

Our algorithms use two efficient data structures for handling permutations described by Kaplan and Verbin [10] and Han [11].

Tree-based data structure. The following data structure from [10] can be traced back to Chroback et al. [12]. It supports the following three operations in logarithmic time: find the i^{th} marker in a linear chromosome, return the position of marker g and perform a reversal operation.

Linear chromosomes can be represented by a balanced tree supporting operations split and merge (e.g. red-black tree or splay tree). The order is the same as the left-to-right order of markers on the chromosome. In each node of the tree we store one marker, its orientation, number of descendants and a reverse flag. A reverse flag being "on" signifies that the whole subtree is reversed. The reverse flag of node v can be cleared ("pushed down") by changing v's orientation, swapping its children and flipping their reverse flags.

Reversing a segment from i to j can be implemented as follows:

1. Find the i^{th} and j^{th} marker (using the information about sizes of subtrees and reverse flags).
2. Split the tree into three parts: T_1 with markers before i, T_3 with markers after j and T_2 with the segment from i to j.
3. Flip the reverse flag in the root of T_2, and
4. Merge T_1, T_2 and T_3.

We store a lookup table with a pointer to the corresponding node of a tree for every marker. In this way we can find the position of any marker in logarithmic time.

This data structure can be easily extended to multiple linear chromosomes and to support different operations such as translocations or block interchanges. Actually, we do not need to have one tree per chromosome: simply concatenate the chromosomes with a delimiter between them and in each node store the number of delimiters in its subtree. This way given a marker g we can tell on which chromosome it is by counting the number of delimiters before g and all the rearrangement operations can be performed using a few reversal operations.[3]

Block-based data structure. The second data structure by Kaplan and Verbin [10] is a two-level version of the previous one. This is the data structure used in the subquadratic algorithms for sorting by reversals [13] and sorting by translocations [14].

As with the previous data structure, we concatenate all the chromosomes using delimiters. We divide the whole sequence into blocks of size between $\frac{1}{2}\sqrt{n \log n}$ and $2\sqrt{n \log n}$. Note that there are $O(\sqrt{n/\log n})$ blocks and one

[3] For example block interchange can be mimicked by 4 reversals; if we add sufficiently many delimiters at the end of the sequence (representing empty chromosomes), we can also mimic fusions and fissions.

block can contain several chromosomes. In each block we store an array of markers and a tree-based data structure storing their paralogs ordered by *positions of the paralogs* in the genome. Furthermore, for each block we keep the number of markers in it and a reverse flag which signifies that the whole block is reversed. We have an additional lookup table with blocks and indices of the markers, so that we can tell the position of a given marker in constant time.

Finding the i^{th} marker can be done trivially in $O(\sqrt{n/\log n})$ time and can be improved to $O(\log n)$ by building a balanced tree over the blocks.

Reversal of a segment can be implemented in $O(\sqrt{n \log n})$ time as follows:

1. Find the two endpoints and split the two blocks (if necessary) so that the endpoints of the reversal correspond to the endpoints of blocks (we can temporarily break the invariant about the block size; the trees with paralogs are rebuilt from scratch in $O(\sqrt{n \log n})$ time).
2. Reverse the order of the blocks between the endpoints and flip their reverse flags.
3. For each block (inside and outside the reversal) take its tree with paralogs T and split it into three parts: T_2 with paralogs within the reversal, T_1 with paralogs before and T_3 with paralogs after the reversal. If a block is outside the reversal, flip the reverse flag in the root of T_2, otherwise flip the flags in roots of T_1 and T_3. Merge T_1, T_2 and T_3. Since there are $O(\sqrt{n/\log n})$ blocks and all the split and merge operations can be done in $O(\log n)$ time, this step can be implemented in $O(\sqrt{n \log n})$ time.
4. Split and merge blocks so that the size of each one is between $\frac{1}{2}\sqrt{n \log n}$ and $2\sqrt{n \log n}$.

Again, we can simulate any other DCJ operation by a constant number of reversals.

The neat thing about this block-based data structure is that we actually maintain the markers according to two orders – by their position in the genome and by the position of their paralogs. This property can be used to implement the following query used in our halving algorithm in $O(\sqrt{n \log n})$ time: Given a chromosome C and two markers i and j on a possibly different chromosome, find the right-most marker on C that has a paralog between markers i and j:

1. Temporarily split the blocks at the ends of chromosome C, so that C is contained in several whole blocks.
2. Find the rightmost block within C containing a marker with paralog between i and j. Since the paralogs are stored in balanced trees, the membership questions can be answered in $O(\log n)$ time and there are $O(\sqrt{n/\log n})$ blocks.
3. In this block find the required marker by a sequential search in $O(\sqrt{n \log n})$ time.
4. Merge the temporarily split blocks.

Note that only a slightly more complicated data structure achieving time complexity $O(\sqrt{n})$ for the same operations was given by Han [11]. We refer the interested reader to this paper.

4 Restricted DCJ Sorting

Previous work. Bergeron et al. [2] gave a linear-time algorithm for DCJ sorting disregarding the constraint of reincorporating circular chromosomes immediately. The solution can be easily adapted to a quadratic algorithm for the restricted version: after each step check whether a circular chromosome was created and if so, find the appropriate DCJ operation acting on adjacencies in the circular and the original linear chromosome that reintegrates the circular chromosome. It is not obvious how to do this fast (say in polylogarithmic time).

Yancopoulos et al. [1] proposed to transform Π into Γ by restricted sorting in four stages: 0. Add caps to the ends of linear chromosomes. 1. By translocations, fusions and fissions transform Π into Π' such that chromosomes in Π' and Γ have the same marker contents. 2. Perform oriented reversals to get Π'' with all markers in the same direction as in Γ. 3. Finally, use block interchanges to transform Π'' into Γ.

Stages 2 and 3 can be implemented in $O(n \log n)$ time using the data structure described in Section 3 [5, 4]. Thus a *unichromosomal* restricted DCJ sorting can be solved in $O(n \log n)$ time. However, it is not obvious how to implement stage 1 in a fast way.

Capping. The ends of linear chromosomes, telomeres, produce some difficulties and nasty special cases. Capping is an elegant technique to deal with them: we adjoin new markers (caps) to the ends so that we do not change the distance and we do not have to worry about telomeres any more.

We find all the paths in the adjacency graph $AG(\Pi, \Gamma)$. Paths of odd length have one end in Π and one in Γ – simply adjoin a new marker (properly oriented) to the two telomeres. This increases the number of markers by one, but instead of an odd path we have a cycle and a 1-path, so the distance does not change. For paths starting and ending in Π add two new markers to the ends of Π and a new chromosome consisting of just these two markers (properly oriented) to Γ. The case with a path starting and ending in Γ is symmetric. The number of markers increases by 2, but instead of an even path, we have a cycle and two odd paths, so the distance does not change. Capping of all chromosomes can be done in linear time.

Our algorithm. The algorithm is based on the following observation:

Observation 1. *Let g, h be two markers that are adjacent in Γ, but not in Π. If g and h are on different chromosomes in Π, there is a translocation that puts them together. This is an optimal operation in the DCJ model. If g and h are on the same chromosome and have a different orientation, there is a reversal that puts them together. This is an optimal operation in the DCJ model. Transposition and block interchange take two DCJ operations. These operations are optimal if they create two new non-telomeric common adjacencies and destroy none.*

This is simply because, even more generally, k operations that create k new non-telomeric adjacencies and destroy none create k new cycles in the adjacency graph, and thus decrease the distance by k.

Theorem 3. *A restricted optimal DCJ scenario transforming multilinear genome* Π *into* Γ *can be found in* $O(n \log n)$ *time.*

Proof. Cap all chromosomes first. Without loss of generality we may assume that the markers in chromosomes of Γ are consecutive numbers $(\overrightarrow{k_0}, \dots, \overrightarrow{k_1 - 1})$, $(\overrightarrow{k_1}, \dots, \overrightarrow{k_2 - 1})$, $\dots$,$(\overrightarrow{k_{s-1}}, \dots, \overrightarrow{k_s - 1})$ where $0 = k_0 < k_1 < k_2 < \dots < k_s = n$ (otherwise renumber the markers).

We will be transforming Π into Γ gradually "from left to right": once we have transformed the beginning of a chromosome in Π to $\overrightarrow{k_i}, \overrightarrow{k_i + 1}, \dots, \overrightarrow{j}$, we extend it by moving $j + 1$ next to $\overrightarrow{j}$.

There are several cases we need to consider:

1. If $\overrightarrow{j+1}$ is already next to $\overrightarrow{j}$, we are done.
2. If $j + 1$ is on a different chromosome than $\overrightarrow{j}$, we can always use a translocation. In the rest of the proof we assume that $j+1$ is on the same chromosome, to the right of $\overrightarrow{j}$.
3. If $\overrightarrow{j}$ and $\overleftarrow{j+1}$ have different orientation, we can use a reversal.

Otherwise, following [3], find the marker m with the highest number between $\overrightarrow{j}$ and $\overrightarrow{j+1}$ and find $m + 1$.

4. If $m + 1$ is on a different chromosome, we can use a translocation to move it next to m; this operation also moves $\overrightarrow{j+1}$ to another chromosome, so we can use another translocation to move it next to $\overrightarrow{j}$.

Otherwise the situation is $\overrightarrow{j}, \dots, m, \dots, \overrightarrow{j+1}, \dots, m + 1$ (since m is the highest number between $\overrightarrow{j}$ and $\overrightarrow{j+1}$ and the part of the chromosome to the left of $\overrightarrow{j}$ is already sorted, $m + 1$ must be to the right of $\overrightarrow{j+1}$).

5. If m and $m + 1$ have different orientation, we can use a reversal to move $m+1$ next to m; this will also change the orientation of $\overrightarrow{j+1}$, so in the next step we can use another reversal to move $\overleftarrow{j+1}$ next to $\overrightarrow{j}$.
6. Finally, if m and $m + 1$ have the same orientation, we interchange blocks

$$\overrightarrow{j}, [\dots, \overrightarrow{m}], \dots, [\overrightarrow{j+1}, \dots], \overrightarrow{m+1} \quad \rightsquigarrow \quad \overrightarrow{j}, \overrightarrow{j+1}, \dots, \overrightarrow{m}, \overrightarrow{m+1}$$

   if both $\overrightarrow{m}$ and $\overrightarrow{m+1}$ have positive direction and

$$\overrightarrow{j}, [\dots], \overleftarrow{m}, \dots, [\overrightarrow{j+1}, \dots, \overleftarrow{m+1}] \quad \rightsquigarrow \quad \overrightarrow{j}, \overrightarrow{j+1}, \dots, \overleftarrow{m+1}, \overleftarrow{m},$$

   if $\overleftarrow{m}$ and $\overleftarrow{m+1}$ have both negative direction. By two operations we move $\overrightarrow{j+1}$ to $\overrightarrow{j}$ and $\overrightarrow{m}$ to $\overrightarrow{m+1}$.

Every step can be implemented in $O(\log n)$ time using an extended version of the data structure from Section 3. We need the data structure to support the following operations: 1. Given a marker, find the chromosome that contains it. 2. Given interval $i, \dots, j$ find the marker with the highest number on the chromosome between i and j (store the highest number in the subtree in each node). 3. Perform a DCJ operation (this can be done by splitting, merging trees and lazy reversals as described). $\square$

Perfect DCJ scenarios. Bérard et al. [15] studied the problem of finding a scenario transforming genome Π into Γ that does not break a given set of common intervals. An *interval* in genome Π is a set of markers such that the subgraph of G_Π induced by their extremities is connected. Intervals of Π have zero or two borders – adjacencies such that one extremity is inside and one outside. Let I be any set of markers with zero or two borders. A DCJ operation *preserves* I, if I still has zero or two borders in the resulting genome. They showed that for nested sets of common intervals (when the intervals do not overlap) the shortest scenario can be found in polynomial time and for weakly separable sets the problem is NP-hard, but fixed parameter tractable.

Since their algorithm uses algorithms for DCJ distance and sorting as a black box, one can use it in conjunction with our algorithm to get perfect restricted DCJ scenarios.

5 Restricted DCJ Halving

Previous work. The halving problem in the DCJ model was studied by Warren and Sankoff [6] and corrected and simplified by Mixtacki [7]. However, the restricted version of the halving problem has not been studied and is stated as open by Tannier, Zheng, and Sankoff [8].

The simple approach that works for sorting – do a DCJ operation, test whether a circular chromosome was created and reincorporate it – does not work for halving: In some cases the circular chromosome cannot be reincorporated. For example take chromosome $(1_1, 1_2, 2_1, 2_2)$ – after excision of circular chromosome $[1_1, 1_2]$ it is not possible to reincorporate it and the algorithm of Mixtacki [7] ends with two (perfectly duplicated) circular chromosomes $[1_1, 1_2]$ and $[2_1, 2_2]$. On the other hand, by fission and translocation we can get

$$(1_1 \mid 1_2, 2_1, 2_2) \quad \rightsquigarrow \quad (1_1 \mid),\ (1_2, 2_1 \mid 2_2) \quad \rightsquigarrow \quad (1_1, 2_2),\ (1_2, 2_1)\ .$$

By giving an algorithm for the restricted halving problem we also show that the halving distance is the same in the restricted and the unrestricted case.

Capping. Find all paths in the natural graph $NG(\Gamma)$. If the length of path is odd, adjoin two paralogous copies of a new marker at both ends. This will create a new 1-path and close the odd path into an even cycle. Thus we will have an extra marker and an extra cycle, the distance is unchanged. If the length of the path is even, we adjoin two different new markers at the ends and create a new linear chromosome consisting of its paralogous copies. The number of markers is increased by 2, but also the number of odd paths increases by 2 and number of even cycles increases by 1, so the distance is unchanged.

Our algorithm. An analogy of Observation 1 from the previous section holds also for the DCJ halving problem:

Observation 2. *Reversals and translocations that create one and block interchanges that create two new non-telomeric common adjacencies and destroy none are optimal.*

Theorem 4. *A restricted optimal DCJ halving scenario transforming duplicated genome* Π *into perfectly duplicated genome* Θ *can be found in* $O(n^{3/2})$ *time. The distance is the same as the unrestricted halving distance.*

Proof. Cap all chromosomes first. This can be done in linear time. Take any chromosome C_1; let i be its first marker (cap). Then there are two cases: the paralog $\overline{i}$ is either on a different chromosome C_2, or it is at the end of C_1 reversed.

In general, we have either two chromosomes C_1 and C_2 starting with paralogous markers, say $i, \ldots, j$ and $\overline{i}, \ldots, \overline{j}$ (with the same orientation), or there is chromosome C starting and ending with paralogous markers (in the opposite orientation).

Case I. The transformation will again go "from left to right": once we have C_1 starting with markers $i, \ldots, j, k$ and C_2 starting with $\overline{i}, \ldots, \overline{j}, \overline{\ell}$, we either move $\overline{k}$ next to $\overline{j}$ in C_2, or ℓ next to j in C_1.

If ℓ is not in C_1, we can use a translocation. Otherwise, if j, ℓ and $\overline{j}, \overline{\ell}$ have different orientation, we can use a reversal to move ℓ next to j. The situation is symmetric and the same holds for $\overline{k}$.

The only hard case arises when j, k, ℓ are on one chromosome, $\overline{j}, \overline{k}, \overline{\ell}$ on another, j, k have the same orientation as $\overline{j}, \overline{k}$ and j, ℓ the same as $\overline{j}, \overline{\ell}$. Then we can write the two chromosomes as

$$C_1 = (i, \ldots, j, k, x_1, \ldots, x_p) \quad \text{and} \quad C_2 = (\overline{i}, \ldots, \overline{j}, y_1, \ldots, y_q, \overline{k}, z_1, \ldots, z_r) \ .$$

We find marker y between $\overline{j}$ and $\overline{k}$ such that its paralog $\overline{y}$ is the rightmost among the x-markers on chromosome C_1. (Note that such a marker exists, since at least $y_1 = \overline{\ell}$ is a marker between $\overline{j}$ and $\overline{k}$ with paralog in C_1.) Let $\overline{y} = x_t$ and let $\overline{z} = x_{t+1}$ – then z is either one of the z-markers on the second chromosome C_2, or it does not lie on C_2 at all. In the latter case we perform two translocations moving first z to y and then $\overline{k}$ to $\overline{j}$, in the former we perform (depending on the mutual orientation of y, z and $\overline{y}, \overline{z}$) either two reversals or one of the two indicated block interchanges:

$$\begin{aligned} &(\overline{i}, \ldots, \overline{j}, \big[y_1, \ldots, y\big], \ldots, y_q, \big[\overline{k}, z_1, \ldots\big], z, \ldots, z_{r+1}) \\ \text{or} \quad &(\overline{i}, \ldots, \overline{j}, \big[y_1, \ldots\big], y, \ldots, y_q, \big[\overline{k}, z_1, \ldots, z\big], \ldots, z_{r+1}) \ . \end{aligned}$$

This way we put $\overline{k}$ next to $\overline{j}$ and y next to z at the same time.

Case II. Chromosome C starts and ends with paralogous markers $i, \ldots, j$:

$$C = (i, \ldots, j, k, \ldots\ldots, -\overline{\ell}, -\overline{j}, \ldots, -\overline{i}) \ .$$

The transformation will go "from outside to the middle": we either move $\overline{k}$ next to $\overline{j}$ or ℓ next to j.

Again, if one of the markers is on a different chromosome, or has opposite orientation, we can move it by a translocation or a reversal to its proper place.

Otherwise, find the leftmost marker m between k and $-\overline{k}$ such that $\overline{m}$ is *not* between k and $-\overline{k}$. If such an m exists, let n be the marker preceding m; note that $\overline{n}$ is between k and $-\overline{k}$. Depending on the orientation of n, m and $\overline{n}, \overline{m}$ and whether $\overline{m}$ is in C or on a different chromosome, we perform either two translocations, two reversals, or a block interchange moving $\overline{k}$ to $\overline{j}$ and $\overline{n}$ to $\overline{m}$. The situation is symmetric and we can try the same with ℓ.

Finally, if in either case such an m does not exist, we have chromosome

$$C = (i, \ldots, j, k,\ x_1, \ldots, x_{2p},\ -\overline{k},\ y_1, \ldots, y_q,\ \ell,\ z_1, \ldots, z_{2r},\ -\overline{\ell}, -\overline{j}, \ldots, -\overline{i}),$$

where $I = \{x_1, \ldots, x_{2p}\}$ and $J = \{z_1, \ldots, z_{2r}\}$, possibly empty, are closed under taking paralogs – these intervals either contain both paralogs g and $\overline{g}$ or neither one. In this case we first recursively reorder markers in I and J and transform C into

$$\begin{gathered} C' = (i, \ldots, j,\ k, k_1, \ldots, k_p, \left[-\overline{k}_p, \ldots, -\overline{k}_1, -\overline{k}\right],\ y_1, \ldots, y_q, \\ \ell, \ell_1, \ldots, \ell_r, \left[-\overline{\ell}_r, \ldots, -\overline{\ell}_1, -\overline{\ell}\right],\ -\overline{j}, \ldots, -\overline{i}), \end{gathered}$$

(the case analysis is the same as Case II) and then perform the indicated block interchange.

At the end we may end up with several chromosomes of the form $C -\overline{C}$, which should be fissioned in two perfectly duplicated chromosomes.

This method shows that the halving distance is the same as the unrestricted distance and the algorithm can be easily implemented in quadratic time. For a more efficient algorithm we need the data structure that supports the following operations: 1. Given a marker find the chromosome that contains it. 2. Given an interval $i, \ldots, j$, find the right-most marker on a given chromosome that has a paralog in the interval $i, \ldots, j$. 3. In a given interval $i, \ldots, j$ find the leftmost/rightmost marker m such that $\overline{m}$ is not in the interval. 4. Perform all the DCJ operations. Using the block-based data structure from Section 3 and the improvement by Han [11] every operation can be performed in $O(\sqrt{n})$ time. $\square$

6 Conclusion

In this work we revisited the restricted DCJ model for multichromosomal linear genomes, where a temporary circular chromosome is immediately reincorporated after its excision. We improved the quadratic algorithm by Yancopoulos et al. [1] and proposed an algorithm that runs in $O(n \log n)$ time.

Furthermore we solved an open problem from [8] by giving an algorithm for the restricted halving problem. The algorithm shows that the halving distance for the restricted version is the same as the distance for the unrestricted version and given a multilinear duplicated genome an optimal *multilinear* perfectly duplicated genome can always be found.

This is not the case for example in the median problem which we did not study and is still open: Consider three linear genomes $(1, 2, 3)$, $(2, 1, 3)$ and

$(2, 3, 1)$. Their median in the unrestricted case consists of linear chromosome $(2, 3)$ and circular $[1]$. The circular genome $[1]$ can be reincorporated into any of the given chromosomes by one operation giving the median score 3. This score however cannot be achieved in the restricted model. Generalizing this pattern, we can get genomes of length $3n$ with unrestricted median score $3n$ and restricted median score $4n$.

Acknowledgements. The authors would like to thank Broňa Brejová for many constructive comments. The research of Jakub Kováč is supported by Marie Curie Fellowship IRG-224886 to Dr. Tomáš Vinař, VEGA grant 1/0210/10, and the Partnership Stipend from the Bielefeld University.

References

1. Yancopoulos, S., Attie, O., Friedberg, R.: Efficient sorting of genomic permutations by translocation, inversion and block interchange. Bioinformatics **21** (2005) 3340–3346
2. Bergeron, A., Mixtacki, J., Stoye, J.: A unifying view of genome rearrangements. In: Proc. of WABI. (2006) 163–173
3. Christie, D.A.: Sorting permutations by block-interchanges. Inf. Process. Lett. **60** (1996) 165–169
4. Feng, J., Zhu, D.: Faster algorithms for sorting by transpositions and sorting by block interchanges. ACM Transactions on Algorithms **3** (2007)
5. Swenson, K.M., Rajan, V., Lin, Y., Moret, B.M.E.: Sorting signed permutations by inversions in $O(n \log n)$ time. JCB **17** (2010) 489–501
6. Warren, R., Sankoff, D.: Genome halving with double cut and join. JBCB **7** (2009) 357–371
7. Mixtacki, J.: Genome halving under DCJ revisited. In Hu, X., Wang, J., eds.: COCOON. Volume 5092 of LNCS., Springer (2008) 276–286
8. Tannier, E., Zheng, C., Sankoff, D.: Multichromosomal median and halving problems under different genomic distances. BMC Bioinformatics **10** (2009)
9. El-Mabrouk, N., Sankoff, D.: The reconstruction of doubled genomes. SIAM J. Comput. **32** (2003) 754–792
10. Kaplan, H., Verbin, E.: Sorting signed permutations by reversals, revisited. J. Comput. Syst. Sci. **70** (2005) 321–341
11. Han, Y.: Improving the efficiency of sorting by reversals. In Arabnia, H.R., Valafar, H., eds.: BIOCOMP, CSREA Press (2006) 406–409
12. Chrobak, M., Szymacha, T., Krawczyk, A.: A data structure useful for finding hamiltonian cycles. Theor. Comput. Sci. **71** (1990) 419–424
13. Tannier, E., Bergeron, A., Sagot, M.F.: Advances on sorting by reversals. Discrete Applied Mathematics **155** (2007) 881–888
14. Ozery-Flato, M., Shamir, R.: An $O(n^{3/2}\sqrt{\log n})$ algorithm for sorting by reciprocal translocations. In Lewenstein, M., Valiente, G., eds.: CPM. Volume 4009 of LNCS., Springer (2006) 258–269
15. Bérard, S., Chateau, A., Chauve, C., Paul, C., Tannier, E.: Computation of perfect dcj rearrangement scenarios with linear and circular chromosomes. JCB **16** (2009) 1287–1309 PMID: 19803733.