Generalizations of the Genomic Rank Distance to Indels*

João Paulo Pereira Zanetti[1][0000−0002−9955−7751], Leonid Chindelevitch[2][0000−0002−6619−6013], and João Meidanis[1][0000−0001−7878−4990]

[1] Institute of Computing
University of Campinas
Av. Albert Einstein, 1251, Campinas SP, Brazil
{joao.zanetti,meidanis}@ic.unicamp.br
[2] School of Computing Science
Simon Fraser University
8888 University Drive, Burnaby BC, Canada
leonid@sfu.ca

Abstract. The rank distance model, introduced by Zanetti *et al.* in 2016, represents genome rearrangements in multi-chromosomal genomes looking at them as matrices. So far, this model only supported comparisons between genomes with the same gene content. We seek to generalize it, allowing for genomes with different gene content. In this paper, we approach such generalization from two different angles, both using the same representation of genomes, and leading to simple distance formulas and sorting algorithms for genomes with different gene contents, but without duplications.

Keywords: Genome rearrangements · Substitutions · Breakpoint graph.

1 Introduction

In the context of genome comparison, one can view a genome as a collection of contiguous, conserved segments arranged in linear and/or circular chromosomes. These segments can be genes or more general markers. Using this abstraction, we pay no attention to smaller mutations affecting just a few nucleotides, and focus instead on bigger mutations that move larger portions of the genome, changing the order of segments with respect to one another. We call these bigger mutations genome rearrangements.

In simpler models of genome rearrangement, the operations only move genomic segments around, without creating or destroying markers. However, to better reflect genome evolution, it is desirable to include operations that alter the content of the genome. For example, we may consider operations that add

* JPPZ is supported by FAPESP grant 2017/02748-3. LC is supported by an NSERC Discovery Grant and a Sloan Foundation Fellowship. JM is supported by FAPESP grant 2018/00031-7.

contiguous segments to the genome, called *insertions*, and operations that remove contiguous segments from the genome, called *deletions*. In general, we call these two types of operation *indels*.

To the best of our knowledge, the work on including indels in genome rearrangement models has so far been limited to the inversion distance [8] for unichromosomal genomes, and the Double-Cut-and-Join (DCJ) distance [14] on multichromosomal genomes [3, 11].

In 2001, El-Mabrouk first studied the problem of sorting by inversions and indels, developing an exact algorithm for the cases where there were only insertions or deletions, but not both [6]. In 2008, Yancopoulos and Friedberg proposed extending the DCJ model to account for insertions and deletions [15], and in 2010 Braga et al. presented a linear time algorithm for the DCJ-Indel problem [4]. Later, Compeau used a different approach, looking at indels as DCJ operations themselves, and arrived at a simpler DCJ-Indel distance formula and sorting algorithm [5]. Another extension of the DCJ model by Braga et al. comes from adding a more powerful operation: a *substitution* of a genome segment for another [2]. The development of DCJ-Indel also led to advancements on the inversion-indel distance, by Willing et al. in 2013 [13].

In this paper, we explore the addition of indels to the rank distance model, which was initially developed for same-content genomes [16]. In this model, genomes are represented as matrices, and the distance between two genomes is the rank of their difference. We expect this model to have a natural extension to genomes with unequal content, leading to simple formulas and algorithms.

The rest of this paper is organized as follows. Section 2 presents the background on the rank distance and defines the representation of genomes that do not necessarily have all the markers being considered. In Section 3 we expand the rank distance to encompass genomes with different genomic content. In Section 4 we present a different approach for adding indels to the rank distance model. Section 5 describes our experiments, and Section 6 presents our conclusions.

2 Definitions

2.1 Markers, Genomes, and Matrices

We begin our definitions with the notion of a *marker*, which is a contiguous DNA stretch that is conserved in all genomes where it appears. This will be our building block in constructing genomes.

Let $\mathcal{G}$ be a set of markers. Each marker $g \in \mathcal{G}$ has two extremities: a head g_h, and a tail g_t. The set

$$V(\mathcal{G}) = \{g_h, g_t | g \in \mathcal{G}\}$$

contains all extremities associated to $\mathcal{G}$. We will fix a 1-1 mapping identifying $V(\mathcal{G})$ with the canonical basis $\{e_1, e_2, \ldots, e_{2n}\}$ of $\mathbb{R}^{2n}$, where $n = |\mathcal{G}|$ and e_i is the $2n \times 1$ column vector whose i^{th} entry is 1 and all others are 0. Since this

mapping is fixed, we will use the same letter and type font to denote both an extremity x and its corresponding column vector.

A genome A over $\mathcal{G}$ consists of a set $V(A) \subseteq V(\mathcal{G})$ of extremities and a set $E(A)$ of *adjacencies*, which are unordered pairs of distinct extremities from $V(A)$, with the extra restriction that each extremity in $V(A)$ can belong to at most one adjacency. Note that a genome does not necessarily contain all the extremities from all the markers in $\mathcal{G}$. We do not even require that $g_t \in V(A)$ if $g_h \in V(A)$, and vice versa. The reason for that will become clear in Section 3.

If a pair $\{x, y\}$ belongs to $E(A)$, we say that x and y are *adjacent* in genome A. From the definitions, we see that each extremity $x \in V(A)$ has to either be adjacent to exactly one other extremity, or be a *free end*, that is, an extremity not adjacent to any other (e.g. near the end of a linear chromosome). In addition, extremities from $V(\mathcal{G})$ that do not belong to $V(A)$ will be called A-*null*, because they will correspond to null rows and columns in the matrix for A, as we will see shortly.

For example, let $\mathcal{G} = \{a, b, c, d\}$, and let A be a genome with $V(A) = \{a_h, b_h, d_h, a_t, b_t, d_t\}$, and $E(A) = \{\{a_h, b_t\}, \{b_h, d_h\}\}$. Genome A is illustrated in Figure 1.

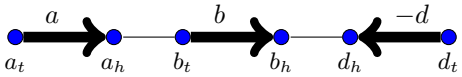

Fig. 1. Genome A with $V(A) = \{a_h, b_h, d_h, a_t, b_t, d_t\}$, and $E(A) = \{\{a_h, b_t\}, \{b_h, d_h\}\}$.

Given that extremities are identified with column vectors of $\mathbb{R}^{2n}$, we may view genomes as matrices as follows. Using the same letter and typeface A to represent the matrix associated to the genome A, we will define:

$$Ax = \begin{cases} y, & \text{when } \{x, y\} \in E(A), \\ x, & \text{when } x \text{ is a free end in } V(A), \\ 0, & \text{when } x \notin V(A). \end{cases}$$

This formula unambiguously define A, since it specifies the image under A of a basis of $\mathbb{R}^{2n}$. As an example, the matrix representation for the genome A in Figure 1 is:

$$\begin{array}{c} \\ a_t \\ a_h \\ b_t \\ b_h \\ c_t \\ c_h \\ d_t \\ d_h \end{array} \begin{array}{c} \begin{array}{cccccccc} a_t & a_h & b_t & b_h & c_t & c_h & d_t & d_h \end{array} \\ \left[\begin{array}{cccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{array}\right] \end{array},$$

where $a_t = e_1$, $a_h = e_2$, ..., $d_h = e_8$. A matrix that can be obtained from a genome in this fashion will be called a *genomic matrix*. It is easy to see that a square binary matrix $A \in \{0,1\}^{2n \times 2n}$ is genomic if and only if $A^T = A$ and A^2 is a diagonal matrix with 0's and 1's on the diagonal; indeed, the 1 entries on the diagonal of A^2 correspond to the extremities present in $V(A)$.

2.2 Rank Distance

Let A and B be two genomic matrices. We can define a distance between them as follows:

$$d_r(A, B) = r(B - A),$$

where $r(X)$ denotes the rank of matrix X. For invertible genome matrices A and B, which do not have zero rows or columns and therefore include all the extremities, this definition generalizes the rank distance of Zanetti et al [16]. This distance satisfies the required properties for a metric:

- $d_r(A, B) = 0 \iff A = B$
- $d_r(A, B) = d_r(B, A)$
- $d_r(A, C) \leq d_r(A, B) + d_r(B, C)$

For example, consider the genome A defined above, and let B be the following genome, illustrated in Figure 2:

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

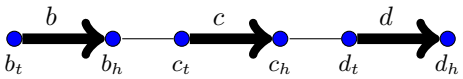

Fig. 2. Matrix and chromosomal representations of a genome B with $V(B) = \{b_h, c_h, d_h, b_t, c_t, d_t\}$, and adjacencies $\{\{b_h, c_t\}, \{c_h, d_t\}\}$.

Having matrices for both A and B on hand, we can compute their difference:

$$B - A = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

And, finally, we have the distance $d_r(A, B) = r(B - A) = 8$. However, computing the rank of the matrix $B - A$ directly is not the most computationally efficient way to compute the rank distance. In Section 3.1 we will see how to do that in $O(n)$ time.

2.3 Augmented Breakpoint Graph

To prepare for the addition of indels to the rank distance model, we defined genomes so that they do not necessarily have the same gene content. We use a structure called the *augmented breakpoint graph*, analogous to the regular breakpoint graph, but, following Compeau [5], with labels at the ends of each path.

The nodes of the augmented breakpoint graph $BG(A, B)$ of A and B are the extremities of the set $V(G) \supseteq V(A) \cup V(B)$, and two nodes x and y are adjacent in $BG(A, B)$ if they are adjacent in either A or B. As in the regular breakpoint graph, all components are either paths or cycles. Sometimes we refer to them as a k-path or a k-cycle when we want to emphasize that k is the number of edges in the path or the cycle.

In the augmented breakpoint graph, all nodes with degree 2 are necessarily in the intersection $V(A) \cap V(B)$, because they are parts of adjacencies in both genomes. On the other hand, nodes x with degree 1 are path endpoints, and at least one of the following cases applies:

- x is a free end in A: $Ax = x$,
- x is a free end in B: $Bx = x$,
- x is A-null: $Ax = 0$,
- x is B-null: $Bx = 0$.

When a path has at least one edge, then it has exactly two distinct end nodes. For each of these two nodes at the ends of the path, exactly one of the cases above apply. When both endpoints are free ends, we call the path *proper*. We say a path is *A-null* (*B-null*) when one of its ends is a free end, and the other is an A-null (B-null) node. When a path has two distinct A-null (B-null) ends, we call the path *AA-null* (*BB-null*). In the case where one end is A-null and the other is B-null, the path is called *AB-null*.

Finally, when a node x has degree zero in $BG(A, B)$, exactly two of the previous cases apply, leading to four possibilities:

- When x is a free end in both A and B, it forms a proper path;
- When x is a free end in A and B-null, it forms a B-null path;
- When x is A-null and a free end in B, it forms an A-null path;
- Finally, when x is null in both A and B, the "natural" definition would be to consider it an AB-null path. However, as we will see in Section 4, for the rank-indel distance it makes more sense to consider this path a proper path. For the rank distance, it makes no difference to consider it as either a proper path or as an AB-null path. We choose to adopt the convention that it is a proper path, to accommodate both versions of the distance.

As an example, Figure 3 is the augmented breakpoint graph $BG(A, B)$ of the genomes A and B seen earlier.

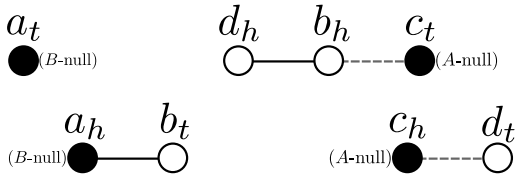

Fig. 3. Augmented breakpoint graph $BG(A, B)$. Black solid edges are adjacencies from A, gray dashed edges are from B. White nodes are extremities in both $V(A)$ and $V(B)$. Black nodes are either A-null or B-null, as specified besides them. The components are two A-null paths and two B-null paths.

Given two genomes A and B, we will define some statistics for $BG(A, B)$. We will use $c(A, B)$ and $p(A, B)$ to denote, respectively, the number of cycles and paths in $BG(A, B)$. The number of paths is the sum of the number of paths of each type: $p_0(A, B)$ is the number of proper paths in $BG(A, B)$, while $p_A(A, B)$, $p_B(A, B)$, $p_{AA}(A, B)$, $p_{BB}(A, B)$ and $p_{AB}(A, B)$ are the number of A-null, B-null, AA-null, BB-null and AB-null paths, respectively.

3 Rank Distance in the Presence of Indels

In this section we discuss the rank distance of genomes with possibly different marker content. First, in Section 3.1, we provide an efficient algorithm to compute the rank distance. Then, in Section 3.2, we define the most concise set of operations needed to transform one genome into another. Finally, in Section 3.3, we show how to use these operations to optimally sort genomes.

3.1 Efficient Computation of the Rank Distance

Given two genomes A and B, we prove the following theorem in the appendix:

Theorem 1.

$$d_r(A, B) = 2n - 2c(A, B) - p_0(A, B) - p_{AB}(A, B).$$

Algorithm 1 (page 7) implements these ideas and runs in $O(n)$ time, efficiently computing $d_r(A, B) = r(A - B)$. It is a Breadth-First Search traversing $BG(A, B)$ that in addition computes a score s for each component, equal to the difference between the number of A-null and B-null extremities in it. Notice that extremities i such that $A[i] > 0$ and $B[i] > 0$ contribute zero to the score. A score of zero means the component has the same number of A-null and B-null extremities, so we decrease d by 1 for a path, or by 2 for a cycle. Since the initial value of d is $2n$, we end up with $d = d_r(A, B)$.

Algorithm 1: Algorithm to compute the distance between genomes A and B. Genome A is given as a list of length $2n$, where $A[i] = j$ if $Ae_i = e_j$, and $A[i] = 0$ if $Ae_i = 0$; similarly for B. The algorithm scores each component in $BG(A, B)$ by comparing the numbers of A-null and B-null extremities. Equal numbers mean the component decreases the distance, by 1 for a path, or by 2 for a cycle.

```
d ← |A|
while ∃x not visited do
    Q ← {x}
    s ← 0
    mark x as visited
    while Q ≠ ∅ do
        take i from Q
        if both A[i] and B[i] are > 0, ≠ i, and visited then
            d ← d − 1
        if A[i] > 0 then
            s ← s + 1
            if A[i] not visited then
                mark A[i] as visited
                add A[i] to Q
        if B[i] > 0 then
            s ← s − 1
            if B[i] not visited then
                mark B[i] as visited
                add B[i] to Q
    if s = 0 then
        d ← d − 1
return d
```

3.2 Basic Operations

A matrix X is an *operation* when there is a genome A such that $A + X$ is a genome. In this case we say that X is *applicable* to A. The *weight* of an operation X is the rank of X.

When the genomes considered have the same marker content, only three types of operations are needed to sort any genome into another: cuts, joins and double swaps [10]. Cuts and joins have weight 1, while double swaps have weight 2. In this paper, we seek to add to our model operations that deal with unequal gene content. This turns out to be a highly non-trivial task, as we explain in more detail in the appendix. Here we summarize our final model.

As expected, we need to consider the insertion or the deletion of an entire chromosome. Insertions and deletions of parts of chromosomes are not needed, as we show in the appendix. The matrix for the insertion or deletion of a chromosome with k markers is, up to the sign, equivalent to a genome with k markers, and always has weight $2k$. Therefore, the weight of such an operation is $2k$.

In addition, we consider a new kind of operation that takes advantage of our relaxed definition of genomes. Recall that, when we defined genomes in Section 2, we mentioned that, given a genome A, we do not require that $g_t \in V(A)$ if $g_h \in V(A)$, or vice-versa. This relaxed definition now comes into play. We define an operation that substitutes a single extremity for an extremity that does not exist in the genome; due to its rank, we assign such an operation a weight of 2.

Introducing this kind of operation implies that the concept of chromosomes also has to be relaxed. In a genome where, for every $g \in \mathcal{G}$, the extremities g_h and g_t are either both present or both absent, a chromosome is a sequence of markers that can be either circular, having no free ends, or linear, with exactly two free ends. In the case of a genome with only one extremity of a marker, there are *semi-chromosomes* that, instead of ending at a free end, end with an unpaired extremity, that is, a head extremity whose corresponding tail is not in the genome, or vice versa. As a result, now an insertion or a deletion can be of a whole chromosome, or of a whole semi-chromosome, always with a weight equal to the number of extremities being inserted or deleted.

With the introduction of extremity substitutions, we now have six types of basic operations:

- Cut, with cost 1.
- Join, with cost 1.
- Double swap, with cost 2.
- Deletion of whole chromosomes or semi-chromosomes, costing the number of extremities deleted.
- Insertion of whole chromosomes or semi-chromosomes, costing the number of extremities inserted.
- Substitution of one extremity, with cost 2.

From here to the end of Section 3, we call these the *basic operations* to transform one genome into another when they do not share the same set of markers. We will see how the basic operations suffice to carry out such transformations.

3.3 Sorting

Let $\mathcal{X} = (X_1, X_2, \ldots, X_k)$ be a sequence of operations such that, for every $1 \leq i \leq k$, the operation X_i is applicable to $A + X_1 + \ldots + X_{i-1}$, and $A + X_1 +$

$\ldots + X_k = B$. We say that $\mathcal{X}$ is a *sorting scenario* from A to B. The *weight* of $\mathcal{X}$ is the sum of the weights of its operations, that is,

$$w(\mathcal{X}) = \sum_{i=1}^{k} r(X_i).$$

We denote by $w(A, B)$ the minimum weight of a sorting scenario from A to B. When a scenario $\mathcal{X}$ from A to B satisfies $w(\mathcal{X}) = w(A, B)$, we call $\mathcal{X}$ *optimal*.

In the appendix we show that the rank distance $d(A, B)$ is equal to the optimum weight of a scenario going from A to B using the basic operations described in Section 3.2. The main result proved there is the following theorem:

Theorem 2. *Given two genomes A and B,*

$$d_r(A, B) = w(A, B).$$

Triangle inequality One of our main concerns with the addition of indel operations to a genomic distance is respecting the triangle inequality. When indels have a constant cost, the triangle inequality is easily violated. Consider the three genomes in Figure 4. Only one deletion is needed to transform either A or B into C, but rearranging A into B takes more operations.

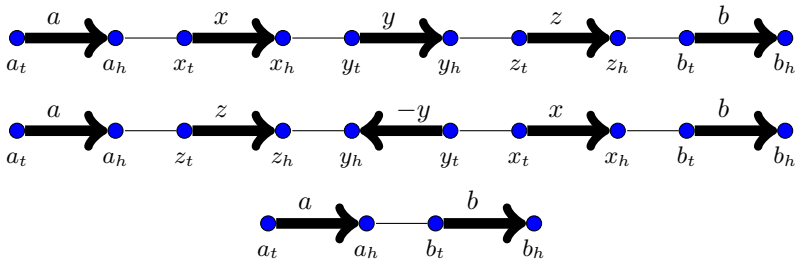

Fig. 4. Example of genomes A, B, C that can violate the triangle inequality.

Yancopoulos and Friedberg call this violation "the free lunch problem" [15]. Their suggestion to deal with this problem is to add a surcharge to the cost of an indel, based on the adjacency graph.

Braga et al. dealt with the violation of the triangle inequality by adding a simpler surcharge after the computation of their DCJ-indel distance [4]. The surcharge to the distance between A and B is equal to $ku(A, B)$, where k is a constant, and $u(A, B)$ is the number of unique markers between genomes A and B. At first they showed the triangle inequality holds when $k \geq 3/2$, but later work showed that $k \geq 1$ is a tight bound [1].

Braga et al. also defined a framework to assign variable costs to indels, a linear function of the number of markers inserted or deleted, and showed that it is equivalent to the *a posteriori* surcharge [1].

Unlike the DCJ distance, the rank distance, with the proposed extension to the matrix representation of genomes, naturally offers an indel mechanism with weights that avoid the free lunch problem.

4 An Alternative: the Rank-Indel Distance

In order to avoid general extremity substitutions and genomes without both extremities of a marker, a different approach to the addition of indels to the rank distance is to define a genomic distance that includes the basic operations of the rank distance for genomes with the same content, plus insertions and deletions, all with the same weight as in the rank distance model. This way, we define the *rank-indel distance* $d_i(A, B)$ of A and B as the minimum cost of an operation sequence sorting A into B, using the basic operations cited above:

- Cuts/joins, with cost 1.
- Double swaps, with cost 2.
- Insertions/deletions of linear or circular chromosomes with k markers, costing $2k$.

We already know that $d_i(A, B) \geq r(B - A)$. This inequality can sometimes be strict. In fact, we prove the following theorem in the Appendix:

Theorem 3. $d_i(A, B) = 2n - 2c(A, B) - p_0(A, B) + p_{AB}(A, B)$.

5 Experiments

We performed an experiment to assess how much evolutionary signal the rank and rank-indel distances are capable of capturing. We used fungal genomes from 13 species: the causal agent of rice blast, *Magnaporthe oryzae*, plus 12 isolates of *Magnaporthe grisea*, a related pathogen, collected from infected wheat plants. The data was kindly made available by A. Nhani Jr, N. Talbot, and D. Soanes.

The 12 isolates were sequenced, assembled, annotated, and mapped onto the complete genome of *M. oryzae*. The annotated genes of *M. oryzae* then provided a gene set containing all the genes in the annotated genomes of the isolates, and serve as our comprehensive marker set $\mathcal{G}$ in this analysis. The resulting `.gff` files were used to determine gene position and orientation of each gene in each genome. We realize that this procedure leaves out genes unique to the isolates, not present in *M. oryzae*, but, even with this bias, the results were encouraging.

Each `.gff` file contains genes assembled in 7 chromosomes, plus an extra chromosome numbered "8" where small pieces that could not be placed anywhere else during assembly were kept. In our analysis, we decided to only consider the first 7 chromosomes, as they correspond to genes that could be effectively placed onto a real chromosome. In addition, the mapping process produces a similarity coefficient, varying from 0.0 to 1.0, that tells how similar the isolate and *M. oryzae* genes are. In our analysis, we only kept the genes for which this

coefficient is 1.0, meaning very high similarity. With this, we lost about 7% or all the genes. Finally, we filtered out any coding sequence (CDS) properly contained in a larger CDS, as these are likely annotation artifacts.

With the position and orientation of all genes, we built the adjacencies in each genome and input the results to our rank and rank-indel distance calculating algorithms. Table 1 contains the pairwise distances obtained. It is interesting to observe that for all pairs, the rank distance coincided with the rank-indel distance, *meaning that there are no AB-paths between any pair*. We built a phylogenetic tree with these distances using the neighbor-joining method [12], as implemented in the Mega software package [9].

Table 1. Pairwise rank and rank-indel distances (they are equal) between 13 fungal genomes. *M. oryzae* is denoted by M; other isolates by their numeric ID. Notice that the distances involving M are much larger, consistent with the fact that it infects rice while all others infect wheat.

	5033	0925	36	5003	6047	86	25	35	6017	5010	M	6045	5035
5033	0	239	332	285	378	429	361	344	185	436	13111	299	170
0925	239	0	277	323	414	444	342	307	264	417	13083	270	245
36	332	277	0	451	282	553	285	126	357	508	13126	211	316
5003	285	323	451	0	369	566	436	413	305	549	13167	419	273
6047	378	414	282	369	0	649	307	230	370	618	13215	322	348
86	429	444	553	566	649	0	499	587	452	321	13038	538	461
25	361	342	285	436	307	499	0	313	338	418	13093	242	333
35	344	307	126	413	230	587	313	0	375	558	13150	229	326
6017	185	264	357	305	370	452	338	375	0	463	13108	272	143
5010	436	417	508	549	618	321	418	558	463	0	13101	483	460
M	13111	13083	13126	13167	13215	13038	13093	13150	13108	13101	0	13115	13111
6045	299	270	211	419	322	538	242	229	272	483	13115	0	249
5035	170	245	316	273	348	461	333	326	143	460	13111	249	0

A larger set of blast pathogens was analysed by Gladieux *et al.* in a recent article [7], where phylogenetic trees were constructed based on several factors, including maximum likelihood on almost 3,000 orthologous CDS, maximum likelihood on 9 loci, and pairwise BLAST distances between repeat-masked genomes. Restricting the trees to just the common genomes, we verify that our trees only differ in the positioning of the PY6045-PY36 subtree (Fig. 5).

6 Conclusion

In this paper, we expanded the rank distance to account for genomes with different gene content, but still without duplications. The first step, in Section 2, was to define genomes that do not necessarily contain all markers of $\mathcal{G}$. This allows for the representation of genomes with different markers from each other, and is done very naturally, by using zeros in the rows/columns corresponding to the missing markers. We then developed ways to compare these genomes.

The first approach simply extends the rank distance, keeping the distance $d_r(A, B)$ between two genomes A and B equal to the rank $r(A - B)$ of their

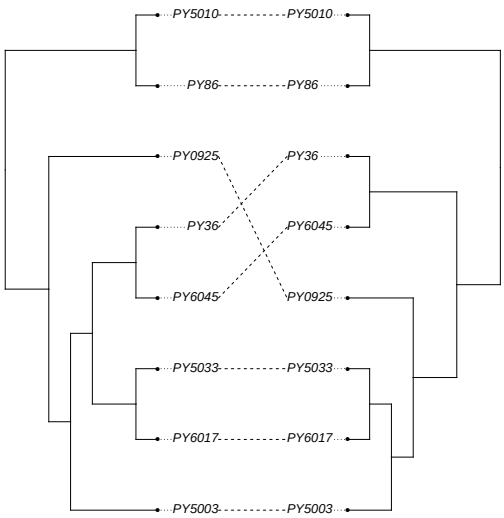

Fig. 5. Phylogenetic trees for the same genome set constructed with our distances and neighbor-joining (left) and with more traditional methods (right). Notice that the trees only differ by the placement of the PY6045-PY36 subtree.

difference. We showed how to efficiently compute d_r, and how to transform A into B using only basic operations, adding insertions, deletions, and the substitution of a single extremity to the cast of basic operations of the rank distance of genome with the same markers.

The substitution of single extremities leads to genomes with incomplete markers. To avoid this, we also present an alternative rank-indel distance that changes the content of a genome only through insertions and deletions of chromosomes. We note that both distances have very simple formulas, and are closely related, with $d_i(A, B) = d_r(A, B) + 2p_{AB}(A, B)$.

Phylogenetic trees constructed with our distances turn out to be very close to trees built with other, more traditional methods, showing that there is enough phylogenetic signal in the order and orientation of genes alone. Further studies will be conducted to better assess the usefulness of these distances in phylogenetics.

Acknowledgments

We would like to thank the EMBRAPA Multiuser Bioinformatics Laboratory (Laboratório Multiusuário de Bioinformática da Embrapa) for helpful discussions, Dr João Leodato Nunes Maciel for the isolates data, and Prof. Dr. Nicholas J. Talbot (University of Exeter, Devon - UK) for the genome sequencing and assembly infrastructure.

References

1. Braga, M., Machado, R., Ribeiro, L., Stoye, J.: On the weight of indels in genomic distances. BMC Bioinformatics **12** (2011). https://doi.org/10.1186/1471-2105-12-S9-S13
2. Braga, M.D.V., Machado, R., Ribeiro, L.C., Stoye, J.: Genomic distance under gene substitutions. BMC Bioinformatics **12**(Suppl 9), S8 (October 2011). https://doi.org/10.1186/1471-2105-12-S9-S8
3. Braga, M.D.: An overview of genomic distances modeled with indels. In: Conference on Computability in Europe. pp. 22–31. Springer (2013)
4. Braga, M.D., Willing, E., Stoye, J.: Double cut and join with insertions and deletions. Journal of Computational Biology **18**(9), 1167–1184 (2011)
5. Compeau, P.E.C.: DCJ-Indel sorting revisited. Algorithms for Molecular Biology **8**(1), 6 (2013). https://doi.org/10.1186/1748-7188-8-6
6. El-Mabrouk, N.: Sorting signed permutations by reversals and insertions/deletions of contiguous segments. Journal of Discrete Algorithms **1**(1), 105–122 (2001)
7. Gladieux, P., Condon, B., Ravel, S., Soanes, D., Maciel, J.L.N., Nhani, A., Chen, L., Terauchi, R., Lebrun, M.H., Tharreau, D., Mitchell, T., Pedley, K.F., Valent, B., Talbot, N.J., Farman, M., Fournier, E.: Gene flow between divergent cereal- and grass-specific lineages of the rice blast fungus *Magnaporthe oryzae*. mBio **9**(1) (2018). https://doi.org/10.1128/mBio.01219-17
8. Hannenhalli, S., Pevzner, P.A.: Transforming cabbage into turnip: Polynomial algorithm for sorting signed permutations by reversals. J. ACM **46**(1), 1–27 (Jan 1999). https://doi.org/10.1145/300515.300516
9. Kumar, S., Stecher, G., Li, M., Knyaz, C., Tamura, K.: MEGA X: Molecular evolutionary genetics analysis across computing platforms. Molecular biology and evolution **35**(6), 1547–1549 (2018)
10. Meidanis, J., Biller, P., Zanetti, J.P.P.: A Matrix-Based Theory for Genome Rearrangements. Tech. Rep. IC-17-11, Institute of Computing, University of Campinas (August 2017), in English, 45 pages.
11. Paten, B., Zerbino, D.R., Hickey, G., Haussler, D.: A unifying model of genome evolution under parsimony. BMC bioinformatics **15**(1), 206 (2014)
12. Saitou, N., Nei, M.: The neighbor-joining method: a new method for reconstructing phylogenetic trees. Molecular biology and evolution **4**(4), 406–425 (1987)
13. Willing, E., Zaccaria, S., Braga, M.D., Stoye, J.: On the inversion-indel distance. In: BMC bioinformatics. vol. 14, p. S3. BioMed Central (2013)
14. Yancopoulos, S., Attie, O., Friedberg, R.: Efficient sorting of genomic permutations by translocation, inversion and block interchange. Bioinformatics **21**(16), 3340–3346 (2005)
15. 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)
16. Zanetti, J.P.P., Biller, P., Meidanis, J.: Median approximations for genomes modeled as matrices. Bulletin of Mathematical Biology **78**(4), 786–814 (2016)