JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 18, Number 9, 2011

Pp. 1167–1184
DOI: 10.1089/cmb.2011.0118

Double Cut and Join with Insertions and Deletions

MARíLIA D.V. BRAGA, EYLA WILLING, and JENS STOYE

ABSTRACT

Many approaches to compute the genomic distance are still limited to genomes with the same content, without duplicated markers. However, differences in the gene content are frequently observed and can reflect important evolutionary aspects. While duplicated markers can hardly be handled by exact models, when duplicated markers are not allowed, a few polynomial time algorithms that include genome rearrangements, insertions and deletions were already proposed. In an attempt to improve these results, in the present work we give the first linear time algorithm to compute the distance between two multichromosomal genomes with unequal content, but without duplicated markers, considering insertions, deletions and double cut and join (DCJ) operations. We derive from this approach algorithms to sort one genome into another one also using DCJ operations, insertions and deletions. The optimal sorting scenarios can have different compositions and we compare two types of sorting scenarios: one that maximizes and one that minimizes the number of DCJ operations with respect to the number of insertions and deletions. We also show that, although the triangle inequality can be disrupted in the proposed genomic distance, it is possible to correct this problem adopting a surcharge on the number of non-common markers. We use our method to analyze six species of Rickettsia, a group of obligate intracellular parasites, and identify preliminary evidence of clusters of deletions.

Key words: algorithms, comparative genomics, DCJ, evolution, indels.

1. INTRODUCTION

THE DISTANCE BETWEEN TWO GENOMES IS OFTEN COMPUTED using only the common markers, that occur in both genomes (Hannenhalli and Pevzner, 1995; Yancopoulos et al., 2005; Bergeron et al., 2006; Braga and Stoye, 2010). However, genomes with the same content are rare, and differences in gene content may reflect important evolutionary aspects. Interesting examples are bacteria which are obligate intracellular parasites. The genomes of such bacteria are observed to have a reductive evolution, that is the process by which genomes shrink and undergo extreme levels of gene degradation and loss (Andersson and Kurland, 1998). Approaches that are able to handle genomes with unequal content could give valuable hints on the evolution of such organisms.

While duplicated markers can hardly be handled by exact models (Sankoff, 1999; Bryant, 2001; Marron et al., 2004; Bader, 2010), some polynomial time approaches that are able to deal with insertions and deletions were proposed. In this context, El-Mabrouk (2001) introduced a method to compare

Technische Fakultät, Universität Bielefeld, Bielefeld, Germany.

unichromosomal genomes with unequal content, but without duplicated markers, considering inversions, insertions and deletions, such that a block of contiguous markers can be inserted or deleted at once. Two algorithms were provided, an exact one, which deals with insertions and deletions asymmetrically, and a heuristic that is able to handle all operations symmetrically.

In the present work, we develop a linear time symmetric approach to compare multichromosomal genomes with unequal content, but without duplicated markers, also allowing a block of contiguous markers to be inserted or deleted at once. In addition to insertions and deletions, we consider double cut and join (DCJ) operations, that allow us to represent most large scale rearrangements, such as inversions, translocations, fusions and fissions, that can occur in genomes (Yancopoulos et al., 2005; Bergeron et al., 2006). We borrow some ideas from a study of Yancopoulos and Friedberg (2009), but in our model we provide a clear separation between insertions, deletions and DCJ operations. We then derive algorithms to sort one genome into another one, showing that the optimal sorting scenarios can have different compositions with respect to the number of each type of operation and propose two types of sorting scenarios: one that minimizes the number of DCJ operations (respectively maximizes the number of insertions and deletions) and one that minimizes the number of insertions and deletions (respectively maximizes the number of DCJ operations).

We use our method to analyze six species of Rickettsia (Blanc et al., 2007), a group of obligate intracellular parasites, and are able to identify preliminary evidence of clusters of deletions in the analyzed genomes. Furthermore, we show that the triangle inequality can be disrupted in our model, but this problem can be corrected *a posteriori* by applying a surcharge on the number of non-common markers.

This article is an extension of the results recently presented in two subsequent works (Braga, 2010; Braga et al., 2010) and is organized as follows. In Section 2, we give the basic definitions of our model, and in Section 3, we develop the indel-potential of two genomes, a property that allows us to obtain a good upper bound to the genomic distance with DCJ and indel operations. Then, in Section 4, we obtain an exact formula for the distance, also showing that it can be computed in linear time, while in Section 5, we derive algorithms to sort genomes with DCJ and indel operations. In Section 6, we apply our method to analyze the evolution of Rickettsia. In Section 7, we propose a surcharge to establish the triangle inequality, and, finally, in Section 8 we summarize our results.

2. DEFINITIONS

We analyze genomes with unequal content but without duplicated markers. Each genome is possibly composed of linear and circular chromosomes and can be represented by a set of strings as follows. From each chromosome $\mathcal{C}$ of each genome, we can build a string s, obtained by the concatenation of all markers in $\mathcal{C}$, read in any of the two directions. Each marker g is a DNA fragment and is represented by the symbol g, if it is read in direct orientation, or by the symbol $\overline{g}$, if it is read in reverse orientation. Each end of a linear chromosome is called a *telomere*, represented by the symbol $\circ$. Thus, if $\mathcal{C}$ is linear, it is represented by $\circ s\circ$. If $\mathcal{C}$ is circular, it is simply represented by s (we can start to build s in any symbol of $\mathcal{C}$).

Given two different genomes A and B, we denote by $\mathcal{G}$ the "reduced" genome (El-Mabrouk, 2001), that is the set of markers that occur once in A and once in B. Moreover, the set $\mathcal{A}$ contains the markers that occur only in A and the set $\mathcal{B}$ contains the markers that occur only in B. Observe that the sets $\mathcal{G}$, $\mathcal{A}$ and $\mathcal{B}$ are disjoint. An example of a pair of genomes is given in Figure 1.

Before introducing the *indel* operation, in the following three subsections we give some important concepts of the DCJ model, that are generalizations of definitions introduced by Bergeron et al. (2006).

2.1. $\mathcal{G}$-adjacencies

Given two different genomes A and B, recall that $\mathcal{G}$ is the set of markers that occur once in A and once in B. For each marker $g \in \mathcal{G}$, denote its two extremities by g^t (tail) and g^h (head). Then, a *$\mathcal{G}$-adjacency* in genome A (respectively in genome B) is in general a linear string $v = \gamma_1 \ell \gamma_2$, such that γ_1 and γ_2 are telomeres or extremities of markers of $\mathcal{G}$ and ℓ, the substring composed of the markers that are between γ_1 and γ_2 in A (respectively in B), contains no marker that also belongs to $\mathcal{G}$. The substring ℓ is said to be the *label* of v, and the extremities γ_1 and γ_2 are said to be *$\mathcal{G}$-adjacent*. If ℓ is a non-empty string, v is said to be *labeled*, otherwise v is said to be *clean*.

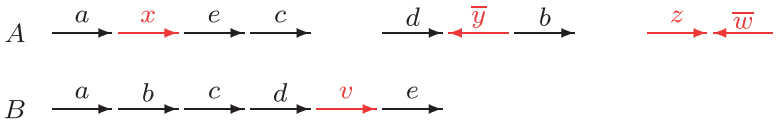

FIG. 1. (A, B) For genomes $A = \{\circ axec\circ\ ,\ \circ d\overline{y}b\circ\ ,\ \circ z\overline{w}\circ\}$, composed of three linear chromosomes, and $B = \{\circ abcdve\circ\}$, composed of one single chromosome, we have $\mathcal{G} = \{a, b, c, d, e\}$, $\mathcal{A} = \{x, y, z, w\}$ and $\mathcal{B} = \{v\}$.

Observe that a $\mathcal{G}$-adjacency $\gamma_1\ell\gamma_2$ can also be represented by $\gamma_2\overline{\ell}\gamma_1$. Moreover, a labeled $\mathcal{G}$-adjacency $v = \circ\ell\circ$ in genome A (or B) indicates that A (or B) contains a linear chromosome composed only of markers that are not in $\mathcal{G}$, that is, v corresponds to a whole linear chromosome. In the same way, a special case is a $\mathcal{G}$-adjacency $v = \ell$, that corresponds to a whole circular chromosome in A (or B), composed only of markers that are not in $\mathcal{G}$. This is the only case of a $\mathcal{G}$-adjacency in which we have a circular instead of a linear string.

Two genomes A and B can then be represented by the sets $V_{\mathcal{G}}(A)$ and $V_{\mathcal{G}}(B)$, containing their $\mathcal{G}$-adjacencies. For the genomes in Figure 1, we have $\mathcal{G} = \{a, b, c, d, e\}$, $V_{\mathcal{G}}(A) = \{\circ a^t, a^h x e^t, e^h c^t, c^h\circ\ ,\ \circ d^t, d^h\overline{y}b^t, b^h\circ\ ,\ \circ z\overline{w}\circ\}$ and $V_{\mathcal{G}}(B) = \{\circ a^t, a^h b^t, b^h c^t, c^h d^t, d^h v e^t, e^h\circ\}$.

2.2. The DCJ operation

A *cut* performed on a genome A separates two adjacent markers of A. A cut affects a $\mathcal{G}$-adjacency v of $V_{\mathcal{G}}(A)$ as follows: if v is linear, the cut is done between two symbols of v, creating two open ends in two separate linear strings; if v is circular, the cut creates two open ends in one linear string. A *double-cut and join* or DCJ applied on a genome A is the operation that performs two cuts in $V_{\mathcal{G}}(A)$, creating four open ends, and joins these open ends in a different way. As an example, considering the genome A from Figure 1 and $\mathcal{G} = \{a, b, c, d, e\}$, if we apply a DCJ on $a^h x e^t$ and $d^h\overline{y}b^t$ of $V_{\mathcal{G}}(A)$ we can create $a^h b^t$ and $d^h\overline{y}x e^t$.

Consider a DCJ applied on $\gamma_1\ell_1\ell_4\gamma_4$ and $\gamma_3\ell_3\ell_2\gamma_2$, that creates $\gamma_1\ell_1\ell_2\gamma_2$ and $\gamma_3\ell_3\ell_4\gamma_4$. We represent such an operation as $(\{\gamma_1\ell_1|\ell_4\gamma_4,\ \gamma_3\ell_3|\ell_2\gamma_2\} \rightarrow \{\gamma_1\ell_1|\ell_2\gamma_2,\ \gamma_3\ell_3|\ell_4\gamma_4\})$. Observe that one or more extremities among γ_1, γ_2, γ_3 and γ_4 can be equal to $\circ$ (a telomere), as well as one or more labels among ℓ_1, ℓ_2, ℓ_3 and ℓ_4 can be equal to ε (the empty string). Particular cases happen when we have circular adjacencies. A DCJ involving two $\mathcal{G}$-adjacencies such that at least one is circular always results in only one $\mathcal{G}$-adjacency: $(\{|\ell_3|,\ \gamma_1\ell_1|\ell_2\gamma_2\} \rightarrow \{\gamma_1\ell_1|\ell_3|\ell_2\gamma_2\})$ or $(\{|\ell_1|,\ |\ell_2|\} \rightarrow \{|\ell_1|\ell_2|\})$. Conversely, a DCJ operation with two cuts in one $\mathcal{G}$-adjacency always results in two $\mathcal{G}$-adjacencies, such that at least one is circular: $(\{\gamma_1\ell_1|\ell_3|\ell_2\gamma_2\} \rightarrow \{|\ell_3|,\ \gamma_1\ell_1|\ell_2\gamma_2\})$ or $(\{|\ell_1|\ell_2|\} \rightarrow \{|\ell_1|,\ |\ell_2|\})$.

2.3. Adjacency graph and the DCJ distance

The problem of sorting A into B can be studied with the help of the following graph, introduced by Bergeron et al. (2006). The *adjacency graph* $AG(A, B)$ is the bipartite graph that has a vertex for each $\mathcal{G}$-adjacency in $V_{\mathcal{G}}(A)$ and a vertex for each $\mathcal{G}$-adjacency in $V_{\mathcal{G}}(B)$. Then, for each $g \in \mathcal{G}$, we have one edge connecting the vertex in $V_{\mathcal{G}}(A)$ and the vertex in $V_{\mathcal{G}}(B)$ that contain g^h and one edge connecting the vertex in $V_{\mathcal{G}}(A)$ and the vertex in $V_{\mathcal{G}}(B)$ that contain g^t. Due to the 1-to-1 correspondence between the vertices of $AG(A, B)$ and the $\mathcal{G}$-adjacencies in $V_{\mathcal{G}}(A)$ and $V_{\mathcal{G}}(B)$, we can identify each adjacency with its corresponding vertex.

The graph $AG(A, B)$ is a collection of connected components and can have cycles and paths that alternate vertices in $V_{\mathcal{G}}(A)$ and $V_{\mathcal{G}}(B)$ (Bergeron et al., 2006). A path that has one endpoint in $V_{\mathcal{G}}(A)$ and the other in $V_{\mathcal{G}}(B)$ is called an *AB-path*. In the same way, both endpoints of an *AA-path* are in $V_{\mathcal{G}}(A)$, as well as both endpoints of a *BB-path* are in $V_{\mathcal{G}}(B)$. Furthermore, $AG(A, B)$ can have two extra types of components: each $\mathcal{G}$-adjacency that corresponds to a linear (respect. circular) chromosome is a *linear* (respect. *circular*) *singleton*. Linear singletons are particular cases of *AA*-paths and *BB*-paths. When $\mathcal{A} = \mathcal{B} = \emptyset$, the adjacency graph is composed only of clean $\mathcal{G}$-adjacencies and has no singletons. In this case, the graph is said to be *clean*. An example of an adjacency graph is given in Figure 2.

Bergeron et al. (2006) observed that the number of *AB*-paths in $AG(A, B)$ is even and that a DCJ either changes the number of cycles by one, or the number of *AB*-paths by two, or does not affect the number of cycles and *AB*-paths in the graph. Singletons, *AB*-paths composed of one single edge, and cycles composed

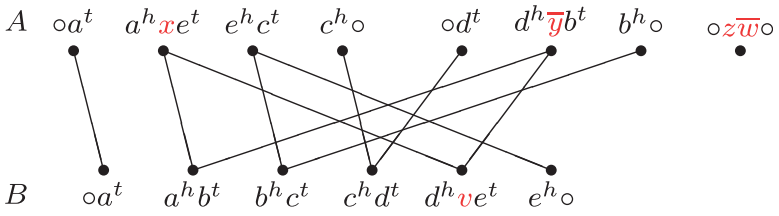

FIG. 2. (A, B) For genomes $A = \{\circ axec\circ\ ,\ \circ d\overline{y}b\circ\ ,\ \circ z\overline{w}\circ\}$ and $B = \{\circ abcdve\circ\}$, the adjacency graph contains one cycle, two *AA*-paths (one is a linear singleton) and two *AB*-paths.

of two edges are said to be *DCJ-sorted*. Longer paths and cycles are said to be *DCJ-unsorted*. The procedure of using DCJ operations to turn *AG*(*A, B*) into DCJ-sorted components is called *DCJ-sorting* of *A* into *B*. The *DCJ distance* of *A* and *B*, denoted by $d_{DCJ}(A, B)$, corresponds to the minimum number of steps required to do a DCJ-sorting of *A* into *B* and can be easily obtained:

Theorem 1 (Bergeron et al., 2006). *Given two genomes A and B without duplicated markers, we have* $d_{DCJ}(A,B) = |\mathcal{G}| - c - \frac{b}{2}$, *where* $\mathcal{G}$ *is the set of common markers between A and B, and c and b are, respectively, the number of cycles and of AB-paths in AG*(*A, B*).

A DCJ operation is *optimal* when it either increases the number of cycles by one, or the number of *AB*-paths by two (decreasing the DCJ distance by one). In the same way, a *neutral* DCJ operation does not affect the number of cycles and *AB*-paths, while a *counter-optimal* DCJ operation either decreases the number of cycles by one, or the number of *AB*-paths by two. The problem of finding an optimal sequence of operations that do a DCJ-sorting of *A* into *B* can be solved with a simple greedy linear time algorithm (Bergeron et al., 2006).

2.4. Indel operations

The markers in $\mathcal{A}$ and $\mathcal{B}$ are represented in the adjacency graph as labels and singletons, and, in order to sort genome *A* into genome *B*, in addition to the DCJ-sorting, the markers in $\mathcal{A}$ have to be deleted, while the markers in $\mathcal{B}$ have to be inserted. No classical DCJ operation is able to perform an insertion or a deletion. Moreover, no operation is able to delete and insert at the same time. Such an event would be a substitution, that is not accepted in the model we consider. An operation is thus either a DCJ, or an *insertion*, or a *deletion*. We refer to insertions and deletions as *indel* operations.

In our model, an insertion or deletion only affects the label of one single $\mathcal{G}$-adjacency (that can be a singleton), by deleting or inserting markers in this label. Given $\ell_3 \neq \varepsilon$, the deletion of ℓ_3 from the $\mathcal{G}$-adjacency $\gamma_1\ell_1\ell_3\ell_2\gamma_2$ is represented as $(\gamma_1\ell_1|\ell_3|\ell_2\gamma_2 \rightarrow \gamma_1\ell_1|\ell_2\gamma_2)$,[1] while the insertion of ℓ_3 in the $\mathcal{G}$-adjacency $\gamma_1\ell_1\ell_2\gamma_2$ is represented as $(\gamma_1\ell_1|\ell_2\gamma_2 \rightarrow \gamma_1\ell_1|\ell_3|\ell_2\gamma_2)$. One or both extremities among γ_1 and γ_2 can be equal to $\circ$(a telomere), as well as one or both labels among ℓ_1 and ℓ_2, can be equal to ε (the empty string). A deletion of $\ell_2 \neq \varepsilon$ from a circular singleton $\ell_1\ell_2$ is represented by $(|\ell_1|\ell_2| \rightarrow |\ell_1|)$ and its insertion is $(|\ell_1| \rightarrow |\ell_1|\ell_2|)$. Observe that at most one chromosome can be entirely deleted or inserted at once. Moreover, since duplications are not allowed, an insertion of a marker that already exists is not allowed. Consequently, in this model, it is not possible to apply insertions and/or deletions involving the markers in $\mathcal{G}$.

The *DCJ-indel distance* of *A* and *B*, denoted by $d^{id}_{DCJ}(A,B)$, is the minimum number of DCJ and indel operations required to transform *A* into *B*.

Note that our definition of an indel operation avoids the "free lunch problem," mentioned by Yancopoulos and Friedberg (2009), which is the possibility of transforming any genome *A* into any genome *B* by simply deleting the whole content of *A* and inserting the whole content of *B*. This would be possible only when *A* and *B* have no common markers. However, the triangle inequality can be disrupted in the DCJ-indel distance. This issue will be addressed in Section 7.

3. RUNS OF UNIQUE MARKERS AND THE INDEL-POTENTIAL

Observe that a $\mathcal{G}$-adjacency with a non-empty label ℓ can be cut in at least two different positions, either before or after ℓ. Since the position of the cut does not change the effect of the DCJ operation on $d_{DCJ}(A, B)$,

[1]For better reading in our notation we omit the curly set brackets for singleton sets.

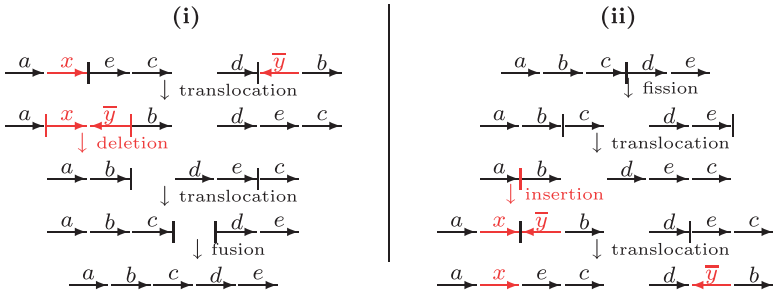

FIG. 3. **(i)** An optimal scenario sorting $\{\circ axec\circ\,,\ \circ d\bar{y}b\circ\}$ into $\{\circ abcde\circ\}$. The first operation (translocation) accumulates x and y, so that they can be deleted at once. **(ii)** Conversely, while sorting $\{\circ abcde\circ\}$ into $\{\circ axec\circ\,,\ \circ d\bar{y}b\circ\}$, we can insert a cluster at once and later split it with a translocation.

we can choose to cut at positions that allow the concatenation of the labels of the original $\mathcal{G}$-adjacencies. As a consequence, a set of labels in $\mathcal{G}$-adjacencies of genome A can be first *accumulated* with optimal DCJ operations and later deleted at once from genome A. In the same way, a set of labels in $\mathcal{G}$-adjacencies of genome B can be first inserted at once as a *cluster* in genome A and later split with optimal DCJ operations, as we can see in Figure 3.

Given a component C of $AG(A, B)$, we can obtain a string $\ell(C)$ by the concatenation of the labels of the $\mathcal{G}$-adjacencies of C in the order in which they appear. Cycles, AA-paths and BB-paths can be read in any direction, but AB-paths should always be read from A to B. If C is a cycle and has labels in both genomes A and B, we should start to read in a labeled $\mathcal{G}$-adjacency v of A, such that the first labeled vertex before v is a $\mathcal{G}$-adjacency in B; otherwise C has labels in at most one genome and we can start anywhere. Each maximal substring of $\ell(C)$ composed only of markers in $\mathcal{A}$ (respectively in $\mathcal{B}$) is called an $\mathcal{A}$-*run* (respectively a $\mathcal{B}$-*run*). Each $\mathcal{A}$-run or $\mathcal{B}$-run can be simply called *run*. A component composed only of clean $\mathcal{G}$-adjacencies has no run and is said to be *clean*, otherwise the component is *labeled*. We denote by $\Lambda(C)$ the number of runs in a component C. A path can have any number of runs, while a cycle has zero, one, or an even number of runs. Figure 4 shows an AB-path with three runs.

Proposition 1. *If $\gamma_1\gamma_2$ is a clean $\mathcal{G}$-adjacency in a DCJ-unsorted component C of $AG(A, B)$, such that neither γ_1 nor γ_2 are telomeres, then it is always possible to extract a clean cycle from C with an optimal DCJ operation.*

Proof. If $\gamma_1\gamma_2$ is in $V_{\mathcal{G}}(B)$, we apply a DCJ on the two vertices $\gamma_1\ell_1\gamma_3$ and $\gamma_2\ell_2\gamma_4$ of $V_{\mathcal{G}}(A)$ that are neighbors of $\gamma_1\gamma_2$, creating the two new vertices $\gamma_3\overline{\ell}_1\ell_2\gamma_4$ and $\gamma_1\gamma_2$. Observe that the vertex $\gamma_1\gamma_2$ in $V_{\mathcal{G}}(B)$ and the new vertex $\gamma_1\gamma_2$ in $V_{\mathcal{G}}(A)$ are extracted into a clean cycle. Analogously, if $\gamma_1\gamma_2$ is in $V_{\mathcal{G}}(A)$, we do the same procedure using the two vertices of $V_{\mathcal{G}}(B)$ that are neighbors of $\gamma_1\gamma_2$. ■

Proposition 2. *A run can be entirely accumulated in the label of one single $\mathcal{G}$-adjacency with optimal DCJ operations.*

Proof. A run that is not yet accumulated is distributed over two or more $\mathcal{G}$-adjacencies in one genome. The $\mathcal{G}$-adjacencies in the other genome within the run are clean. We can thus apply optimal DCJs that extract clean cycles (Proposition 1) and accumulate the entire run in the label of one $\mathcal{G}$-adjacency. ■

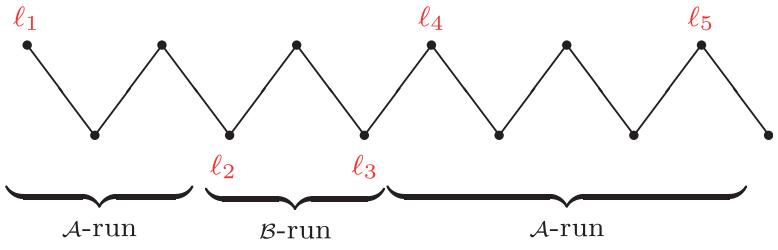

FIG. 4. An AB-path with three runs. Only the labels of the $\mathcal{G}$-adjacencies are represented.

Proposition 2 immediately gives an upper bound for the distance:

Lemma 1. *Given two genomes A and B without duplicated markers, we have*

$$d_{DCJ}^{id}(A,B) \leq d_{DCJ}(A,B) + \sum_{C \in AG(A,B)} \Lambda(C).$$

For some instances of A and B, the upper bound of Lemma 1 gives the exact number of steps required to sort A into B. However, since two runs can be *merged* together with a DCJ operation, the DCJ-indel distance is often smaller than this upper bound. Given a DCJ operation ρ, let Λ_0 and Λ_1 be, respectively, the number of runs in $AG(A, B)$ before and after ρ. We define $\Delta\Lambda(\rho) = \Lambda_1 - \Lambda_0$.

Proposition 3. *Given any DCJ operation ρ, we have $\Delta\Lambda(\rho) \geq -2$.*

Proof. If ρ cuts between an $\mathcal{A}$-run r_1 and a $\mathcal{B}$-run r_2 and between an $\mathcal{A}$-run r_3 and a $\mathcal{B}$-run r_4, with $r_1 \neq r_3$ and $r_2 \neq r_4$, and joins r_1 with r_3 and r_2 with r_4, then $\Delta\Lambda(\rho) = -2$. As a DCJ has at most two cuts and two joins, it is not possible to do better, that is $\Delta\Lambda(\rho) \geq -2$. ■

In order to obtain the exact formula for the DCJ-indel distance, we will first analyze the components of the adjacency graph separately. Given two genomes A and B and a component $C \in AG(A,B)$, we denote by $d_{DCJ}(C)$ the minimum number of DCJ operations required to do a separate DCJ-sorting in C, applying DCJs only on vertices of C (or vertices that result from DCJs applied on vertices that were in C). From Braga and Stoye (2010), we know that it is possible to do a separate DCJ-sorting using only optimal DCJs in any component of $AG(A, B)$, or, in other words, $d_{DCJ}(A,B) = \sum_{C \in AG(A,B)} d_{DCJ}(C)$.

We then define the *indel-potential* of a component C, denoted by $\lambda(C)$, as the minimum number of runs that we can obtain doing a separate DCJ-sorting in C with optimal DCJ operations. The indel-potential of a component C can be computed with a simple formula that depends only on the number of runs in C:

Proposition 4. *Given a component C in AG(A,B), the indel-potential of C is given by $\lambda(C) = \lceil \frac{\Lambda(C)+1}{2} \rceil$, if $\Lambda(C) \geq 1$. Otherwise $\lambda(C) = 0$.*

Proof. The proof is by induction on $i = \Lambda(C)$ and the hypothesis is $T(i) = \lceil \frac{i+1}{2} \rceil$. A labeled DCJ-sorted component can have one or two runs, thus we need two base cases, $T(1) = 1$ and $T(2) = 2$. These cases can be easily verified. More intricate is the inductive step, for $i \geq 3$.

If $i = 3$, the best we can do is to merge the first and the last runs with an optimal DCJ, obtaining a cycle with two runs. This gives $T(3) = T(2) = \lceil \frac{2+1}{2} \rceil = \lceil \frac{3+1}{2} \rceil$. If $i = 4$, any optimal DCJ merging runs would split the original component C into a cycle with two runs and another component with only one run. This gives $T(4) = T(2) + T(1) = 3 = \lceil \frac{4+1}{2} \rceil$.

If $i \geq 5$, the best we can do is to use a DCJ with $\Delta\Lambda = -2$, that extracts three consecutive runs into a cycle of two runs and leaves the other component with $i-4$ runs. We then have $T(i) = T(2) + T(i-4) = 2 + \lceil \frac{i-4+1}{2} \rceil = \lceil \frac{i-4+1+4}{2} \rceil = \lceil \frac{i+1}{2} \rceil$. ■

If λ_0 and λ_1 are, respectively, the sum of the indel-potentials for the components of the adjacency graph before and after a DCJ operation ρ, we define $\Delta\lambda(\rho) = \lambda_1 - \lambda_0$. By the definition of λ, any optimal DCJ ρ acting on a single component has $\Delta\lambda(\rho) \geq 0$. However, considering the case in which only one component is affected by ρ, we still need to investigate $\Delta\lambda(\rho)$ when ρ is neutral or counter-optimal.

Proposition 5. *Given a DCJ operation ρ acting on a single component, we have $\Delta\lambda(\rho) \geq 0$, if ρ is counter-optimal, and $\Delta\lambda(\rho) \geq -1$, if ρ is neutral.*

Proof. The linearization of a cycle is the only counter-optimal DCJ that acts on a single component. This can decrease neither Λ, nor λ. Moreover, when $\Lambda \leq 2$, it is not possible to decrease the indel-potential with any DCJ. When the component has $\Lambda = 3$, the best we can get with a neutral ρ is $\Delta\Lambda(\rho) = -1$. This gives $\lambda_1 = \lceil \frac{(3-1)+1}{2} \rceil = \lceil \frac{3}{2} \rceil = \lceil \frac{3+1}{2} \rceil = \lambda_0$, that is, $\Delta\lambda(\rho) = 0$. And when the component has $\Lambda \geq 4$, we can get $\Delta\Lambda(\rho) = -2$ with a neutral ρ, resulting in $\lambda_1 = \lceil \frac{(\Lambda(C)-2)+1}{2} \rceil = \lceil \frac{\Lambda(C)+1}{2} \rceil - 1 = \lambda_0 - 1$, that is, $\Delta\lambda(\rho) = -1$. ■

We denote by $d^{id}_{DCJ}(C)$ the minimum number of DCJ and indel operations required to sort separately a component C of $AG(A, B)$.

Proposition 6. *If C is a component of $AG(A, B)$, then we have $d^{id}_{DCJ}(C) = d_{DCJ}(C) + \lambda(C)$.*

Proof. By the definition of λ, the best we can do with optimal DCJs is $d_{DCJ}(C) + \lambda(C)$. From Proposition 5, we know that $\Delta\lambda(\rho) \geq 0$ if ρ is a counter-optimal DCJ, thus we can only get longer sorting scenarios if we use such operations. We also know that $\Delta\lambda(\rho) \geq -1$ if ρ is neutral, and, since this kind of operation increases the sorting scenario by one with respect to the scenario with only optimal DCJs, this gives at least $d_{DCJ}(C) + \lambda(C)$. ■

Proposition 6 gives a new upper bound for the DCJ-indel distance:

Lemma 2. *Given two genomes A and B without duplicated markers, we have*

$$d^{id}_{DCJ}(A, B) \leq d_{DCJ}(A, B) + \sum_{C \in AG(A,B)} \lambda(C).$$

Proof. We can sort the components separately with $\sum_{C \in AG(A,B)} d^{id}_{DCJ}(C)$ steps, which corresponds exactly to $d_{DCJ}(A, B) + \sum_{C \in AG(A,B)} \lambda(C)$. ■

Since $\lambda(C) \leq \Lambda(C)$, the upper bound given by Lemma 2 is tighter than the one given by Lemma 1, but can still be improved. Observe that a parsimonious scenario may not simply consist of optimal DCJ operations, insertions and deletions. Sometimes a neutral DCJ can lead to a shorter sequence of operations sorting one genome into another one, as we can see in Figure 5.

4. THE DCJ-INDEL DISTANCE

A DCJ operation ρ that acts on two components is called a *recombination* and can have $\Delta\lambda(\rho) = -2$. The two components on which the cuts are applied are called *sources* and the components obtained after the joinings are called *resultants* of the recombination.

Proposition 7. *Given any recombination ρ, we have $\Delta\lambda(\rho) \geq -2$.*

Proof. Only the recombinations that decrease or do not change the number of runs ($\Delta\Lambda \leq 0$) have to be analyzed (we can not have $\Delta\lambda \leq -1$ if the number of runs increases). First consider the recombination of two paths with i and j runs, respectively, that results in two new paths with i' and j' runs. Observe that the best we can have is when i and j are even, i' and j' are odd and $\Delta\Lambda = -2$, that gives: $\lambda_1 = \lceil \frac{i'+1}{2} \rceil + \lceil \frac{j'+1}{2} \rceil = \frac{i'+j'+2}{2} = \frac{i+j}{2} = \frac{i}{2} + \frac{j}{2} = \lceil \frac{i+1}{2} \rceil - 1 + \lceil \frac{j+1}{2} \rceil - 1 = \lambda_0 - 2$. The analysis of recombinations involving cycles is analogous. ■

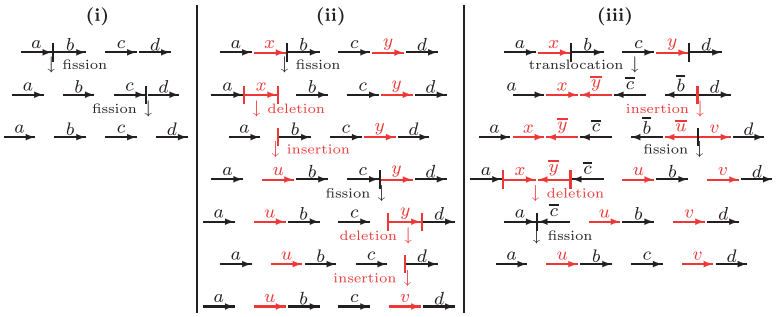

FIG. 5. An optimal scenario sorting $\{\circ ab\circ, \circ cd\circ\}$ into $\{\circ a\circ, \circ b\circ, \circ c\circ, \circ d\circ\}$ **(i)** and two different scenarios sorting $\{\circ axb\circ, \circ cyd\circ\}$ into $\{\circ a\circ, \circ ub\circ, \circ c\circ, \circ vd\circ\}$. In **(ii)** in addition to the two optimal DCJ operations (fissions) from (i) we have two insertions and two deletions, using six steps. In **(iii)**, we first use a neutral DCJ operation (translocation) that allows us to do only one deletion and one insertion, achieving a total of five steps.

Given a recombination ρ, let $\Delta_{dcj}(\rho)$ be respectively 0, +1 and +2 depending whether ρ is optimal, neutral or counter-optimal. Any recombination applied to a vertex of an *AA*-path and a vertex of a *BB*-path is optimal (Braga and Stoye, 2010). A recombination applied to vertices of two different *AB*-paths can be either neutral, when the result is also a pair of *AB*-paths, or counter-optimal, when the result is a pair composed of an *AA*-path and a *BB*-path. All other types of path recombinations are neutral. In addition, all recombinations involving at least one cycle are counter-optimal. We define $\Delta d(\rho) = \Delta_{dcj}(\rho) + \Delta\lambda(\rho)$. Any counter-optimal recombination has $\Delta d \geq 0$, thus only path recombinations can have $\Delta d \leq -1$.

Let $\mathcal{A}$ (respectively $\mathcal{B}$) be a sequence with an odd ($\geq$1) number of runs, starting and ending with an $\mathcal{A}$-run (respectively $\mathcal{B}$-run). We can then make any combination of $\mathcal{A}$ and $\mathcal{B}$, such as $\mathcal{AB}$, that is a sequence with an even ($\geq$2) number of runs, starting with an $\mathcal{A}$-run and ending with a $\mathcal{B}$-run. An empty sequence (with no run) is represented by ε. Then each one of the notations AA_ε, $AA_\mathcal{A}$, $AA_\mathcal{B}$, $AA_\mathcal{AB}$, BB_ε, $BB_\mathcal{A}$, $BB_\mathcal{B}$, $BB_\mathcal{AB}$, AB_ε, $AB_\mathcal{A}$, $AB_\mathcal{B}$, $AB_\mathcal{AB}$ and $AB_\mathcal{BA}$ represents a particular type of path (*AA, BB* or *AB*) with a particular structure of runs (ε, $\mathcal{A}$, $\mathcal{B}$, $\mathcal{AB}$ or $\mathcal{BA}$). The complete set of path recombinations with $\Delta d \leq -1$ is given in Table 1. In Table 2 we list recombinations with $\Delta d = 0$ that create at least one source of recombinations of Table 1. We denote by $AB_\bullet$ an *AB*-path that can not be a source of a recombination in Tables 1 and 2, such as $AB_\varepsilon, AB_\mathcal{A}$ *and* $AB_\mathcal{B}$.

Proposition 8. *The recombinations with* $\Delta d = 0$ *involving cycles or circular singletons cannot create new components that can be used as sources of recombinations listed in Tables 1 and 2.*

Proof. A recombination ρ with $\Delta d = 0$ involving a cycle or a circular singleton C would integrate C to another component C' without changing the type or the structure of runs in C'. Thus, if C' is a source of a recombination in these tables after ρ, C' was already the same type of source before ρ. And if C' was not a source before ρ, C' cannot become a source after ρ. ■

With Proposition 8, we already have an exact formula to compute d^{id}_{DCJ} for a particular set of instances. Given a $\mathcal{G}$-adjacency $\gamma\ell\circ$ of a genome A such that $\gamma \neq \circ$, then γ is said to be a *tail* of a linear chromosome in A. Two genomes are *co-tailed* if their sets of tails are equal (this includes two genomes composed only of circular chromosomes).

Theorem 2. *Given two co-tailed genomes A and B without duplicated markers, we have* $d^{id}_{DCJ}(A,B) = d_{DCJ}(A,B) + \sum_{C \in AG(A,B)} \lambda(C)$.

Proof. The graph $AG(A, B)$ for co-tailed genomes A and B can have only cycles, singletons (that could be $AA_\mathcal{A}$ and $BB_\mathcal{B}$) and *AB*-paths of one edge (that could be $AB_\mathcal{AB}$, but never $AB_\mathcal{BA}$). Observe that with these paths no recombination with $\Delta d \leq -1$ is possible. From Proposition 8, we also know that cycles cannot lead to recombinations with $\Delta d \leq -1$. ■

Now we continue the analysis for the general case. The two sources of a recombination can also be called *partners*. Looking at Table 1 we observe that all partners of $AB_\mathcal{AB}$ and $AB_\mathcal{BA}$ paths are also partners of $AA_\mathcal{AB}$ and $BB_\mathcal{AB}$ paths, all partners of $AA_\mathcal{A}$ and $AA_\mathcal{B}$ paths are also partners of $AA_\mathcal{AB}$ paths and all partners of $BB_\mathcal{A}$ and $BB_\mathcal{B}$ paths are also partners of $BB_\mathcal{AB}$ paths. Moreover, some resultants of recombinations in Tables 1 and 2 can be used in other recombinations. These observations allow the identification of *recombination groups*, as listed in Table 3.

TABLE 1. PATH RECOMBINATIONS THAT HAVE $\Delta d \leq -1$ AND ALLOW THE BEST REUSE OF THE RESULTANTS

Sources	*Resultants*	$\Delta\lambda$	Δ_{dcj}	Δd	*Sources*	*Resultants*	$\Delta\lambda$	Δ_{dcj}	Δd
$AA_\mathcal{AB} + BB_\mathcal{AB}$	$AB_\bullet + AB_\bullet$	−2	0	−2	$AA_\mathcal{AB} + AA_\mathcal{AB}$	$AA_\mathcal{A} + AA_\mathcal{B}$	−2	+1	−1
$AA_\mathcal{A} + BB_\mathcal{AB}$	$AB_\bullet + AB_\mathcal{AB}$	−1	0	−1	$BB_\mathcal{AB} + BB_\mathcal{AB}$	$BB_\mathcal{A} + BB_\mathcal{B}$	−2	+1	−1
$BB_\mathcal{A} + AA_\mathcal{AB}$	$AB_\bullet + AB_\mathcal{BA}$	−1	0	−1	$AA_\mathcal{AB} + AB_\mathcal{AB}$	$AB_\bullet + AA_\mathcal{A}$	−2	+1	−1
$AA_\mathcal{B} + BB_\mathcal{AB}$	$AB_\bullet + AB_\mathcal{BA}$	−1	0	−1	$AA_\mathcal{AB} + AB_\mathcal{BA}$	$AB_\bullet + AA_\mathcal{B}$	−2	+1	−1
$BB_\mathcal{B} + AA_\mathcal{AB}$	$AB_\bullet + AB_\mathcal{AB}$	−1	0	−1	$BB_\mathcal{AB} + AB_\mathcal{AB}$	$AB_\bullet + BB_\mathcal{B}$	−2	+1	−1
$AA_\mathcal{A} + BB_\mathcal{A}$	$AB_\bullet + AB_\bullet$	−1	0	−1	$BB_\mathcal{AB} + AB_\mathcal{BA}$	$AB_\bullet + BB_\mathcal{A}$	−2	+1	−1
$AA_\mathcal{B} + BB_\mathcal{B}$	$AB_\bullet + AB_\bullet$	−1	0	−1	$AB_\mathcal{AB} + AB_\mathcal{BA}$	$AB_\bullet + AB_\bullet$	−2	+1	−1

Optimal recombinations are on the left, neutral recombinations on the right.

TABLE 2. RECOMBINATIONS THAT HAVE $\Delta d = 0$ AND CREATE RESULTANTS THAT CAN BE USED IN RECOMBINATIONS WITH $\Delta d \leq 1$

Sources	*Resultants*	$\Delta\lambda$	Δ_{dcj}	Δd	*Sources*	*Resultants*	$\Delta\lambda$	Δ_{dcj}	Δd
$AA_{\mathcal{A}} + AB_{\mathcal{BA}}$	$AB_{\bullet} + AA_{\mathcal{AB}}$	-1	$+1$	0	$AA_{\mathcal{A}} + BB_{\mathcal{B}}$	$AB_{\bullet} + AB_{\mathcal{AB}}$	0	0	0
$AA_{\mathcal{B}} + AB_{\mathcal{AB}}$	$AB_{\bullet} + AA_{\mathcal{AB}}$	-1	$+1$	0	$AA_{\mathcal{B}} + BB_{\mathcal{A}}$	$AB_{\bullet} + AB_{\mathcal{BA}}$	0	0	0
$BB_{\mathcal{A}} + AB_{\mathcal{AB}}$	$AB_{\bullet} + BB_{\mathcal{AB}}$	-1	$+1$	0	$AB_{\mathcal{AB}} + AB_{\mathcal{AB}}$	$AA_{\mathcal{A}} + BB_{\mathcal{B}}$	-2	$+2$	0
$BB_{\mathcal{B}} + AB_{\mathcal{BA}}$	$AB_{\bullet} + BB_{\mathcal{AB}}$	-1	$+1$	0	$AB_{\mathcal{BA}} + AB_{\mathcal{BA}}$	$AA_{\mathcal{B}} + BB_{\mathcal{A}}$	-2	$+2$	0

The deductions shown in Table 3 can be computed with an approach that greedily maximizes the number of recombinations in *P, Q, T, S, M* and *N* in this order. The *P* part contains only one operation and is thus very simple. The same happens with *Q*, since the two groups in this part are mutually exclusive after applying *P*. The part *T* is only the application of all possible remaining groups of two operations with $\Delta d = -2$. Similarly, the part *S* is only the application of all possible remaining operations with $\Delta d = -1$. After *S*, the two groups in *M* are mutually exclusive and then the same happens to the six groups in *N*.

TABLE 3. ALL RECOMBINATION GROUPS (*P, Q, T,* AND *S*) OBTAINED FROM TABLE 1

	Sources	*Resultants*	Δd	*scr*
P	$AA_{\mathcal{AB}} + BB_{\mathcal{AB}}$	$2AB_{\bullet}$	-2	-1
Q	$2AA_{\mathcal{AB}} + BB_{\mathcal{A}} + BB_{\mathcal{B}}$	$4AB_{\bullet}$	-3	$-3/4$
	$2BB_{\mathcal{AB}} + AA_{\mathcal{A}} + AA_{\mathcal{B}}$	$4AB_{\bullet}$	-3	$-3/4$
T	$AA_{\mathcal{AB}} + BB_{\mathcal{A}} + AB_{\mathcal{AB}}$	$3AB_{\bullet}$	-2	$-2/3$
	$AA_{\mathcal{AB}} + BB_{\mathcal{B}} + AB_{\mathcal{BA}}$	$3AB_{\bullet}$	-2	$-2/3$
	$BB_{\mathcal{AB}} + AA_{\mathcal{A}} + AB_{\mathcal{BA}}$	$3AB_{\bullet}$	-2	$-2/3$
	$BB_{\mathcal{AB}} + AA_{\mathcal{B}} + AB_{\mathcal{AB}}$	$3AB_{\bullet}$	-2	$-2/3$
	$2AA_{\mathcal{AB}} + BB_{\mathcal{A}}$	$2AB_{\bullet} + AA_{\mathcal{B}}$	-2	$-2/3$
	$2AA_{\mathcal{AB}} + BB_{\mathcal{B}}$	$2AB_{\bullet} + AA_{\mathcal{A}}$	-2	$-2/3$
	$2BB_{\mathcal{AB}} + AA_{\mathcal{A}}$	$2AB_{\bullet} + BB_{\mathcal{B}}$	-2	$-2/3$
	$2BB_{\mathcal{AB}} + AA_{\mathcal{B}}$	$2AB_{\bullet} + BB_{\mathcal{A}}$	-2	$-2/3$
S	$AA_{\mathcal{A}} + BB_{\mathcal{A}}$	$2AB_{\bullet}$	-1	$-1/2$
	$AA_{\mathcal{B}} + BB_{\mathcal{B}}$	$2AB_{\bullet}$	-1	$-1/2$
	$AB_{\mathcal{AB}} + AB_{\mathcal{BA}}$	$2AB_{\bullet}$	-1	$-1/2$
	$AA_{\mathcal{AB}} + BB_{\mathcal{A}}$	$AB_{\bullet} + AB_{\mathcal{BA}}$	-1	$-1/2$
	$AA_{\mathcal{AB}} + BB_{\mathcal{B}}$	$AB_{\bullet} + AB_{\mathcal{AB}}$	-1	$-1/2$
	$BB_{\mathcal{AB}} + AA_{\mathcal{A}}$	$AB_{\bullet} + AB_{\mathcal{AB}}$	-1	$-1/2$
	$BB_{\mathcal{AB}} + AA_{\mathcal{B}}$	$AB_{\bullet} + AB_{\mathcal{BA}}$	-1	$-1/2$
	$AA_{\mathcal{AB}} + AB_{\mathcal{AB}}$	$AB_{\bullet} + AA_{\mathcal{A}}$	-1	$-1/2$
	$AA_{\mathcal{AB}} + AB_{\mathcal{BA}}$	$AB_{\bullet} + AA_{\mathcal{B}}$	-1	$-1/2$
	$BB_{\mathcal{AB}} + AB_{\mathcal{AB}}$	$AB_{\bullet} + BB_{\mathcal{B}}$	-1	$-1/2$
	$BB_{\mathcal{AB}} + AB_{\mathcal{BA}}$	$AB_{\bullet} + BB_{\mathcal{A}}$	-1	$-1/2$
	$AA_{\mathcal{AB}} + AA_{\mathcal{AB}}$	$AB_{\mathcal{A}} + AA_{\mathcal{B}}$	-1	$-1/2$
	$BB_{\mathcal{AB}} + BB_{\mathcal{AB}}$	$BB_{\mathcal{A}} + BB_{\mathcal{B}}$	-1	$-1/2$
M	$2AB_{\mathcal{AB}} + AA_{\mathcal{B}} + BB_{\mathcal{A}}$	$4AB_{\bullet}$	-2	$-1/2$
	$2AB_{\mathcal{BA}} + AA_{\mathcal{A}} + BB_{\mathcal{B}}$	$4AB_{\bullet}$	-2	$-1/2$
N	$AB_{\mathcal{AB}} + AA_{\mathcal{B}} + BB_{\mathcal{A}}$	$2AB_{\bullet} + AB_{\mathcal{BA}}$	-1	$-1/3$
	$AB_{\mathcal{BA}} + AA_{\mathcal{A}} + BB_{\mathcal{B}}$	$2AB_{\bullet} + AB_{\mathcal{AB}}$	-1	$-1/3$
	$2AB_{\mathcal{AB}} + AA_{\mathcal{B}}$	$2AB_{\bullet} + AA_{\mathcal{A}}$	-1	$-1/3$
	$2AB_{\mathcal{AB}} + BB_{\mathcal{A}}$	$2AB_{\bullet} + BB_{\mathcal{B}}$	-1	$-1/3$
	$2AB_{\mathcal{BA}} + AA_{\mathcal{A}}$	$2AB_{\bullet} + AA_{\mathcal{B}}$	-1	$-1/3$
	$2AB_{\mathcal{BA}} + BB_{\mathcal{B}}$	$2AB_{\bullet} + BB_{\mathcal{A}}$	-1	$-1/3$

Observe that the last four groups in *T* are subsets of groups in *Q* and the last ten groups in *S* are subsets of groups in *T*. The groups in *M* and *N* contain operations from Tables 1 and 2. All groups in *N* are subsets of the groups in *M*. The table is sorted in descending order with respect to the contribution of each path in the distance decrease (column **scr**).

Although some groups in *T, S, M* and *N* have some reusable resultants, those are actually never reused (if operations that are lower in the table use as sources resultants from higher operations, the sources of all referred operations would be previously consumed in operations that occupy even higher positions in the table). Due to this fact, the number of operations in *P, Q, T, S, M* and *N* depends only on the initial number of each type of component.

The results presented in this section give rise to the following theorem, which gives the exact formula for the DCJ-indel distance:

Theorem 3. *Given two genomes A and B without duplicated markers, we have*

$$d_{DCJ}^{id}(A,B) = d_{DCJ}(A,B) + \sum_{C \in AG(A,B)} \lambda(C) - 2P - 3Q - 2T - S - 2M - N,$$

where the values P, Q, T, S, M and N are computed as described above.

Both $AG(A, B)$ and $d_{DCJ}(A, B)$ can be computed in linear time (Bergeron et al., 2006). The runs can be obtained by a single walk through each component of $AG(A, B)$, which is also linear. The algorithm to compute *P, Q, T, S, M* and *N* is a finite sequence of **if** and **else** statements, that depends only on the number of each type of labeled path in $AG(A, B)$, thus the whole procedure takes linear time.

5. DCJ-INDEL SORTING

One may have observed that the approach presented in the previous sections is bi-directional, allowing operations on both genomes *A* and *B*, in order to be able to accumulate labels in $\mathcal{G}$-adjacencies of both genomes. As a result, we actually transform genomes *A* and *B* into an intermediate genome *I*. Regarding the operations applied on genome *B*, this can be seen as a backtracing to find the best moment to do a cluster insertion in genome *A*, as shown in Figure 6. Thus, with the operations that transform *A* and *B* into *I*, we can derive an optimal sequence of operations simply sorting genome *A* into *B*.

Given any DCJ or indel operation $\rho = (X \to Y)$, we define the *inversion* of ρ as $\rho^{-1} = (Y \to X)$. Observe that the inversion of a deletion is an insertion, and *vice-versa*. We can also extend this notation to a sequence of operations: given a sequence $s = \rho_1 \rho_2 \ldots \rho_n$, we have $s^{-1} = \rho_n^{-1} \rho_{n-1}^{-1} \ldots \rho_2^{-1} \rho_1^{-1}$.

Proposition 9. *Given two genomes A and B, and a pair of sequences s_1 and s_2 composed of DCJ and indel operations acting on both genomes A and B, transforming respectively A and B into an intermediate genome I, such that $|s_1| + |s_2| = d_{DCJ}^{id}(A,B)$, then $s_1 s_2^{-1}$ is an optimal sequence of DCJ and indel operations that transforms A into B.*

Proof. First note that, since s_1 sorts *A* into *I* and s_2^{-1} sorts *I* into *B*, $s_1 s_2^{-1}$ sorts *A* into *B*. Furthermore, $s_1 s_2^{-1}$ has length $d_{DCJ}^{id}(A,B)$ and is an optimal sorting sequence. Otherwise, there would exist a sorting sequence shorter than $d_{DCJ}^{id}(A,B)$, which is a contradiction. ■

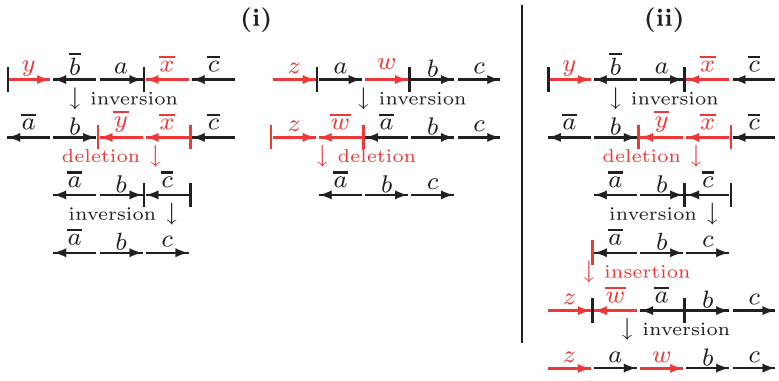

FIG. 6. **(i)** Two sequences of lengths 3 and 2, sorting $A = \{\circ y\overline{b}a\overline{x}\overline{c}\circ\}$ and $B = \{\circ zawbc\circ\}$ into $I = \{\circ\overline{a}bc\circ\}$. **(ii)** A corresponding sequence of length 5 sorting *A* into *B*.

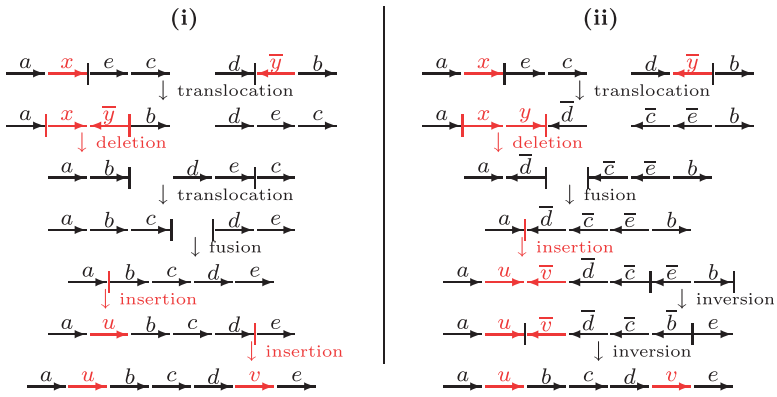

FIG. 7. Two optimal scenarios sorting $\{\circ axec\circ\,,\circ d\bar{y}b\circ\}$ into $\{\circ aubcdve\circ\}$. (i) Minimizing DCJs gives three DCJs and three indels. **(ii)** Minimizing indels gives four DCJs and two indels.

It is also interesting to observe that the space of solutions of the sorting problem contains scenarios with different compositions and, using the same number of steps, one could look for a scenario with more indels and less DCJ operations, or for a scenario with less indels and more DCJ operations. Figure 7 shows examples of two different approaches: one that minimizes the number of DCJs, and one that minimizes the number of indels. Both approaches will be presented in the following.

5.1. Sorting with a minimum number of DCJ operations

First we will derive a sorting algorithm directly from the formula given in Theorem 3. In general lines, Algorithm 1 constructs incrementally two sequences of operations, s_1 and s_2, such that the operations in s_1 are applied on A and the operations in s_2 are applied on B (although we do not explicitly identify in the pseudo-code which operations are in s_1 and which operations are in s_2). As previously stated, an optimal sequence sorting A into B is given by $s_1 s_2^{-1}$.

Algorithm 1: Sorting genome A into B with a minimum number of DCJs

1. Apply all *P, Q, T, S, M* and *N*-recombinations, in this order.
2. For each component $C \in AG(A,B)$:
   (a) Split C with **optimal** DCJs (that have $\mathbf{\Delta\lambda = 0}$) until only components with at most two runs are obtained and the total number of runs in all new components is equal to $\lambda(C)$.
   (b) Accumulate all runs in the smaller components derived from C with **optimal** DCJ operations (that have $\mathbf{\Delta\lambda = 0}$).
   (c) Apply **optimal** DCJ operations (that have $\mathbf{\Delta\lambda = 0}$) in the smaller components derived from C until only DCJ-sorted components exist.
   (d) **Delete** all runs in the DCJ-sorted components derived from C.

Algorithm 1 does the minimum number of DCJs with $\Delta\lambda \leq -1$ (only those that are in recombinations of step 1). All other operations applied are optimal DCJs with $\Delta\lambda = 0$ and indels. As a consequence, this gives a sorting scenario that minimizes the number of DCJs, with respect to indels.

5.2. Sorting with a minimum number of indels

In order to design an algorithm to sort a genome A into a genome B minimizing the number of indels, we need to use the DCJs with lower $\Delta\lambda$, among those with $\Delta d = 0$. Thus, instead of using optimal DCJs with $\Delta\lambda = 0$, as in steps 2(a,b,c) of the previous algorithm, we shall maximize the use of counter-optimal DCJs with $\Delta\lambda = -2$ and neutral DCJs with $\Delta\lambda = -1$.

First we will analyze the operations acting on a single component. In this case, any counter-optimal DCJ has $\Delta\lambda \geq 0$. The same happens with any neutral DCJ acting on a single component C with $\lambda(C) \leq 3$. Only neutral DCJs acting on a single component with at least four runs can have $\Delta\lambda = -1$:

Proposition 10. *Given a component C with $\Lambda(C) \geq 4$, the best we can get with a neutral DCJ operation ρ acting only on C is $\Delta\Lambda(\rho) = -2$ and $\Delta\lambda(\rho) = -1$.*

Proof. When the component has $\Lambda \geq 4$, the operation ρ can cut after the first and before the last run, and simply invert this segment. In this case we get $\Delta\Lambda(\rho) = -2$, resulting in $\lambda_1 = \lceil \frac{(\Lambda(C)-2)+1}{2} \rceil = \lceil \frac{\Lambda(C)+1}{2} \rceil - 1 = \lambda_0 - 1$. (Since we can apply only two cuts and two joins, it is not possible to do better.) ■

Proposition 10 guarantees that, with neutral DCJs, we can merge runs in any component C with $\Lambda(C) \geq 4$, such that in the end of the process C has only two or three runs. In order to maximize the use of these neutral DCJs, a good strategy is to first regroup runs efficiently into one component. Thus, the recombinations in *P, Q, T, S, M* and *N* should be done such that one of the resultants, that should be an *AB*-path whenever it is possible, has zero or one run, and the other resultant has all the remaining runs (this can be done for all mentioned recombinations). In the following steps, we can regroup runs by using recombinations with $\Delta d = 0$, as those listed in Table 2. Additional recombinations with $\Delta d = 0$ are listed in Table 4, in which $O_{\mathcal{AB}}$ is a cycle with at least one $\mathcal{A}$-run and one $\mathcal{B}$-run, and $S_{\mathcal{A}}$ and $S_{\mathcal{B}}$ are circular singletons in genomes *A* and *B*, respectively. All recombinations in Table 4 regroup all remaining runs in one single component, thus they should be applied before the neutral DCJs from Proposition 10. The same happens with the neutral and optimal recombinations in Table 2. (Instead of using the two counter-optimal recombinations of Table 2 we use the two neutral recombinations of Table 4 that have the same sources, but regroup all remaining runs in one component.)

From the previous observations we can derive Algorithm 2, to sort a genome *A* into a genome *B* minimizing the number of insertions and deletions.

It is interesting to observe that Algorithm 2 merges into only two runs all runs from all components that have two or more runs, together with the runs from all paths that have only one run. On the other hand, the runs that appear in cycles that have only one run could only be merged with circular singletons.

5.3. Implementation and complexity

The labels can be stored in bi-directional linked lists, so that the operations *reverse* and *concatenate* can be done in constant time. Each label is associated to an adjacency, that can be stored in tables similar to those proposed by Bergeron et al. (2006) for the traditional DCJ model, with the difference that here we may apply operations on both genomes *A* and *B*.

We also need to store bi-directional links from the first to the last labelled adjacency in each run (these adjacencies are in the same genome), and from the last labelled adjacency in one run to the first labelled adjacency in the next run (these adjacencies are in different genomes). Additionaly, we only need links to the first labelled adjacency in the first run and to the last labelled adjacency in the last run of each component. All these links require linear space and can be set while traversing the components once.

Table 4. Further Neutral and Counter-Optimal Recombinations with $\Delta d = 0$ (in Addition to Those Listed in Table 2)

Sources	*Resultants*	$\Delta\lambda$	Δ_{dcj}	Δd	*Sources*	*Resultants*	$\Delta\lambda$	Δ_{dcj}	Δd
$O_{\mathcal{AB}} + C_1^*$	C_1^*	-2	$+2$	0	$AA_{\mathcal{A}} + P_1^*$	$AA_{\varepsilon} + P_1^*$	-1	$+1$	0
$S_{\mathcal{A}} + C_2^*$	C_1^*	-1	$+1$	0	$AA_{\mathcal{B}} + P_2^*$	$AA_{\varepsilon} + P_2^*$	-1	$+1$	0
$S_{\mathcal{B}} + C_3^*$	C_3^*	-1	$+1$	0	$BB_{\mathcal{A}} + P_3^*$	$BB_{\varepsilon} + P_3^*$	-1	$+1$	0
$AB_{\mathcal{AB}} + AB_{\mathcal{AB}}$	$AB_{\varepsilon} + AB_{\mathcal{AB}}$	-1	$+1$	0	$BB_{\mathcal{B}} + P_4^*$	$BB_{\varepsilon} + P_4^*$	-1	$+1$	0
$AB_{\mathcal{BA}} + AB_{\mathcal{BA}}$	$AB_{\varepsilon} + AB_{\mathcal{BA}}$	-1	$+1$	0	$AB_{\mathcal{A}} + P_5^*$	$AB_{\varepsilon} + P_5^*$	-1	$+1$	0
					$AB_{\mathcal{B}} + P_6^*$	$AB_{\varepsilon} + P_6^*$	-1	$+1$	0

*The components C_1, C_2, C_3, P_1, P_2, P_3, P_4, P_5 and P_6 can be as follows:
C_1 : $O_{\mathcal{AB}}$, $AA_{\mathcal{AB}}$, $AA_{\mathcal{ABA}}$, $AA_{\mathcal{BAB}}$, $BB_{\mathcal{AB}}$, $BB_{\mathcal{ABA}}$, $BB_{\mathcal{BAB}}$, $AB_{\mathcal{AB}}$, $AB_{\mathcal{BA}}$, $AB_{\mathcal{ABA}}$ or $AB_{\mathcal{BAB}}$.
C_2 : $S_{\mathcal{A}}$, $O_{\mathcal{A}}$, $O_{\mathcal{AB}}$, $AA_{\mathcal{A}}$, $AA_{\mathcal{AB}}$, $AA_{\mathcal{BAB}}$, $BB_{\mathcal{A}}$, $BB_{\mathcal{AB}}$, $BB_{\mathcal{BAB}}$, $AB_{\mathcal{A}}$, $AB_{\mathcal{AB}}$, $AB_{\mathcal{BA}}$ or $AB_{\mathcal{BAB}}$.
C_3 : $S_{\mathcal{B}}$, $O_{\mathcal{B}}$, $O_{\mathcal{AB}}$, $AA_{\mathcal{B}}$, $AA_{\mathcal{AB}}$, $AA_{\mathcal{ABA}}$, $BB_{\mathcal{B}}$, $BB_{\mathcal{AB}}$, $BB_{\mathcal{ABA}}$, $AB_{\mathcal{B}}$, $AB_{\mathcal{AB}}$, $AB_{\mathcal{BA}}$ or $AB_{\mathcal{ABA}}$.
P_1 : $AA_{\mathcal{AB}}$, $AA_{\mathcal{A}}$ or $AB_{\mathcal{AB}}$.
P_2 : $AA_{\mathcal{AB}}$, $AA_{\mathcal{B}}$ or $AB_{\mathcal{BA}}$
P_3 : $BB_{\mathcal{AB}}$, $BB_{\mathcal{A}}$ or $AB_{\mathcal{BA}}$.
P_4 : $BB_{\mathcal{AB}}$, $BB_{\mathcal{B}}$ or $AB_{\mathcal{AB}}$.
P_5 : $AA_{\mathcal{A}}$, $BB_{\mathcal{A}}$, $AB_{\mathcal{AB}}$, $AB_{\mathcal{BA}}$ or $AB_{\mathcal{A}}$.
P_6 : $AA_{\mathcal{B}}$, $BB_{\mathcal{B}}$, $AB_{\mathcal{AB}}$, $AB_{\mathcal{BA}}$, or $AB_{\mathcal{B}}$.
Each operation here has a resultant of the same type as one of its sources.

Algorithm 2: Sorting genome A into B with a minimum number of indels

1. Apply all *P, Q, T, S, M* and *N*-recombinations, in this order, such that, for each recombination, one of the resultants, that should be an AB-path whenever it is possible, has zero or one run, and the other resultant has all the remaining runs.
2. While there is a DCJ ρ, such that ρ is either a **counter-optimal** recombination from Table 4, or a **neutral** recombination from Tables 2 or 4, or an **optimal** recombination from Table 2, apply ρ.
   [After this step, there is at most one comp. with two or more runs; the others have at most one run.]
3. For each component $C \in AG(A, B)$:
   (a) While $\Lambda(C) \geq 4$, apply a **neutral** DCJ on C with $\Delta\lambda = -\mathbf{1}$ (Prop. 10).
   (b) If $\Lambda(C) = 3$ (C is a path), merge the last and the first runs of C, extracting a cycle with all runs (**optimal** DCJ with $\Delta\lambda = \mathbf{0}$).
   (c) Accumulate all runs in the smaller components derived from C with **optimal** DCJ operations that have $\Delta\lambda = \mathbf{0}$.
   (d) Apply **optimal** DCJ operations in the smaller components derived from C until only DCJ-sorted components exist (these DCJs have $\Delta\lambda = \mathbf{0}$).
   (e) **Delete** all runs in the DCJ-sorted components derived from C.

All recombinations in both Algorithms 1 and 2 can be applied either cutting before or after the first run, or before or after the last run in each one of the two sources. These positions can be accessed and updated in constant time with the bi-directional links. After the recombinations, the components may be sorted separately.

- In Algorithm 1, for each component C with $\Lambda(C) \geq 3$, we use optimal DCJ operations to split C properly, such that we end up with several components with one or two runs each, and the sum of all runs is equal to $\lambda(C)$. These optimal DCJs may be consecutively applied while traversing the component C once through the links between runs.
- In Algorithm 2, for each component C with $\Lambda(C) \geq 4$, we use neutral DCJs with $\Delta\lambda = -1$ that do not split the component. These neutral DCJs may be consecutively applied, while traversing the component C once through the links between runs. We end up with a component C with two or three runs. In the last case, we merge the first and the last run with an optimal DCJ.

The remaining steps are equivalent in Algorithms 1 and 2. First we accumulate runs extracting clean cycles and then we do the final DCJ-sorting steps and deletions.

All merging and accumulating DCJ operations concatenate labels of adjacencies. However, as previously mentioned, in each operation the affected labels can be updated in constant time. All DCJ operations that merge runs only in A and all DCJ operations that merge runs in A and in B should be applied on genome A, as well as the DCJ operations that accumulate runs in genome A. On the other hand, all DCJ operations that merge runs only in B should be applied on genome B, as well as the DCJ operations that accumulate runs in genome B. The DCJ operations that do no affect labels can be applied on genome A.

In the end of the process, we have s_1, which is the sequence of operations applied on genome A, and s_2, that is the sequence of operations applied on genome B, and we can obtain $s_1 s_2^{-1}$ in linear time. Since the DCJ-indel distance and the sizes of the components are bounded by $|\mathcal{G}| + |\mathcal{A}| + |\mathcal{B}|$, both Algorithms 1 and 2 can be implemented in $O(|\mathcal{G}| + |\mathcal{A}| + |\mathcal{B}|)$ time and space.

6. EXPERIMENTS

We used our method to analyze the evolution of *Rickettsia*, a group of obligate intracellular parasites that are carried by many vectors (frequently hematophagous arthropods) and occasionally transmitted from the vector to mammalians (including humans), causing several diseases (typhus, spotted fever, etc.) (Blanc et al., 2007). There are several completely sequenced *Rickettsia* genomes, and most of them are closely related (Blanc et al., 2007). The exception is *R. bellii*, which shows a high level of rearrangement with respect to the others. Figure 8 shows the phylogenetic tree of seven species of *Rickettsia*, given in Blanc et al. (2007).

We compared *R. bellii* with six other species of *Rickettsia*, observing in all pairwise analyses a considerable reduction of the indels (Table 5), when they are grouped into runs (column $\Sigma\Lambda$) and into merged runs (column $\Sigma\lambda$). We also computed the number of each type of operation with respect to the two given sorting approaches (two last columns of Table 5). Our results show that, while the number of indels varies

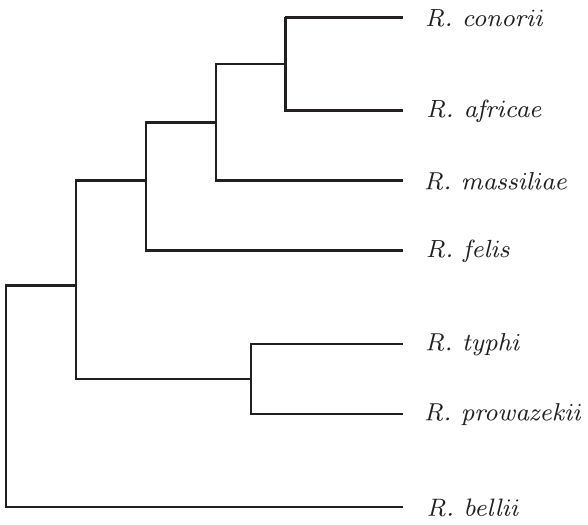

FIG. 8. Phylogenetic tree of seven species of *Rickettsia* (given in Blanc et al., 2007).

from 114 to 181 with the algorithm that maximizes indels, we found a much smaller variation (from 87 to 92) with the algorithm that minimizes indels. The few occurrences of duplicated markers in the considered species were ignored in our analysis.

These experiments were carried out in order to show the contrast of the number of clusters that can be obtained with the different approaches, but they helped to identify interesting points to be explored. The runs can be merged in many different ways, resulting in many different possible combinations of clusters. However, in each pairwise comparison, there are runs that cannot be merged–those that occur in cycles containing only one run, also called *1-run cycles*. Our goal is to use the 1-run cycles, listed in Table 6, to identify deletions that occurred immediately after the differentiation of *R. bellii* from the other six species and that are coherent with all the optimal DCJ-indel sequences sorting *R. bellii* into each one of the other six species.

As we can see in Table 6, the distribution of the 1-run cycles allows to distinguish three different groups, that are coherent with the topology of the given phylogenetic tree (Fig. 8). In the first group, composed only of *R. felis*, we have 46 runs in *R. bellii* and 39 in *R. felis*. In the second group, composed of *R. massiliae, R. africae* and *R. conorii*, we have around 53 runs in *R. bellii* and around 33 runs in the second genome. And finally in the third group, composed of *R. prowazekii* and *R. thyphi*, we have around 80 runs in *R. bellii* and around 10 runs in the second genome.

Examining the genes that form all runs presented in Table 6, we could actually identify, out of possible 9 (that is the minimum in the fourth column), six deletions in *R. bellii*. These deletions correspond to identical runs of the other six species that occur in 1-run cycles in the comparison against *R. bellii*. We could also identify, out of 46 (that is the minimum in the third column), almost 30 deletions in the branch that leads to the other six species. Analogously, these deletions correspond to identical runs of *R. bellii* that occur in 1-run cycles in the comparison against the other six species.

Although these are preliminary results, they show that our methods can be used to guide the study of evolution in practice.

TABLE 5. COMPARING *R. BELLII* (1.52 MBP) WITH SIX OTHER SPECIES OF *RICKETTSIA*

Species	*Mbp*	$\lvert\mathcal{A}\rvert + \lvert\mathcal{B}\rvert$	$\Sigma\Lambda$	$\Sigma\lambda$	d_{DCJ}	d_{DCJ}^{id}	*MIN DCJs (DCJs + indels)*	*MIN indels (DCJs + indels)*
R. felis	1.55	333	241	181	312	493	312 + 181	406 + 87
R. massiliae	1.36	302	218	172	276	448	276 + 172	358 + 90
R. africae	1.28	290	212	166	260	426	260 + 166	338 + 88
R. conorii	1.27	277	192	153	261	414	261 + 153	326 + 88
R. prowazekii	1.11	241	130	117	197	314	197 + 117	222 + 92
R. typhi	1.11	239	126	114	195	309	195 + 114	217 + 92

TABLE 6. ONE-RUN CYCLES OBTAINED WHEN COMPARING *R. BELLII* WITH SIX OTHER SPECIES OF *RICKETTSIA*

Species (genome B)	*1-run cycles*	*Runs in R. bellii*	*Runs in genome B*
R. felis	85	46	39
R. massiliae	88	55	33
R. africae	86	52	34
R. conorii	86	53	33
R. prowazekii	90	80	10
R. typhi	90	81	9
Identical in all species		27	6

7. ESTABLISHING THE TRIANGLE INEQUALITY

A problem with d_{DCJ}^{id} is that, given any three genomes *A, B* and *C* without duplicated markers, there is no guarantee that the triangle inequality $d_{DCJ}^{id}(A,B) \leq d_{DCJ}^{id}(A,C) + d_{DCJ}^{id}(B,C)$ holds. Consider for example the genomes $A = \{\circ abcde\circ\}$, $B = \{\circ ac\overline{d}be\circ\}$ and $C = \{\circ ae\circ\}$ (Yancopoulos and Friedberg, 2009). In this case, we have $d_{DCJ}^{id}(A,B) = 3, d_{DCJ}^{id}(A,C) = 1$ and $d_{DCJ}^{id}(B,C) = 1$ and the triangle inequality is disrupted.

This happens because, when sorting *A* into *C* and *C* into *B*, a set of common markers of *A* and *B* are deleted and then reinserted (the so-called "free lunch problem") (Yancopoulos and Friedberg, 2009). By definition, our model avoids this problem, since we state that common markers cannot be deleted or inserted. However, this would be a major issue if we want to use the DCJ-indel distance to compute the median of three or more genomes and in phylogenetic reconstructions.

In order to establish the triangle inequality, we borrow some ideas from Yancopoulos and Friedberg (2009) and propose to adopt a surcharge in the DCJ-indel distance. Since we do not want to favor scenarios with more or less indel operations, we propose a surcharge that depends only on the number of unique markers and not on the type of operations applied. Such a correction can be applied *a posteriori*, without interfering with the algorithms to compute the distance and sort genomes presented in the previous sections.

Let *A, B* and *C* be three genomes without duplicated markers and let $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, $\mathcal{D}$, $\mathcal{E}$, $\mathcal{F}$and $\mathcal{G}$ be seven disjoint sets of markers, such that $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ are the sets of unique markers that occur respectively only in *A, B* and *C*. Furthermore, $\mathcal{D}$ is the set of markers that are common only to *A* and *B*, $\mathcal{E}$ is the set of markers that are common only to *B* and *C*, $\mathcal{F}$ is the set of markers that are common only to *A* and *C*, and, finally, $\mathcal{G}$ is the set of markers that are common to *A, B* and *C*. The sets $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, $\mathcal{D}$, $\mathcal{E}$, $\mathcal{F}$ and $\mathcal{G}$ are represented in Figure 9.

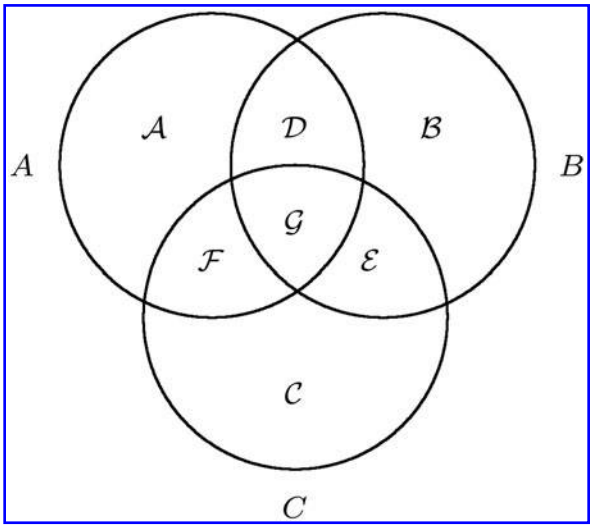

FIG. 9. The disjoint sets $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, $\mathcal{D}$, $\mathcal{E}$, $\mathcal{F}$ and $\mathcal{G}$ for three genomes *A, B*, and *C*--each circle represents the markers that occur in each one of the three genomes.

We define $m(A,B) = d^{id}_{DCJ}(A,B) + k(|\mathcal{A}| + |\mathcal{F}| + |\mathcal{B}| + |\mathcal{E}|)$, where $\mathcal{A} \cup \mathcal{F}$ is the set of markers that occur in A but not in B and $\mathcal{B} \cup \mathcal{E}$ is the set of markers that occur in B but not in A. Analogously, $m(A,C) = d^{id}_{DCJ}(A,C) + k(|\mathcal{A}| + |\mathcal{D}| + |\mathcal{C}| + |\mathcal{E}|)$ and $m(B,C) = d^{id}_{DCJ}(B,C) + k(|\mathcal{B}| + |\mathcal{D}| + |\mathcal{C}| + |\mathcal{F}|)$. Observe that m depends only on the DCJ-indel distance and the number of unique markers and not on how many DCJ or indel operations are used in a sorting sequence. Now, we have to find a positive value k such that $m(A, B) \leq m(A, C) + m(B, C)$ holds. This is equivalent to the inequality

$$\begin{aligned} d^{id}_{DCJ}(A,B) + k(|\mathcal{A}| + |\mathcal{F}| + |\mathcal{B}| + |\mathcal{E}|) \;\leq\; & d^{id}_{DCJ}(A,C) + k(|\mathcal{A}| + |\mathcal{D}| + |\mathcal{C}| + |\mathcal{E}|) + \\ & d^{id}_{DCJ}(B,C) + k(|\mathcal{B}| + |\mathcal{D}| + |\mathcal{C}| + |\mathcal{F}|), \end{aligned}$$

which can be re-written as

$$d^{id}_{DCJ}(A,B) \;\leq\; d^{id}_{DCJ}(A,C) + d^{id}_{DCJ}(B,C) + 2k(|\mathcal{C}| + |\mathcal{D}|).$$

Similarly, for the same value of k, the following inequalities should also hold:

$$\begin{aligned} d^{id}_{DCJ}(A,C) \;&\leq\; d^{id}_{DCJ}(A,B) + d^{id}_{DCJ}(B,C) + 2k(|\mathcal{B}| + |\mathcal{F}|) \text{ and} \\ d^{id}_{DCJ}(B,C) \;&\leq\; d^{id}_{DCJ}(A,B) + d^{id}_{DCJ}(A,C) + 2k(|\mathcal{A}| + |\mathcal{E}|). \end{aligned}$$

Without loss of generality, assume that $d^{id}_{DCJ}(A,B) \geq d^{id}_{DCJ}(A,C)$ and $d^{id}_{DCJ}(A,B) \geq d^{id}_{DCJ}(B,C)$. Then, any $k \geq 0$ guarantees the two last inequalities. We still have to find a positive k such that $d^{id}_{DCJ}(A,B) \leq d^{id}_{DCJ}(A,C) + d^{id}_{DCJ}(B,C) + 2k(|\mathcal{C}| + |\mathcal{D}|)$ also holds. However, if it holds for a certain $k \geq 0$, it would surely hold for any value greater than or equal to k. Observe that, since the set $\mathcal{C}$ does not affect $d^{id}_{DCJ}(A,B)$ and could potentially increase $d^{id}_{DCJ}(A,C)$, $d^{id}_{DCJ}(B,C)$, and the surcharge given by $2k(|\mathcal{C}| + |\mathcal{D}|)$, the worst case would be to have $\mathcal{C} = \emptyset$ and $d^{id}_{DCJ}(A,B) \leq d^{id}_{DCJ}(A,C) + d^{id}_{DCJ}(B,C) + 2k|\mathcal{D}|$. We then have to check two different cases, that are $\mathcal{D} = \emptyset$ and $\mathcal{D} \neq \emptyset$. The first case is covered by the next proposition.

Proposition 11. *Given three genomes A, B and C without duplicated markers, such that A and B have no common marker that does not occur in C, $d^{id}_{DCJ}(A,B) \geq d^{id}_{DCJ}(A,C)$ and $d^{id}_{DCJ}(A,B) \geq d^{id}_{DCJ}(B,C)$, then $d^{id}_{DCJ}(A,B) \leq d^{id}_{DCJ}(A,C) + d^{id}_{DCJ}(B,C)$.*

Proof. We know that $\mathcal{D} = \emptyset$ and, without loss of generality, we can also assume $\mathcal{C} = \emptyset$. Let s_1 be an optimal sequence sorting A into C. The sequence s_1 has some deletions of elements from $\mathcal{A}$, insertions of elements from $\mathcal{E}$ and some DCJs. In the same way, an optimal sequence s_2 sorting C into B has some deletions of elements from $\mathcal{F}$, insertions of elements from $\mathcal{B}$ and also some DCJs. Note that $s_1 s_2$ is a valid sequence sorting A into B (no insertion or deletion of common markers is applied). Thus $|s_1 s_2| \geq d^{id}_{DCJ}(A,B)$, otherwise there would be a valid sequence shorter than $d^{id}_{DCJ}(A,B)$ sorting A into B, which is a contradiction. Since $|s_1| = d^{id}_{DCJ}(A,C)$ and $|s_2| = d^{id}_{DCJ}(B,C)$, we have $d^{id}_{DCJ}(A,C) + d^{id}_{DCJ}(B,C) \geq d^{id}_{DCJ}(A,B)$ ■

In general, as we will see next, the triangle inequality holds for m if we take $k \geq 3/2$.

Proposition 12. *Given $k \geq 3/2$ and any three genomes A, B and C without duplicated markers, such that $d^{id}_{DCJ}(A,B) \geq d^{id}_{DCJ}(A,C)$ and $d^{id}_{DCJ}(A,B) \geq d^{id}_{DCJ}(B,C)$, then $m(A, B) \leq m(A, C) + m(B, C)$.*

Proof. Recall that, to prove that the triangle inequality holds for m, we only need to find a k such that $d^{id}_{DCJ}(A,B) \leq d^{id}_{DCJ}(A,C) + d^{id}_{DCJ}(B,C) + 2k|\mathcal{D}|$ holds. The case in which $\mathcal{D} = \emptyset$ is covered by Proposition 11. It remains to examine the case in which $\mathcal{D} \neq \emptyset$, that is, when A and B have at least one common marker that does not occur in C. Here, the worst case would be to have an empty genome C. Note that in this case we have $\mathcal{C} = \mathcal{E} = \mathcal{F} = \mathcal{G} = \emptyset$. Let X_A, X_B be the number of chromosomes in A and B, L_A, L_B be the number of linear chromosomes in A and B, and S_A, S_B be the number of circular singletons in A and B. Since C is empty, we know that $d^{id}_{DCJ}(A,C) = X_A$ and $d^{id}_{DCJ}(B,C) = X_B$. Moreover, $d_{DCJ}(A,B) \leq |\mathcal{D}|$ and the maximum number of indels between A and B is given by $2|\mathcal{D}| + L_A + S_A + L_B + S_B$ (the circular singletons are obvious indels; furthermore, in each genome we can have one indel per element in $\mathcal{D}$ and one extra indel per linear chromosome). This gives $|\mathcal{D}| + 2|\mathcal{D}| + L_A + S_A + L_B + S_B \leq X_A + X_B + 2k|\mathcal{D}|$. Since $X_A \geq L_A + S_A$ and $X_B \geq L_B + S_B$, we have $3|\mathcal{D}| \leq 2k|\mathcal{D}|$, that holds for any $k \geq 3/2$. ■

As we could see here, when using the DCJ-indel distance to do a pairwise comparison of three genomes without duplicated markers, the triangle inequality can only be disrupted when two genomes have common markers that do not occur in the third genome. In general, a surcharge of at least 3/2 for each unique marker is enough to guarantee the triangle inequality. However, since the value of 3/2 was obtained by an overestimation of the maximum possible DCJ-indel distance between genomes A and B, we conjecture that a surcharge of 1 would be enough to this purpose. It is interesting to emphasize that this correction can be applied *a posteriori*, without interfering with the algorithms to compute the distance and sort genomes presented in the previous sections.

8. CONCLUSION

In this work, we have given a linear time algorithm to compute the distance between two genomes with unequal content, but without duplicated markers, taking into consideration DCJ and indel operations. We also developed algorithms to sort one genome into another one using DCJ and indel operations. We have shown that, in the space of solutions of this problem, the optimal sorting scenarios can have different compositions with respect to the number of each type of operation. We took a first step in the exploration of this solution space proposing approaches that give two different types of sorting scenarios: one that minimizes the number of DCJ operations (respectively maximizes the number of insertions and deletions) and one that minimizes the number of insertions and deletions (respectively maximizes the number of DCJ operations). However, the characterization of the whole space of solutions remains an open problem. With our method we have analyzed a group of bacteria, obtaining preliminary evidence of the occurrence of clusters of deletions in the considered species.

We have also shown that, although the triangle inequality can be disrupted in the DCJ-indel distance, we can correct this problem with a simple function that adds a surcharge to the DCJ-indel distance. Since we do not want to favor scenarios with more or less indel operations, we proposed a surcharge that depends only on the number of unique markers and not on the type of operations applied. We proved that a surcharge of at least 3/2 for each unique marker guarantees the triangle inequality (although we conjecture that a surcharge of 1 would be enough to this purpose). Furthermore, we proved that the triangle inequality can only be disrupted when two genomes have common markers that do not occur in the third genome.

This work opens some perspectives. In addition to the characterization of the whole solution space of the DCJ-indel sorting, one issue that could be addressed next is the incorporation of substitutions in the model, when an insertion and a deletion occur at the same position of the genome.

DISCLOSURE STATEMENT

No competing financial interests exist.

REFERENCES

Andersson, S.G.E., and Kurland, C.G. 1998. Reductive evolution of resident genomes. *Trends Microbiol* 6, 263–268.

Bader, M. 2010. Genome rearrangements with duplications. *BMC Bioinform* 11, Suppl 1, S27.

Bergeron, A., Mixtacki, J., and Stoye, J. 2006. A unifying view of genome rearrangements. *Lect. Notes Comput. Sci.* 4175, 163–173.

Blanc, G., Ogata, H., Robert, C., et al. 2007. Reductive genome evolution from the mother of Rickettsia. *PLoS Genet.* 3, e14.

Braga, M.D.V. 2010. On sorting genomes with DCJ and indels. *Lect. Notes Bioinform.* 6398, 62–73.

Braga, M.D.V., and Stoye, J. 2010. The solution space of sorting by DCJ. *J. Comput. Biol.* 17, 1145–1165.

Braga, M.D.V., Willing, E., and Stoye, J. 2010. Genomic distance with DCJ and indels. *Lect. Notes Bioinform.* 6293, 90–101.

Bryant, D. 2001. The complexity of calculating exemplar distances, 207–212. In Sankoff, D., and Nadeau J.H., eds. *Comparative Genomics.* Kluwer, New York.

El-Mabrouk, N. 2001. Sorting signed permutations by reversals and insertions/deletions of contiguous segments. *J Discrete Algorithms* 1, 105–122.

Hannenhalli, S., and Pevzner, P. 1995. Transforming men into mice (polynomial algorithm for genomic distance problem). *Proc. FOCS 1995* 581–592.

Marron, M., Swenson, K.M., and Moret, B.M.E. 2004. Genomic distances under deletions and insertions. *Theor. Comput. Sci.* 325, 347–360.

Sankoff, D. 1999. Genome rearrangement with gene families. *Bioinformatics* 15, 909–917.

Yancopoulos, S., and Friedberg, R. 2009. DCJ path formulation for genome transformations which include insertions, deletions, and duplications. *J. Comput. Biol.* 16, 1311–1338.

Yancopoulos, S., Attie, O., and Friedberg, R. 2005. Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics* 21, 3340–3346.

Address correspondence to:
Dr. Marília D.V. Braga
Instituto Nacional de Metrologia
Normalização e Qualidade Industrial (Inmetro)
Diretoria de Metrologia Científica (DIMCI)
Divisão de Metrologia em Telecomunicações (DITEL)
Av. Nossa Senhora das Graças, 50
Xerém—Duque de Caxias—RJ—CEP 25250-020, Brazil

E-mail: mdbraga@inmetro.gov.br