# A Linear Time Approximation Algorithm for the DCJ Distance for Genomes with Bounded Number of Duplicates

Diego P. Rubert[1], Pedro Feijão[2], Marília D. V. Braga[2], Jens Stoye[2], and Fábio V. Martinez[1,⋆]

[1] Faculdade de Computação, Universidade Federal de Mato Grosso do Sul, Campo Grande, MS, Brazil, (diego,fhvm)@facom.ufms.br,

[2] Faculty of Technology and Center for Biotechnology (CeBiTec), Bielefeld University, Bielefeld, Germany, (pfeijao,mbraga,stoye)@cebitec.uni-bielefeld.de

**Abstract.** Rearrangements are large-scale mutations in genomes, responsible for complex changes and structural variations. Most rearrangements that modify the organization of a genome can be represented by the *double cut and join* (DCJ) operation. Given two genomes with the same content, so that we have exactly the same number of copies of each gene in each genome, we are interested in the problem of computing the *rearrangement distance* between them, i.e., finding the minimum number of DCJ operations that transform one genome into the other. We propose a linear time approximation algorithm with approximation factor $O(k)$ for the DCJ distance problem, where $k$ is the maximum number of duplicates of any gene in the input genomes. Our algorithm uses as an intermediate step an $O(k)$-approximation for the minimum common string partition problem, which is closely related to the DCJ distance problem. Experiments on simulated data sets show that the algorithm is very competitive both in efficiency and quality of the solutions.

## 1 Introduction

Large-scale mutations or rearrangements can produce complex changes and structural variations in genomes. They include inversions of chromosome segments, translocations of chromosome ends, fusions and fissions of chromosomes. All these rearrangements can be represented by the *double cut and join* (DCJ) operation [15], which basically consists of cutting a genome in two distinct positions (possibly in two distinct chromosomes) and joining the four resultant open ends in a different way.

A basic task in comparative genomics is to find the rearrangement distance between two given genomes, i.e., the minimum number of rearrangements that transform one genome into the other. For genomes without duplicate genes, there are linear time algorithms to compute the distance allowing only DCJ operations [4]. On the other hand, for genomes with duplicate genes, computing the rearrangement distance is NP-hard, even when the genomes have the same content and only DCJ operations are allowed [2, 3].

In this paper we study the problem of computing the DCJ distance between two genomes with the same content and possibly duplicate genes, with the restriction that we

⋆ Corresponding author.

have exactly the same number of copies of each gene in each genome. We propose a linear time approximation algorithm with approximation factor $O(k)$, where $k$ is the maximum number of duplicates of any gene in the input genomes. The main goal is a construction of a consistent decomposition of the corresponding adjacency graph, which is a disjoint cycle decomposition of this graph. And then, we can easily compute the DCJ distance from this decomposition.

To obtain such a decomposition we use a linear time approximation algorithm for the minimum common string partition problem with approximation factor $O(k)$ [11]. It is an efficient approximation for the breakpoint distance (the number of genes in the genome minus the number of preserved adjacencies), an intermediate step of our proposed algorithm. As we will show, the whole procedure is an approximation algorithm with approximation factor $O(k)$ and linear running time for the DCJ distance problem for genomes with the same content and exactly the same number of copies of each gene in each genome. The proposed algorithm works properly on inputs that are linear unichromosomal genomes.

The next section presents a background for describing the DCJ distance problem and Section 3 presents it formally. The subsequent section discusses the algorithm for the minimum common string partition problem and correlates it to the DCJ distance. In Section 5 we develop our approach to compute the DCJ distance. Experiments on simulated data sets are presented in Section 6. The last section concludes the paper.

## 2 Preliminaries

A *gene* $g$ in a genome is an oriented DNA fragment that can be represented by the symbol $g$ itself, if it has direct orientation, or by the symbol $-g$, if it has reverse orientation. Genomes can be partitioned into *chromosomes*, that are linear or circular sequences of genes. Each one of the two ends of a linear chromosome is a *telomere*, represented by the symbol $\circ$.

Each chromosome in a genome can be represented by a string of its genes that can be circular, if the chromosome is circular, or linear and flanked by the symbols $\circ$, if the chromosome is linear. Given a gene $g$, let $m_A(g)$ be the number of occurrences of $g$ in a genome $A$. To refer to each occurrence of a gene $g$ unambiguously, we number the occurrences of $g$ from 1 to $m_A(g)$. When there exists at least one gene that occurs more than once in genome A, we say that $A$ has *duplicate genes*. Consider for instance the genome $A = \{(\circ\ c_1\ -a_1\ d_1\ \circ), (\circ\ b_1\ -a_2\ c_2\ \circ)\}$, composed of two linear chromosomes (each chromosome is flanked by parentheses). In $A$ we have one occurrence of genes $b$ and $d$ and two occurrences of genes $a$ and $c$, that is, $A$ has duplicate genes.

We use the notations $\mathcal{G}(A)$ and $\mathcal{G}^N(A)$, respectively, to refer to the set of (non-numbered) genes and to the set of numbered genes of a genome $A$. Considering again the genome $A$ above, we have $\mathcal{G}(A) = \{a, b, c, d\}$ and $\mathcal{G}^N(A) = \{a_1, a_2, b_1, c_1, c_2, d_1\}$. Observe that the genomes $A' = \{(\circ\ c_1\ -a_2\ d_1\ \circ), (\circ\ b_1\ -a_1\ c_2\ \circ)\}$ and $A'' = \{(\circ\ c_2\ -a_2\ d_1\ \circ), (\circ\ b_1\ -a_1 c_1\ \circ)\}$ are equivalent to $A = \{(\circ\ c_1\ -a_1\ d_1\ \circ), (\circ\ b_1\ -a_2\ c_2\ \circ)\}$. Given a genome $A$, possibly with

duplicate genes, we denote by $[A]$ the equivalence class of genomes that can be obtained from $A$ by swapping indices between occurrences of the same gene.

### 2.1 Balanced Genomes

Given genomes $A$ and $B$ possibly with duplicate genes, if they contain the same number of occurrences of each gene, i.e. $\mathcal{G}^N(A) = \mathcal{G}^N(B)$, we say that $A$ and $B$ are *balanced*. Consequently, $|A| = |B| = n$. Moreover, we define $occ(A) = \max_{g \in A}\{m_A(g)\}$ as the maximum number of duplicates of any gene $g$ in $A$. Thus, if $A$ and $B$ are balanced genomes then $occ(A) = occ(B)$. For simplicity, in this case we use only $occ$. For example, genomes $A = \{(\circ\ c_1\ -a_1\ d_1\ \circ), (\circ\ b_1\ c_2\ \circ), (c_3)\}$ and $B = \{(\circ\ a_1\ \circ), (\circ\ c_3\ -c_1\ -b_1\ \circ), (\circ\ d_1\ c_2\ \circ)\}$ are balanced, since $\mathcal{G}^N(A) = \{a_1, b_1, c_1, c_2, c_3, d_1\} = \mathcal{G}^N(B)$, and $occ = 3$.

### 2.2 DCJ Operations

Rearrangements can change the organization of a genome, i.e., the number of chromosomes in a genome or the order and the orientation of its genes. In general, such a rearrangement cuts a genome in two different positions, creating four open ends, and joins these open ends in a different way. It can be modeled by a *double-cut and join* (DCJ) operation [15]. Consider, for example, a DCJ applied to genome $\{(\circ\ c_1\ -a_1\ d_1\ \circ), (\circ\ b_1\ -a_2\ c_2\ \circ)\}$, that cuts the first chromosome before and after $-a_1\ d_1$, creating the segments $(\circ\ c_1\ \bullet)$, $(\bullet\ -a_1\ d_1\ \bullet)$ and $(\bullet\ \circ)$ (the symbol $\bullet$ represents the open ends). If we then join the first with the third and the second with the fourth open end, we obtain $\{(\circ\ c_1\ -d_1\ a_1\ \circ), (\circ\ b_1\ -a_2\ c_2\ \circ)\}$. This DCJ corresponds to the inversion of contiguous genes $-a_1\ d_1$. DCJ operations can also correspond to other rearrangements, such as translocations, fusions and fissions [15].

### 2.3 DCJ Distance and Adjacency Graph

Observe that the DCJ operation alone can only sort balanced genomes. We formally define the DCJ distance problem:

> **Problem** DCJ-DISTANCE$(A, B)$: Given two balanced genomes $A$ and $B$, compute their DCJ distance $\mathrm{d}_{\text{DCJ}}(A, B)$, i.e., the minimum number of DCJ operations required to transform $A$ into $B'$, such that $B' \in [B]$.

Any sequence of $\mathrm{d}_{\text{DCJ}}(A, B)$ DCJ operations transforming $A$ into $B' \in [B]$ is called an *optimal* sequence of DCJ operations.

Given two balanced genomes $A$ and $B$, DCJ-DISTANCE$(A, B)$ can be computed with the help of the following concepts. First note that, since a gene $g$ has an orientation, we can distinguish its two ends, also called its *extremities*, and denote them by $g^t$ (*tail*) and $g^h$ (*head*). An *adjacency* in a genome either is *telomeric* and corresponds to the extremity of a

gene that is adjacent to one of its chromosome ends, or it is an unordered pair of consecutive extremities in one of its chromosomes. Thus, a genome $A$ can also be defined as a set of adjacencies adj($A$) of its numbered genes. Given genome $A = \{(\circ\ c_1\ -a_1\ d_1\ \circ), (\circ\ b_1\ -a_2\ c_2\ \circ)\}$, for example, we have adj($A$) = { $c_1^t$ , $c_1^h a_1^h$ , $a_1^t d_1^t$ , $d_1^h$ , $b_1^t$ , $b_1^h a_2^h$ , $a_2^t c_2^t$ , $c_2^h$ }.

Given two balanced genomes $A$ and $B$, the *adjacency graph* $AG(A, B)$ [4] is a bipartite multigraph such that each partition corresponds to the set of adjacencies of one of the two input genomes, and an edge connects the same extremities of adjacencies in both partitions, regardless of their index numbers. We say that the edge *represents* those extremities. The *length* of a path or cycle in $AG(A, B)$ is the number of edges it contains.

**Without Duplicate Genes** When the genomes $A$ and $B$ contain no duplicate genes, there is a one-to-one correspondence between the set of edges in $AG(A, B)$ and the set of gene extremities. In this case, vertices have degree one or two and thus the adjacency graph is a collection of disjoint paths and cycles. Here, problem DCJ-DISTANCE can easily be solved in linear time [4] using the formula

$$\mathrm{d}_{\text{DCJ}}(A, B) = n - c - i/2 \ ,$$

where $n = |\mathcal{G}(A)| = |\mathcal{G}(B)|$ is the number of genes in any of the two genomes, $c$ is the number of cycles and $i$ is the number of odd-length paths in $AG(A, B)$.

**With Duplicate Genes** When genomes have duplicate genes, problem DCJ-DISTANCE becomes NP-hard [13]. In the same paper, the authors present an exact, exponential-time algorithm for its solution, phrased in form of an Integer Linear Program (ILP).

## 3 An Approach to Compute the DCJ Distance with Duplicate Genes

Observe that in the presence of duplicate genes, the adjacency graph may contain vertices of degree larger than two. A *decomposition* of $AG(A, B)$ is a collection of vertex-disjoint cycles and paths covering all vertices of $AG(A, B)$. Cycles and paths of a decomposition $D$ are collectively called *components* of $D$.

There can be multiple ways of selecting a decomposition of the adjacency graph. We need to find one that allows to match each occurrence of a gene in genome $A$ with exactly one occurrence of the same gene in genome $B$. In order to build such a decomposition, we need the following definitions.

Let $g_i$ and $g_j$ be, respectively, occurrences of the same gene $g$ in genomes $A$ and $B$. The edge $e$ that represents the connection of the head of $g_i$ to the head of $g_j$ and the edge $f$ that represents the connection of the tail of $g_i$ to the tail of $g_j$ are called *siblings*. Two edges are *compatible* if they are siblings, or they represent the connection of extremities of distinct occurrences of the same gene, or they represent the connection of extremities of distinct genes. Otherwise they are *incompatible*. A set of edges is *compatible* if it has no pair of

incompatible edges. A path or cycle $C$ of $AG(A, B)$ is *consistent* if the set $E(C)$ of edges of $C$ is compatible. Note that, when constructing a decomposition, by choosing consistent components one may still select incompatible edges that occur in separate components (see the three dotted cycles of length 2 in Fig. 1). Thus, consistency cannot be taken into account in components separately.

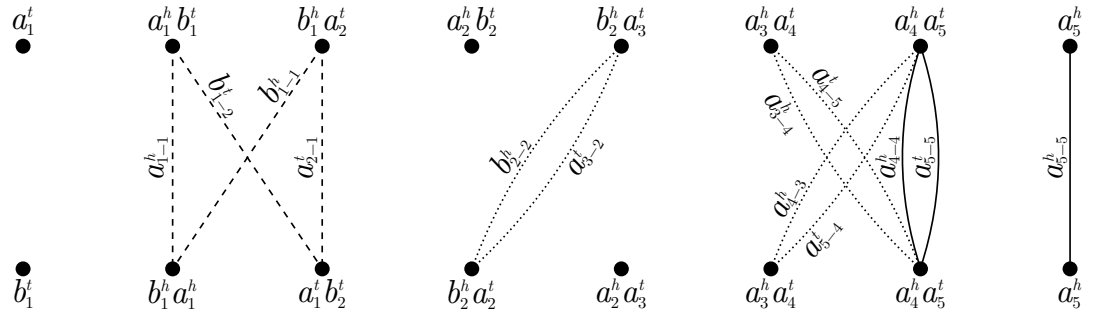

Fig. 1: Examples of an inconsistent cycle (dashed edges) and an inconsistent set of cycles (dotted edges): the adjacency graph for $A = (\circ\ a_1\ b_1\ a_2\ b_2\ a_3\ a_4\ a_5\ \circ)$ and $B = (\circ\ b_1\ -a_1\ b_2\ a_2\ a_3\ a_4\ a_5\ \circ)$, with some edges omitted. For the sake of clarity, edges are labeled with extremities they represent. For example, an edge labeled $g^t_{i-j}$ represents extremities $g^t_i$ from $A$ and $g^t_j$ from $B$.

A set of paths and cycles $\{C_1, C_2, \ldots, C_k\}$ of $AG(A, B)$ is *consistent* if and only if $E(C_1) \cup E(C_2) \cup \cdots \cup E(C_k)$ is compatible. A *consistent decomposition* $D$ of $AG(A, B)$ is a consistent set of vertex-disjoint cycles and paths that cover all vertices in $AG(A, B)$. Observe that in a consistent decomposition $D$ we have only pairs of siblings, i.e., either an edge $e$ and its sibling $f$ are in $D$ or both $e$ and $f$ are not in $D$. Thus, a consistent decomposition corresponds to a matching of occurrences of genes in both genomes and allows us to compute the value

$$d_D = n - c_D - i_D/2 \ ,$$

where $n = |\mathcal{G}^N(A)| = |\mathcal{G}^N(B)|$ and $c_D$ and $i_D$ are the numbers of cycles and odd-length paths in $D$, respectively. This provides a way to compute the DCJ distance.

**Theorem 1.** *Given two genomes $A$ and $B$, possibly with duplicate genes, the solution for the problem* DCJ-DISTANCE *is given by the following equation:*

$$\mathrm{d}_{\text{DCJ}}(A, B) = \min_{D \in \mathcal{D}} \{d_D\} \ ,$$

*where $\mathcal{D}$ is the set of all consistent decompositions of $AG(A, B)$.*

*Proof.* Since a consistent decomposition allows to match duplicates in both genomes, clearly $\mathrm{d}_{\text{DCJ}}(A, B) \leq \min_{D \in \mathcal{D}} \{d_D\}$. Now, assume that $\mathrm{d}_{\text{DCJ}}(A, B) < \min_{D \in \mathcal{D}} \{d_D\}$. By definition, this

distance corresponds to an optimal rearrangement scenario from $A$ to some $B' \in [B]$ and therefore implies a matching between the genes of $A$ and the genes of $B'$. Furthermore, this matching gives rise to a consistent decomposition $D'$ of $AG(A, B)$ such that $d_{D'} < \min_{D \in \mathcal{D}}\{d_D\}$, which is a contradiction. □

A consistent decomposition $D$ such that $d_D = \mathrm{d}_{\text{DCJ}}(A, B)$ is said to be *optimal*.

Once a consistent decomposition $D$ of the adjacency graph $AG(A, B)$ is found, following [4] it is easy to derive in linear time a DCJ rearrangement scenario with $d_D$ DCJ operations transforming $A$ into $B$. Moreover, an optimal consistent decomposition allows to find all optimal rearrangement scenarios [5].

### 3.1 Capping Telomeres

A general technique for simplifying algorithms that handle genomes with possibly unequal telomeric adjacencies is called *capping* and consists of transforming each telomeric into a non-telomeric adjacency [9, 12, 16]. Let *null extremities* be represented by $\tau$ and *null adjacencies* be represented by $\tau\tau$. Given two genomes $A$ and $B$ with $2i$ and $2j$ telomeres, respectively, in both genomes each telomeric adjacency $x$ is replaced by the adjacency $x\tau$. Furthermore, in order to add the same number of null extremities to both genomes, $|j - i|$ null adjacencies $\tau\tau$ are added to genome $A$, if $i < j$, or to genome $B$, if $j < i$. Let $A_\tau$ and $B_\tau$ be the new sets of adjacencies obtained by this procedure. Observe that in $AG(A_\tau, B_\tau)$ each null extremity of $A_\tau$ must be connected to each null extremity of $B_\tau$.

Observe that any consistent decomposition $D$ of $AG(A_\tau, B_\tau)$ is composed of cycles only, allowing to compute the value

$$d_D = n - c_D \ ,$$

where $n = |\mathcal{G}^N(A)| = |\mathcal{G}^N(B)|$ and $c_D$ is the number of cycles in $D$.

**Theorem 2.** *Let $A$ and $B$ be two genomes and let $A_\tau$ and $B_\tau$ be the genomes obtained from $A$ and $B$ by capping telomeric adjacencies. Then,*

$$\mathrm{d}_{\text{DCJ}}(A, B) = \min_{D \in \mathcal{D}_\tau}\{d_D\} \ ,$$

*where $\mathcal{D}_\tau$ is the set of all consistent decompositions of $AG(A_\tau, B_\tau)$.*

*Proof.* Each consistent decomposition $D$ of $AG(A, B)$ corresponds to a consistent decomposition $D'$ of $AG(A_\tau, B_\tau)$, such that each path in $D$ becomes a cycle in $D'$. The null extremities added to both genomes ensure that $d_D = d_{D'}$: in the formula to compute $d_{D'}$ each path adds one to the term $c_D$ but (i) each even path has two new null extremities and adds one to the term $n$ and (ii) each pair of odd paths has two new null extremities and adds one to the term $n$ and decreases two from the term $i_D$. □

## 4 Approximating the DCJ Distance by Cycles of Length 2

All definitions and properties for the DCJ distance for balanced genomes presented from the beginning to here work properly for the general case, where genomes are multichromosomal. However, as we will see in this section, to solve the DCJ distance problem we use an intermediate procedure whose inputs are strings. Thus, from now on, we restrict our inputs for linear unichromosomal genomes. The extension to general genomes is left as an open problem.

As mentioned in the previous section, the adjacency graph for balanced and capped genomes is a collection of cycles and thus we have to find a disjoint cycle decomposition of the adjacency graph to compute the DCJ distance according to Theorems 1 and 2. Recall that it is an NP-hard problem [13].

Given a consistent decomposition $D \in \mathcal{D}_\tau$ of an adjacency graph $AG(A_\tau, B_\tau)$, we can see that

$$d_D = n - c_D = n - c_2 - c_> \ ,$$

where $n = |\mathcal{G}^N(A)| = |\mathcal{G}^N(B)|$, $c_2$ is the number of cycles of length 2, and $c_>$ is the number of cycles of length longer than 2 in $D$. A naive approach to solve the DCJ distance problem could be, as a first step, maximizing $c_2$. However, this strategy is not able to solve properly the DCJ distance problem for two main reasons: (i) finding the maximum number of cycles of length 2 is itself an NP-hard problem, as we will justify below; and (ii) this strategy is not optimal to solve the DCJ distance, as we can see in Fig. 2.

The problem of finding a decomposition maximizing the number of cycles of length 2 is equivalent to the *adjacency similarity problem* [3], the complement of the breakpoint distance problem, where one wants to minimize $n - c_2$. Moreover, from an optimal solution for the adjacency similarity (or the breakpoint distance) problem it is possible to approximate the DCJ distance, as stated in Lemma 1.

**Lemma 1.** *A consistent decomposition $D'$ of $AG(A_\tau, B_\tau)$ containing the maximum number of cycles of length 2 is a 2-approximation for the* DCJ-DISTANCE *problem.*

*Proof.* Let $c_2^*$ and $c_>^*$ be the number of cycles of length 2 and longer than 2, respectively, of an optimal decomposition $D^*$ of $AG(A_\tau, B_\tau)$. Let $c_2'$ and $c_>'$ be the numbers analogous to $c_2^*$ and $c_>^*$ with respect to the decomposition $D'$. It it easy to see that $c_2^* + 2c_>^* \leq n$, then

$$\begin{aligned} 0 &\leq n - c_2^* - 2c_>^* \\ n - c_2^* &\leq n - c_2^* - 2c_>^* + n - c_2^* \\ n - c_2^* &\leq 2(n - c_2^* - c_>^*) \ . \end{aligned} \tag{1}$$

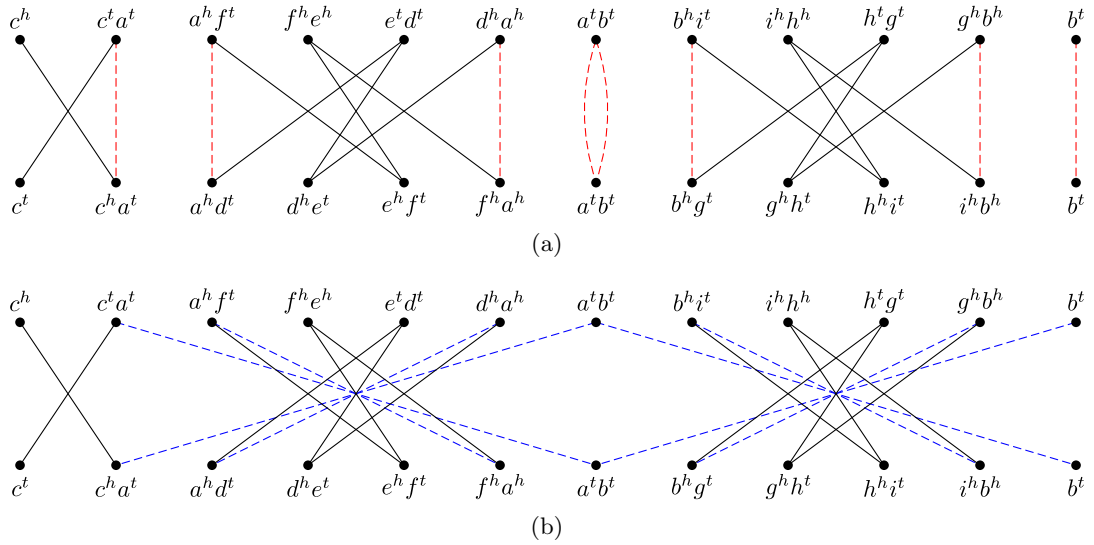


Fig. 2: Consistent decompositions for genomes $A = (\circ\ {-c}\ a\ f\ {-e}\ d\ {-a}\ b\ i\ {-h}\ g\ {-b}\ \circ)$ and $B = (\circ\ c\ a\ d\ e\ f\ {-a}\ b\ g\ h\ i\ {-b}\ \circ)$, where solid edges are in both decompositions. Gene indexes were omitted. (a) A consistent decomposition $D'$ containing the maximum number of cycles of length 2, composed of 1 cycle of length 2, 1 cycle of length 8, 1 odd path of length 1 and 1 odd path of length 3, resulting in $d_{D'} = 11 - 4 - 2/2 = 7$. (b) An optimal consistent decomposition $D^*$, composed of 4 cycles of length 4 and 2 odd paths of length 3, resulting in $d_{D^*} = 11 - 4 - 2/2 = 6$.

Therefore, we have

$$\begin{aligned}
\frac{d_{D'}}{d_{D^*}} &= \frac{n - c'_2 - c'_>}{n - c^*_2 - c^*_>} \\
&\leq \frac{n - c^*_2 - c'_>}{n - c^*_2 - c^*_>} && (2) \\
&\leq \frac{n - c^*_2}{n - c^*_2 - c^*_>} \\
&\leq \frac{2(n - c^*_2 - c^*_>)}{n - c^*_2 - c^*_>} && (3) \\
&= 2\ , && (4)
\end{aligned}$$

where (2) holds since $c'_2 \geq c^*_2$, and (3) is true from (1). □

Recall that the adjacency similarity and breakpoint distance problems are both NP-hard [3, 6]. The former can be approximated by a factor of 4 for balanced genomes [1]. However, an approximation with constant approximation factor for the former problem

does not lead to an approximation with constant approximation factor for the latter. The breakpoint distance for balanced genomes has a 1.1037-approximation when $occ = 2$ [8], a 4-approximation when $occ = 3$ [8], and an $O(k)$-approximation when $occ = k$ [11]. Those approximations were developed for the minimum common string partition problem (MCSP) [14], which is equivalent to the breakpoint distance problem [10].

## 5 Finding Consistent Decompositions

In this section we present a linear time approximation algorithm Consistent-Decomposition, which receives two linear unichromosomal balanced genomes $A$ and $B$ with $occ = k$ and returns a consistent decomposition for genomes $A$ and $B$, which is an $O(k)$-approximation for the DCJ distance. The main steps of Consistent-Decomposition can be briefly described as follows. First, from the input genomes, we obtain capped genomes and then we build the adjacency graph of them. Next, we use an approximation for the (signed) minimum common string partition problem, which gives an approximation for the number of breakpoints in the adjacency graph. After that we clean the chosen cycles of length 2 from the adjacency graph. Following, we iteratively collect arbitrary cycles of length longer than 2, cleaning up the remaining graph after each iteration. Finally, we return the set of collected cycles as a consistent decomposition of the prior adjacency graph.

**Algorithm 1** Consistent-Decomposition$(A, B)$

**Input:** balanced genomes $A$ and $B$ such that $occ = k$
**Output:** a consistent decomposition of $AG(A, B)$
1: Add null extremities/adjacencies to $A$ and $B$ and obtain $A_\tau$ and $B_\tau$, respectively
2: Build the adjacency graph $AG(A_\tau, B_\tau)$
3: Obtain an $O(k)$-approximation $\mathcal{S}_2$ for the set of cycles of length 2 in $AG(A_\tau, B_\tau)$ using the $O(k)$-approximation algorithm for the minimum common string partition problem [11]
4: Remove from the adjacency graph vertices covered by $\mathcal{S}_2$ and all edges incompatible with edges of $\mathcal{S}_2$
5: Decompose the remaining graph into consistent cycles by iteratively finding a consistent cycle $C$ and then removing from the graph vertices covered by $C$ and edges incompatible with edges of $C$, collecting them in $\mathcal{S}_>$
6: Remove null extremities/adjacencies of cycles in $\mathcal{S}_2 \cup \mathcal{S}_>$ and obtain a consistent decomposition $D$ of $AG(A, B)$
7: Return $D$

Step 1 of Consistent-Decomposition consists of capping telomeres from the given balanced genomes $A$ and $B$ as described in Section 3.1. In Step 2, Consistent-Decomposition builds the adjacency graph for capped genomes $A_\tau$ and $B_\tau$. After that, Step 3 collects cycles of length 2 using an $O(k)$-approximation algorithm for the minimum common string partition problem [11] as described in Section 4. Step 4 removes from $AG(A_\tau, B_\tau)$ vertices covered by cycles in $\mathcal{S}_2$ and edges incompatible with edges of cycles in $\mathcal{S}_2$. Step 5

constructs the set $\mathcal{S}_>$ by decomposing the remaining graph into consistent cycles. Iteratively, it chooses a consistent cycle $C$ and then removes from the remaining graph vertices covered by $C$ and edges incompatible with edges of $C$. Hence the algorithm does not choose an inconsistent set of components. Further, this guarantees that for every edge in the decomposition, its sibling edge will also be in the decomposition, avoiding for example the selection of the path of length 1 composed of the edge that connects $a_5^h$ of $A$ to $a_5^h$ of $B$ and then the cycle of length 2 composed of the edge that connects $a_4^h$ of $A$ to $a_3^h$ of $B$ and the edge that connects $a_5^t$ of $A$ to $a_4^t$ of $B$ in Fig. 1. In order to obtain the consistent decomposition of $AG(A, B)$, CONSISTENT-DECOMPOSITION removes in Step 6 null extremities/adjacencies of cycles in $\mathcal{S}_2 \cup \mathcal{S}_>$, returning the resulting set $D$ in Step 7.

There is one implicit but important step in the algorithm above, which is to obtain the set $\mathcal{S}_2$ given the output of the $k$-MCSP approximation algorithm [11]. This algorithm outputs a common string partition $(\mathcal{A}, \mathcal{B})$. Both $\mathcal{A}$ and $\mathcal{B}$ are composed of the same set of substrings, in different orders and possibly reversed. Each substring of length $l > 1$ in $\mathcal{A}$ and $\mathcal{B}$ induces $l - 1$ preserved adjacencies in $A$ and $B$. First of all, we must normalize strings in $\mathcal{A}$ and $\mathcal{B}$, that is, for each substring $s$ and its reverse $r$, only $s$ appears in $\mathcal{A}$ and $\mathcal{B}$ (we reverse each occurrence of $r$, resulting in $s$). Then we just have to map each substring in $\mathcal{A}$ to the same substring in $\mathcal{B}$ (in case of multiple occurrences, we choose any of them), which can be performed using a prefix tree. Thus this implicit step can be done in linear time on the adjacency graph size.

**Lemma 2.** *Given balanced genomes $A$ and $B$ such that $|A| = |B|$, the running time of* CONSISTENT-DECOMPOSITION *algorithm is linear on the size of the corresponding adjacency graph.*

*Proof.* Let $m$ be the size of $AG(A_\tau, B_\tau)$. It is easy to see that Steps 1 and 2 of Algorithm 1 have both linear running time, i.e. $O(m)$. The implementation of the $k$-MCSP [11] in Step 3 with suffix trees [7] and disjoint sets has running time $O(m)$ (note that $m = O(n^2)$). The running time of Step 4 is $O(m)$ since we have just to traverse vertices and edges of the remaining adjacency graph. Step 5 consists of collecting cycles arbitrarily, and therefore its running time is also linear, we just have to walk in $AG(A_\tau, B_\tau)$ finding cycles. To be sure we walk only in consistent paths, we can use a hash table of size $\Theta(n)$ and store, for each edge of previously chosen cycles in Steps 1 to 5, genes of $A_\tau$ ($B_\tau$) associated to genes of $B_\tau$ ($A_\tau$). For instance, the selection of an edge representing the connection of extremities $a_i^t$ of $A_\tau$ and $a_j^t$ of $B_\tau$ is consistent if both $a_i$ and $a_j$ are associated with no gene of $B_\tau$ and $A_\tau$, respectively, or both are already associated with each other (this edge is the sibling of a previously chosen edge). This consistency check takes $O(1)$ time. The last step (Step 6) is similar to Step 4 and thus has running time $O(m)$. Therefore, CONSISTENT-DECOMPOSITION has running time $O(m)$. □

To conclude this section, we present the following result which establishes an approximation factor for DCJ-DISTANCE.

**Theorem 3.** *Let $A$ and $B$ be balanced genomes such that $occ = k$. Given a common string partition $(\mathcal{A}, \mathcal{B})$ with approximation factor $O(k)$ for the $k$-MCSP problem, a consistent decomposition $D$ of $AG(A, B)$, containing cycles of length 2 reflecting preserved adjacencies in $(\mathcal{A}, \mathcal{B})$, is an $O(k)$-approximation for the* DCJ-DISTANCE *problem.*

*Proof.* Let $c_2^*$ and $c_>^*$ be the number of cycles of length 2 and longer than 2, respectively, of an optimal decomposition $D^*$ of $AG(A, B)$. Let $\mathcal{S}_2$ be the set of cycles of length 2 reflecting preserved adjacencies in $(\mathcal{A}, \mathcal{B})$, and let $\mathcal{S}_>$ be an arbitrary decomposition of cycles in $AG(A, B) \setminus \mathcal{S}_2$. Let $D = \mathcal{S}_2 \cup \mathcal{S}_>$, a consistent decomposition, $c_2 = |\mathcal{S}_2|$, and $c_> = |\mathcal{S}_>|$. From [11] we have an $O(k)$-approximation for the $k$-MCSP problem, and then $n - c_2 \leq \ell(n - c_2')$, where $\ell = O(k)$ and $c_2'$ is the number of cycles of length 2 in a consistent decomposition $D'$ with maximum number of cycles of length 2. Hence,

$$\begin{aligned}
\frac{d_D}{d_{D^*}} &= \frac{n - c_2 - c_>}{n - c_2^* - c_>^*} \\
&\leq \frac{\ell\,(n - c_2') - c_>}{n - c_2^* - c_>^*} \\
&\leq \frac{\ell\,(n - c_2')}{n - c_2^* - c_>^*} \\
&\leq 2\ell \left( \frac{n - c_2' - c_>'}{n - c_2^* - c_>^*} \right) \qquad (5) \\
&\leq 4\ell \ , \qquad (6)
\end{aligned}$$

where (5) is analogous to (1) and (6) holds from (4), both in the proof of Lemma 1. □

## 6 Experimental Results

We have implemented our approximation algorithm in C++, with the addition of a linear time greedy heuristic for the decomposition of cycles not induced by the $k$-MCSP approximation. The experiments for this approach were performed on an Intel i3 3.3GHz machine.

We compare our algorithm with Shao *et al.*'s ILP [13] on simulated datasets. Given two genomes, the ILP based experiments first build the adjacency graph, followed by capping of the telomeres, fixing some safe cycles of length two, and finally invoking an ILP solver to obtain an optimal solution with a time limit of 2 hours.

Following [13], we simulate artificial genomes with segmental duplications and DCJs. We uniformly select a position to start duplicating a segment of the genome and place the new copy to a new position. From a genome of $s$ distinct genes, we generate an ancestor genome with $1.5s$ genes by randomly performing $s/2l$ segmental duplications of length $l$, resulting in an average $k = 1.5$. Then we simulate two extant genomes from the ancestor by randomly performing $r$ DCJs (reversals) independently. Thus, the simulated

evolutionary distance between the two extant genomes is $2r$. We set $s = 1000$ and test three different lengths for segmental duplications ($l = 1, 2, 5$). We also vary the $r$ value over a range $200, 220, \ldots, 500$. Figure 3 shows the average difference "*computed number of DCJs − simulated evolutionary distance*", taking as input five pairs of genomes for each combination of $l$ and $r$, while Fig. 4 shows the average running time. Note that, although the DCJ distance is unknown whenever the ILP solver is stopped after the time limit, it is always less than or equal to the simulated evolutionary distance for these artificial genome pairs.

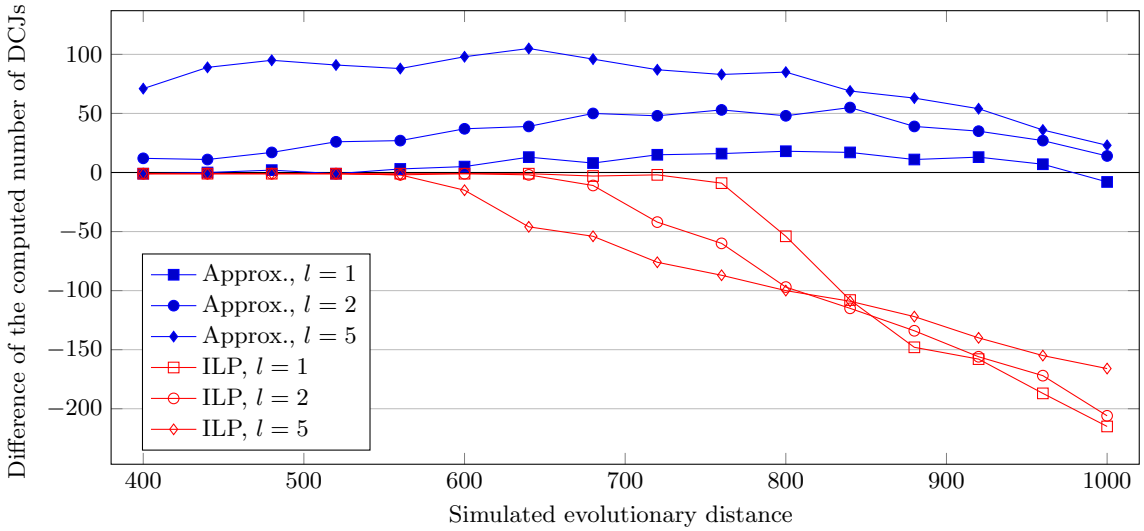


Fig. 3: The computed number of DCJs vs. the simulated evolutionary distance for $s = 1000$.

The difference of the number of DCJs (blue lines in Fig. 3) calculated by our approximation algorithm remains very close to the simulated evolutionary distance for small values of $l$. Moreover, it remains roughly the same for the same value of $l$ even for greater values of $r$. The values obtained by the ILP approach (red lines in Fig. 3) are very close to those obtained by the approximation algorithm and to the simulated evolutionary distance from the simulations for $l \leq 2$ and smaller values of $r$. However, beyond some point the DCJ distance calculated by the ILP gets even lower than the simulated evolutionary distance in the simulations, showing the limitations of parsimony for larger distance ranges.

Regarding the running time, our implementation time increases slowly from $\approx 0.03$ ($2r = 400$) to $\approx 0.08$ seconds ($2r = 1000$), on average, according to Fig. 4(a), while the ILP approach takes $\approx 0.3$ seconds to finish for smaller values of $r$ (where the preprocessing step fixes a considerable amount of cycles of length 2 in the adjacency graph), always reaching the time limit of 2 hours beyond some point, as displayed in Fig. 4(b).

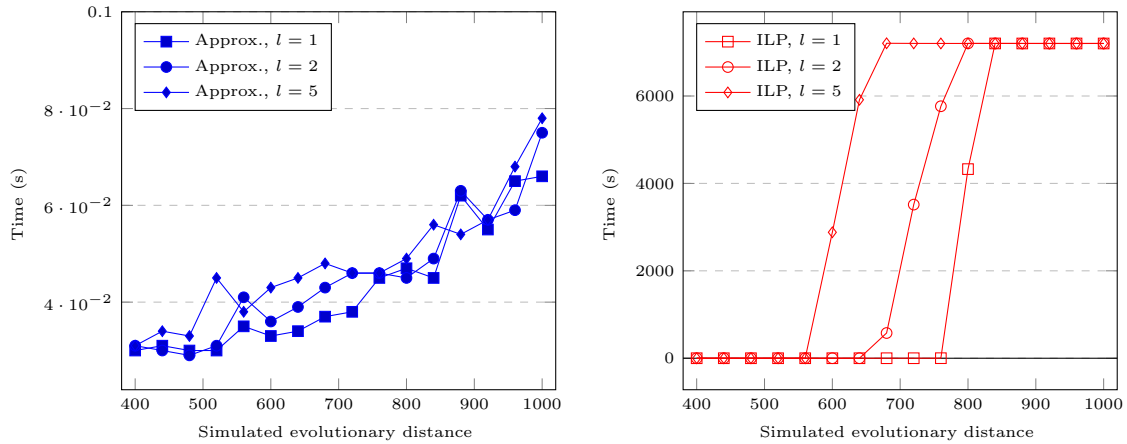


Fig. 4: Execution time for $s = 1000$ of (a) approximation and (b) ILP based programs.

## 7 Conclusion

In this paper, we have proposed a new approximation algorithm for the DCJ distance for genomes where each gene occurs the same number of times in each input genome and there exists at least one gene that occurs more than once in one of them. This so called DCJ distance with duplicates for balanced genomes problem is NP-hard [13]. Our algorithm works on input genomes where the amount of duplicates is bounded by $O(k)$, where $k$ is the maximum number of duplicates of any gene in the input genomes. The approximation factor of our algorithm is $O(k)$. Furthermore, our algorithm has linear running time on the adjacency graph size. As experiments on simulated genomes have shown, our algorithm is very competitive both in efficiency and quality of the solutions, in comparison to the ILP exact solution.

Due to an intermediate step which approximates the minimum common string partition problem, our algorithm works properly only on linear unichromosomal genomes as input. A natural extension of this work is modifying our algorithm to work with multichromosomal genomes as well. Moreover, we have to extend our experiments, running our algorithm on more simulated data sets and also on biological data sets.

## References

1. Angibaud, S., Fertin, G., Rusu, I., Thévenin, A., Vialette, S.: Efficient tools for computing the number of breakpoints and the number of adjacencies between two genomes with duplicate genes. Journal of Computational Biology 15(8), 1093–1115 (2008)
2. Angibaud, S., Fertin, G., Rusu, I., Thévenin, A., Vialette, S.: On the approximability of comparing genomes with duplicates. Journal of Graph Algorithms and Applications 13(1), 19–53 (2009)

3. Angibaud, S., Fertin, G., Rusu, I., Vialette, S.: A pseudo-boolean framework for computing rearrangement distances between genomes with duplicates. Journal of Computational Biology 14(4), 379–393 (2007)
4. Bergeron, A., Mixtacki, J., Stoye, J.: A unifying view of genome rearrangements. In: Proc. of WABI 2006. LNBI, vol. 4175, pp. 163–173 (2006)
5. Braga, M.D.V., Stoye, J.: The solution space of sorting by DCJ. J. Comp. Biol. 17(9), 1145–1165 (2010)
6. Bryant, D.: The complexity of calculating exemplar distances. In: Sankoff, D., Nadeau, J.H. (eds.) Comparative Genomics, pp. 207–211. Kluwer Academic Publishers, Dortrecht (2000)
7. Farach, M.: Optimal suffix tree construction with large alphabets. In: Proc. of IEEE/FOCS 1997. pp. 137–143 (1997)
8. Goldstein, A., Kolman, P., Zheng, J.: Minimum common string partition problem: Hardness and approximations. Eletronic Journal of Combinatorics 12(R50) (2005)
9. Hannenhalli, S., Pevzner, P.: Transforming men into mice (polynomial algorithm for genomic distance problem). In: Proc. of FOCS 1995. pp. 581–592 (1995)
10. Jiang, H., Zheng, C., Sankoff, D., Zhu, B.: Scaffold filling under the breakpoint distance. In: Proc. of RECOMB-CG 2010. Lecture Notes on Bioinformatics, vol. 6398, pp. 83–92 (2010)
11. Kolman, P., Waleń, T.: Reversal distance for strings with duplicates: Linear time approximation using hitting set. The Electronic Journal of Combinatorics 14(1), R50 (2007)
12. Shao, M., Lin, Y.: Approximating the edit distance for genomes with duplicate genes under DCJ, insertion and deletion. BMC Bioinformatics 13(Suppl 19), S13 (2012)
13. Shao, M., Lin, Y., Moret, B.: An exact algorithm to compute the double-cut-and-join distance for genomes with duplicate genes. Journal of Computational Biology 22(5), 425–435 (2015)
14. Swenson, K., Marron, M., Earnest-DeYong, K., Moret, B.M.E.: Approximating the true evolutionary distance between two genomes. In: Proc. of ALENEX/ANALCO 2005. pp. 121–129 (2005)
15. Yancopoulos, S., Attie, O., Friedberg, R.: Efficient sorting of genomic permutations by translocation, inversion and block interchanges. Bioinformatics 21(16), 3340–3346 (2005)
16. Yancopoulos, S., Friedberg, R.: DCJ path formulation for genome transformations which include insertions, deletions, and duplications. Journal of Computational Biology 16(10), 1311–1338 (2009)