Counting Sorting Scenarios and Intermediate Genomes for the Rank Distance*

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. An important problem in genome comparison is the genome sorting problem, that is, the problem of finding a sequence of basic operations that transforms one genome into another whose length (possibly weighted) equals the distance between them. These sequences are called optimal sorting scenarios. However, there is usually a large number of such scenarios, and a naïve algorithm is very likely to be biased towards a specific type of scenario, impairing its usefulness in real-world applications. One way to go beyond the traditional sorting algorithms is to explore all possible solutions, looking at all the optimal sorting scenarios instead of just an arbitrary one. Another related approach is to analyze all the intermediate genomes, that is, all the genomes that can occur in an optimal sorting scenario. In this paper, we show how to count the number of optimal sorting scenarios and the number of intermediate genomes between any two given genomes, under the rank distance.

Keywords: Genome rearrangements · Enumeration · Breakpoint graph.

1 Introduction

At a high level of abstraction, genomes can be represented as a list of blocks (that can be genes, markers, or other syntenic regions), and their evolution can be modeled by large-scale mutation events we call *genome rearrangements*. Different genome rearrangement models define different events and costs (weights) for them, and their most basic application is to determine the lowest cost required to transform one genome into another. This is the *genome distance problem*. Although this definition does not require it, in this paper we make the assumption

* 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.

that the genomes being considered have the same gene content. Consequently, as the rearrangement events modeled here do not alter the content of genomes, we do not model loss or duplication of blocks. Although there are models that include insertion and deletion events [3, 5] and duplications [8], we do not include them here, instead focusing only on events that alter the order of the blocks.

Another fundamental problem that applies to a genome distance is to find sequences of rearrangement operations with minimum cost. We call this the *genome sorting problem*. However, a single arbitrary sequence of events among many optimal ones is hardly representative of the evolutionary process, especially considering that sorting algorithms might be biased towards certain kinds of sorting sequences. On the other hand, listing all possible optimal scenarios is not practical, because their number is simply too large.

A first step towards exploring the solution space of the genome sorting problem as a whole is to count the number of optimal sorting scenarios between two given genomes. This can reveal helpful patterns in the optimal solutions. The sorting scenario as a sequence of operations taking one genome into another is not the only way to represent a solution to the sorting problem. Shao, Lin, and Moret suggested a structure called the trajectory graph, which groups together scenarios with commuting or non-interfering operations [20].

Larget, Kadane, and Simon enumerated possible inversion scenarios in circular unichromosomal genomes in order to compute probability distributions of ancestral genome rearrangements [14]. For multi-chromosomal genomes, to the best of our knowledge, the studies of the solution space of genome sorting have been limited to the Double-Cut-and-Join (DCJ) [23] and the Single-Cut-or-Join (SCJ) [9] distances. For the latter model, Miklós, Kiss, and Tannier showed how to count optimal SCJ scenarios between two genomes [16]. In the DCJ model, the problem becomes more complex due to the possibility of recombinations. Braga and Stoye showed how to count optimal DCJ scenarios, with and without recombination [4]. Ouangraoua and Bergeron also counted optimal scenarios without recombination, and established bijections between these scenarios and known combinatorial objects [17]. Feijão took these results one step further, and showed how to count the intermediate genomes between two genomes and how to use those to reconstruct ancestral genomes [7]. Here, we produce an analogous set of results for the rank distance model [24] instead of the DCJ model.

The rank distance, introduced by Zanetti, Biller, and Meidanis [24] is based on a representation of genomes using matrices. The rank distance is twice the algebraic distance [10], and very close to the DCJ distance [23, 10]. By using the rank distance, we can add linear algebra tools to the traditional breakpoint graph analysis techniques usually employed in rearrangement studies. The rank distance is equivalent to the DCJ for genomes with the same free ends. One advantage of the rank distance is that it ensures that there is no recombination while sorting. On the other hand, since the basic operations have different weights, the scenarios have variable lengths, and this represents a challenge when counting all possible scenarios. Fortunately, we were able to overcome this difficulty by adding an extra parameter in the recurrence relationship for the counts.

The rest of this paper is organized as follows. Section 2 introduces the background concepts we use: the rank distance, its basic operations, and the multi-genome breakpoint graph. In Section 3 we show how to count the number of optimal scenarios between two genomes in the rank distance. In Section 4 we count the number of intermediate genomes. In Section 5 we discuss our experiments on real genomic data. Finally, we summarize our work in Section 6. Most proofs, details, and extra material are contained in the Appendix (http://www.ic.unicamp.br/∼meidanis/research/rear/).

2 Background

In this section we give the theoretical background needed to explore the solution space in the next sections. First, we present the basic concepts of the rank distance model for comparing genomes. Next, we discuss the breakpoint graph and how it relates to the rank distance. Finally, we show the three basic operations necessary and sufficient for sorting genomes under the rank distance.

2.1 Genomes as Matrices and the Rank Distance

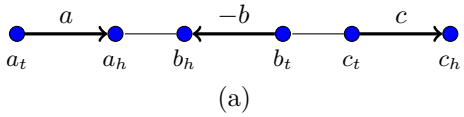
In this subsection we introduce our representation of genomes as matrices, the rank distance, and its basic operations.

We define genomes as a collection of *chromosomes*, each of them a linear or circular sequence of *genes*. A gene is a linear segment with two *extremities*: a *tail* and a *head*. When two genes appear consecutively in a chromosome, we indicate this fact by linking their closest extremities with an *adjacency*. Adjacencies are thus unordered pairs of extremities. An extremity may not be involved in more than one adjacency. If an extremity is not in any adjacency, it is called a *free end*. A genome is completely characterized by its adjacencies and free ends. Figure 1 shows an example of a genome with three genes.

We can encode this representation of a genome into a matrix [24]. In this paper we focus on comparing genomes with the same gene content. To represent genomes with matrices, we first fix an ordering of all the extremities, and use this ordering for the rows and the columns of a matrix. For instance, in Figure 1, there are three genes, a, b, and c, and the ordering is a_t, a_h, b_t, b_h, c_t, c_h. After fixing the order of the extremities (and hence the meaning of the rows and columns of a matrix), we define the corresponding *genome matrix* as follows:

$$A_{ij} = \begin{cases} 1 & \text{if } i \neq j \text{ and } i, j \text{ are adjacent in } G, \text{ or } i = j \text{ and } i \text{ is a free end in } G \\ 0 & \text{otherwise} \end{cases}$$

An example of a genome matrix can be seen in Figure 1. These matrices have the following properties: they are symmetric, that is, $A^T = A$, orthogonal, that is, $A^T = A^{-1}$, and binary, that is, $A \in \{0, 1\}^{2n \times 2n}$, where n is the number of genes. In particular, this implies that they are involutions, i.e. permutations of order 1 or 2. These matrices are therefore sparse, and can be stored efficiently.



(a)

	a_t	a_h	b_t	b_h	c_t	c_h
a_t	1	0	0	0	0	0
a_h	0	0	0	1	0	0
b_t	0	0	0	0	1	0
b_h	0	1	0	0	0	0
c_t	0	0	1	0	0	0
c_h	0	0	0	0	0	1

(b)

Fig. 1. (a) Genome with one linear chromosome, and adjacencies $\{a_h, b_h\}$ and $\{b_t, c_t\}$. The extremities a_t and c_h are free ends. (b) Matrix representation of the same genome.

If A and B are genomes over the same genes, the *distance* $d(A, B)$ between A and B is defined as

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

where r is the rank of a matrix.

Although the rank distance is defined in terms of matrices, it is also possible to characterize it as the weight of an optimal series of operations transforming A into B. We say that a matrix X is *applicable* to a genome A if $A + X$ is also a genome. A matrix applicable to at least one genome is called an *operation*.

An operation is *basic* if it is a *cut* of an adjacency $\{x, y\} \to \{x\}\{y\}$, a *join* of two free ends $\{x\}\{y\} \to \{x, y\}$, or a *double swap* of two adjacencies into two new ones using the same four extremities, for example $\{x, y\}\{a, b\} \to \{x, a\}\{y, b\}$. Any other rearrangement operation can be decomposed as a sum of these three kinds of operations, as we show in Section 2.3. We are thus able to narrow down the list of operations to just three, without loss of generality. In the context of this work, the most important information about the basic operations is that cuts and joins are rank 1 matrices, while double swaps have rank 2.

2.2 Breakpoint Graph

Genomes, as we describe here, are matchings over the set of gene extremities. We can graphically represent two genomes A and B as matchings, using one color (or line style) for A and another for B, as shown in Figure 2.

The graph we use, called *breakpoint graph* [22] is based on the original breakpoint graph introduced by Hannenhalli and Pevzner in 1995 [11]. The difference is that we do not use caps at chromosome ends. Our free ends are simply extremities that are not adjacent to any other extremity. We believe that this makes the presentation simpler and clearer.

Given two genomes A and B of equal gene content, we build the two-genome breakpoint graph $BG(A, B)$ as follows. Its vertices are the extremities of the genomes, and there are two sets of edges: dashed edges, which connect pairs of extremities that are adjacent in A, and solid edges, which connect pairs of extremities that are adjacent in B.

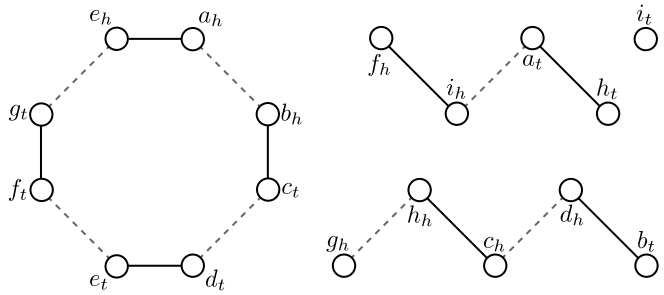


Fig. 2. Example breakpoint graph $BG(A, B)$. Genome A has adjacencies $a_h b_h$, $c_t d_t$, $e_t f_t$, $e_h g_t$, $a_t i_h$, $g_h h_h$, and $c_h d_h$, drawn as dashed edges, and free ends b_t, f_h, h_t, and i_t. Genome B has adjacencies $a_h e_h$, $b_h c_t$, $d_t e_t$, $f_t g_t$, $f_h i_h$, $a_t h_t$, $c_h h_h$, and $b_t d_h$, drawn as solid edges, and free ends g_h and i_t.

Every node in the breakpoint graph has degree 0, 1, or 2. Because of this, the connected components of the graph are disjoint cycles and paths. The paths can be further classified into two types: paths with an even or odd number of edges. The breakpoint graph is very similar to another structure widely used in rearrangement analysis, the adjacency graph [1]. In fact, the breakpoint graph is the line graph of the adjacency graph [22]. Furthermore, as the adjacency graph has no isolated vertices, both graphs have the same number of paths and cycles.

The process of *sorting* consists of the application of basic operations to genome A until we get genome B. When sorting from A to obtain B, we call A the *source genome*, and B the *target genome*. In the graph $BG(A, B)$, we draw the edges from the source genome A with dashed lines, and the edges from the target genome B with solid edges.

Paths with an even number of edges are called *balanced*, because they have the same number of edges of each type. Odd paths are called *unbalanced*; they begin and end with edges from the same genome. Unbalanced paths can be further classified into two types, *dashed paths*, and *solid paths*, according to which genome accounts for more edges.

Note that when both genomes are equal, the edges of both colors coincide, leaving only two types of components, cycles with two edges (shared adjacencies), and isolated vertices (shared free ends). Therefore, considering $BG(A, B)$, sorting A into B can be seen as the process of applying basic operations to the dashed edges until they all coincide with the solid edges.

A cut either transforms a cycle into a path, or splits one path into two. A join does the reverse: it either transforms a path into a cycle, or joins two paths into one. A double swap can extract a cycle from any type of component or reverse a part of a component. When acting on two separate components, a double swap can also insert a circular component into another (linear or circular) one or swap the end segments of two paths.

We call cycles with four or more edges *big cycles* and paths with at least one edge *big paths*. When we refer to a component that is either a big cycle or a big

path, we call it a *big component*. These components need to be worked on in order to sort the source genome into the target one. Cycles with two edges and paths of length zero (isolated vertices) will be called *small components*. The next result shows how to compute the rank distance using the breakpoint graph.

Theorem 1. *Given two genomes A and B over the same set of extremities, the rank distance between them is given by*

$$d(A, B) = 2n - 2c - p,$$

where n is the number of genes, and c and p are respectively the number of cycles and paths in $BG(A, B)$

Proof. Feijão and Meidanis in 2013 showed that the algebraic distance $d_{alg}(A, B)$ between two genomes A and B is given by $d_{alg}(A, B) = n - n_C - \frac{n_P}{2}$, where n is the number of genes, and n_C and n_P are respectively the number of cycles and paths in the adjacency graph of A and B [10, Theorem 3.11]. Since the breakpoint graph and the adjacency graph have the same number of cycles and paths, we have $c = n_C$ and $p = n_P$. Later, Zanetti, Biller and Meidanis showed that $d(A, B) = 2d_{alg}(A, B)$ [24], and, therefore, $d(A, B) = 2n - 2c - p$. □

2.3 Sorting

In this subsection we show that is possible to sort genome A into genome B using only the three basic operations: cut, join, and double swap.

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 $w(A, B)$ the minimum weight of a sorting scenario from A to B. When a scenario $\mathcal{X}$ from A to B has $w(\mathcal{X}) = w(A, B)$, we say $\mathcal{X}$ is *optimal*. Any operation in an optimal scenario is called a *sorting operation*.

A series of lemmas leads to the desired results. We only state the main conclusion here. All the supporting proofs can be found in the appendix.

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

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

3 Counting the Number of Scenarios

An optimal solution for sorting genome A into genome B is a sequence of genomes separated by basic operations, going from A to B with minimum cost. Some

authors call such a sequence a *geodesic* between A and B, or a *geodesic patch* when the basic operations have different weights [12]. Braga and Stoye present a similar definition for *sorting scenarios* in their work, as a sequence of operations involved in sorting A into B. We define an *operation scenario* from A to B as a list of matrices $\mathcal{L} = [X_1, \ldots, X_\ell]$ such that $A+X_1+X_2+\ldots+X_\ell = B$, and each matrix X_i is one of the basic operations and applicable to $A+X_1+X_2+\ldots+X_{i-1}$. The *total weight* of a scenario, denoted by $w(\mathcal{L})$, is the sum of the weights of all its operations. Such a scenario is *optimal* if and only if $w(\mathcal{L}) = d(A, B)$. An *optimal operation* is any basic operation that is the first in an optimal scenario. In the context of the rank distance, optimal scenarios are geodesic patches, because basic operations can have weight 1 or 2.

In this section, we show how to count the number of optimal rank sorting scenarios between two genomes, with the aid of the breakpoint graph. First, we show that no optimal operation acts on different components of the breakpoint graph. Therefore, we can solve each component separately. We then recall a formula from Braga and Stoye [4] to count DCJ sorting scenarios, which also applies to rank sorting scenarios for cycles, and a recurrence to count rank sorting scenarios for paths. Finally, we show how to get the count for the whole graph.

3.1 Recombination

As a first step to count the number of sorting scenarios, we want to prove a useful property, namely, that no optimal rearrangement recombines the components of the breakpoint graph. In other words, we show that no optimal operation acts on the extremities of more than one component at the same time.

Lemma 3. *No optimal operation in sorting from A to B involves extremities in different components of $BG(A, B)$.*

Proof. To prove this, we list every possible operation recombining two components C_1 and C_2 of $BG(A, B)$ and show that they do not change the graph in a way that reduces the distance.

A double swap can be applied to two cycles, generating one cycle; it can be applied to a cycle and a path, generating a path; or it can be applied to two paths, resulting in two paths. All of these moves reduce the number of components or keep their number unchanged, and are therefore not optimal.

The other option of a basic operation on two separate components is a join of two paths; however, this results in a single path, reducing the number of components, and is therefore also not optimal.

A cut is not considered here, because it only acts on two connected extremities. Therefore, it never affects more than one component. □

According to Braga and Stoye, under the DCJ model the only operation that recombines different components of the graph is a double swap between two unbalanced paths resulting in two balanced ones [4]. With regard to the breakpoint graph, the difference between the DCJ and the rank distances is

that, under the rank distance, every path counts towards reducing the distance, not just balanced ones. Because of this difference, recombining two unbalanced paths into two balanced ones is not an optimal move in a rank sorting scenario.

With this result, we conclude that it is possible to count the optimal scenarios for each component in the breakpoint graph independently. Now we determine how to obtain sorting scenarios for the whole graph from the separate solutions for each component.

Let s_1 and s_2 be scenarios for the components C_1 and C_2 respectively, with respective lengths ℓ_1 and ℓ_2. From Lemma 3, the number of scenarios resulting in the combination of s_1 and s_2 is the number of sequences that have both as subsequences, and this is the shuffle product of s_1 and s_2, whose size is given by the binomial coefficient $\binom{\ell_1+\ell_2}{\ell_1,\ell_2} = \frac{(\ell_1+\ell_2)!}{\ell_1!\ell_2!}$. In general, the number of sequences obtained by shuffling k subsequences is given by the multinomial coefficient $\binom{\ell_1+\ell_2+\ldots+\ell_k}{\ell_1,\ell_2,\ldots,\ell_k} = \frac{(\ell_1+\ell_2+\ldots+\ell_k)!}{\ell_1!\ell_2!\ldots\ell_k!}$, where ℓ_i is the length of the ith subsequence.

3.2 Cycles

Given a cycle in $BG(A, B)$, the only optimal operation that can be applied to it is a double swap that splits the cycle into two smaller ones, as illustrated in Figure 3. This is a rank-two operation that increases the number of cycles by one, therefore (by Theorem 1) decreasing the distance by two.

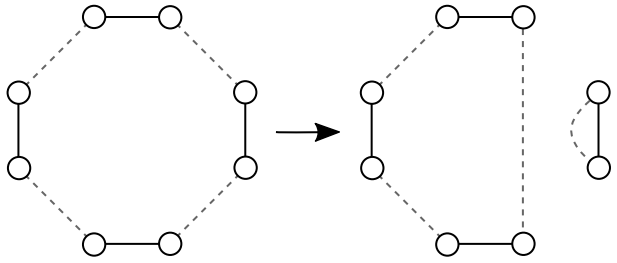


Fig. 3. One example of optimal rearrangement for a cycle. A double swap splits the cycle into two smaller ones. Here, an 8-cycle is decomposed into a 6-cycle, which will require two more double swaps to complete the sorting, and a 2-cycle, already sorted.

In this case, the sorting moves are equivalent to the ones for the DCJ distance. We can use the same formula for DCJ to compute the number $S_c(2\ell + 2)$ of scenarios to solve a cycle of length $2\ell + 2$ [4, Theorem 3]:

$$S_c(2\ell + 2) = (\ell + 1)^{(\ell-1)}$$

Each of these $S_c(2\ell+2)$ scenarios has length ℓ, and total weight 2ℓ. When A and B are co-tailed genomes (that is, A and B have the same free ends), the only big components in $BG(A, B)$ are cycles, and we can compute the total number

S_{ct} of optimal sorting scenarios from A to B [4, Theorem 4] as follows:

$$S_{ct} = \frac{(\ell_1 + \ell_2 + \ldots + \ell_p)!}{\ell_1!\ell_2!\ldots\ell_p!} \prod_{i=1}^{p} (\ell_i + 1)^{\ell_i - 1},$$

where p is the number of big cycles in $BG(A, B)$.

3.3 Paths

We have shown a simple formula to compute the number of scenarios for the cyclic components in $BG(A, B)$. For paths the computation is less straightforward. The obstacle that arises when sorting paths is that, because cuts and joins have rank 1, while double swaps have rank 2, scenarios have variable length, and information on the length of the sub-solutions is necessary for the shuffling. Thus, we need a recurrence with two variables: the length of the path, and the length of the scenario.

For a path, three optimal operations are possible. First, a cut of any dashed edge, provided there is at least one dashed edge in the path. Such a cut results in two smaller paths that are solved separately. A second option is to execute a double swap on any two dashed edges, resulting in a cycle and a path, provided there are at least two dashed edges. Here again, both new components have independent scenarios that are then shuffled. The last option is to join the ends of a path, if both ends are incident to solid edges. This leads to a single cycle, that we already know how to process. These possibilities are illustrated in Figure 4.

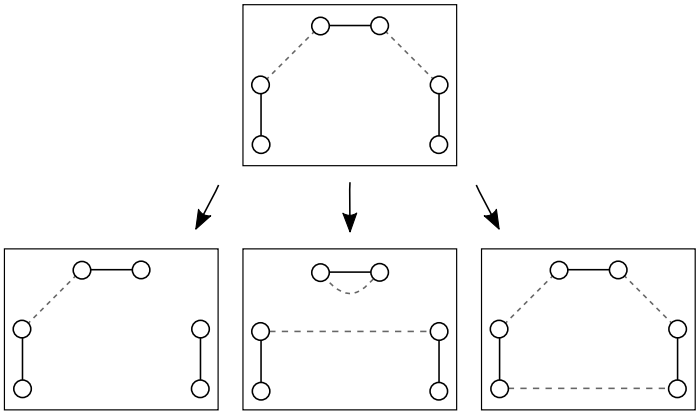


Fig. 4. Examples of the three options of optimal moves from a solid path. The first one, from the top, is a cut, splitting the 5-path into two paths with 3 and 1 edges, respectively. The second is a double swap extracting a cycle from the path. The last option, only possible in this kind of path, is to join the ends, forming a cycle.

Given a path with length k, the sizes of the optimal scenarios fall in a limited range. The longest operation scenarios for a k-path are the ones with only cuts

and joins, making up a total of k operations. Since the double swaps have twice the weight of a cut or join, scenarios with more double swaps are shorter.

With this set of optimal operations, and the range for the length of a scenario, we arrive at three recurrences, one for each type of path. Since the cuts and double swaps are only applied to dashed edges, the indices are different according to the type of the path, and the paths obtained after splitting also have different types. The details are left for an appendix, for lack of space. The following theorem summarizes our results.

Theorem 4. *If the graph $BG(A, B)$ has q big paths with lengths $k_1, \ldots, k_q$, and p big cycles with lengths $2\ell_1 + 2, \ldots, 2\ell_p + 2$, the total number of optimal sorting scenarios from A to B is given by the product:*

$$\mathit{Cycles}(\ell_1, \ldots, \ell_p)\mathit{ShufflePaths}(k_1, \ldots, k_q; \ell_1, \ldots, \ell_p),$$

where $\mathit{ShufflePaths}(k_1, \ldots, k_q; \ell_1, \ldots, \ell_p)$ is given by

$$\sum_{\ell'_1=\lfloor k_1/2 \rfloor+1}^{k_1} \cdots \sum_{\ell'_q=\lfloor k_q/2 \rfloor+1}^{k_q} \frac{(\ell_1 + \ldots + \ell_p + \ell'_1 + \ldots + \ell'_q)!}{\ell_1! \ldots \ell_p! \ell'_1! \ldots \ell'_q!} \prod_{j=1}^{q} S_p(k_j, \ell'_j)$$

and

$$\mathit{Cycles}(\ell_1, \ldots, \ell_p) = \prod_{i=1}^{p} S_c(2\ell_i + 2).$$

4 Intermediate Genomes

We say a genome B is an intermediate genome between genomes A and C when

$$d(A, C) = d(A, B) + d(B, C).$$

We have already shown that there is never recombination of different components in the rank distance. Therefore, we can get all possible intermediates by looking separately at each component of $BG(A, C)$. If the graph $BG(A, C)$ has p big cycles with lengths $k_1, k_2, \ldots, k_p$, and q big paths with lengths $k'_1, k'_2, \ldots, k'_q$, the total number $I(A, C)$ of intermediate genomes between A and C is

$$I(A, C) = \prod_{i=1}^{p} I_c(k_i) \prod_{j=1}^{q} I_p(k'_j),$$

where $I_c(k)$ is the number of intermediates for a k-cycle, and $I_p(k)$ is the number of intermediates for a k-path.

For cycles of length $2k$, with $k \geq 1$, rank optimal operations are the same as DCJ optimal operations, so we can use the known result for the DCJ distance [7]:

$$I_c(2k) = \frac{1}{k+1}\binom{2k}{k}.$$

For paths, we derive in the appendix a similar formula for $I_p(k)$, the number of intermediate genomes for a path of length $k \geq 0$:

$$I_p(k) = \binom{k+1}{\lfloor (k+1)/2 \rfloor}.$$

The following theorem summarizes the intermediate counts.

Theorem 5. *If the graph $BG(A, B)$ has p big cycles with lengths $2\ell_1, \ldots, 2\ell_p$, and q big paths with lengths $k_1, \ldots, k_q$, the total number of intermediate genomes between A and B is given by:*

$$\prod_{i=1}^{p} I_c(2\ell_i) \prod_{j=1}^{q} I_p(k_j) = \prod_{i=1}^{p} \frac{1}{\ell_i + 1} \binom{2\ell_i}{\ell_i} \prod_{j=1}^{q} \binom{k_j+1}{\lfloor (k_j+1)/2 \rfloor}.$$

4.1 Relationship with Sperner Families

The number $I_p(k)$ is equivalent to the number of $\lfloor (k+1)/2 \rfloor$-element subsets of a set with $(k+1)$ elements. According to Sperner's Theorem [21, 15], this is the maximum number of subsets of a set with $k+1$ elements where no set contains another. Such a family of sets is called a *Sperner family*. Building on this idea, we provide a bijection between the intermediates of a k-path, and the Sperner family of all $\lfloor (k+1)/2 \rfloor$-element subsets of a $k+1$-set in the Appendix.

5 Experiments

We implemented our formulas and tested our method for counting the scenarios and the intermediates between pairs of a number of genomes from the literature.

5.1 Data Sets

We used four data sets from different sources. The first and simplest data set is from the work of Palmer and Hebron on plants of the Brassica genus [18]. It consists of two pairs of circular mitochondrial DNA, comparing *Brassica campestris* against *B. oleracea* and *B. napus*. Both instances have 5 synteny blocks. Other comparisons in this work have insertions or deletions and were not considered.

Another data set is the human and mouse X chromosome, from Pevzner and Tesler [19]. This pair of linear, single-chromosome, inputs has 11 synteny blocks.

The third data set is composed of 13 chloroplast genomes, of which 12 are from the Campanulaceae family, and 1 from tobacco as an outgroup, with 105 synteny blocks, taken from Cosner, Raubeson and Jansen [6], and also used by Bourque and Pevzner [2]. These created 78 pairs of inputs to the sorting problem.

The fourth and largest data set contains genomes used by Kim et al. to test their reconstruction algorithm DESCHRAMBLER [13]. It consists of 20 instances comparing the human genome against the genome from other animals, namely, 18 Eutherian mammals, plus opossum and chicken as outgroups. These pairs have between 101 and 621 synteny blocks.

5.2 Code

The code to run our experiments was implemented in Python, and executed on a virtual machine using a single 2.3GHz processor core and 2GB of memory. Building the breakpoint graph of our tests instances, with up to 1242 extremities, is not a computationally intensive task, and neither is computing the number of intermediates, a simple product of binomial coefficients. The most demanding task is to compute the number of scenarios, especially when the breakpoint graph has a large number of paths. This is a consequence of the fact that each path adds an extra summation in the expression for the number of scenarios.

5.3 Results and Discussion

The two Brassica instances have the same rank distance of 6 (in this case three double swaps), and the same results: 9 scenarios and 10 intermediate genomes.

For the pair of X chromosomes, with a rank distance of 14, we get 237440 scenarios and 560 intermediates.

With the chloroplast genome pairs, we get varying results. Some pairs, like *Trachelium* and *Campanula* are only one double swap apart, and therefore have only one optimal scenario and two intermediates (the input genomes). The pair of genomes that are farthest apart is *Merciera* and *Platycodon*, with a distance of 48. They have 1.4×10^{32} optimal scenarios, and 4.9×10^{12} intermediate genomes.

With the Eutherian data set, due to the large number of terms in the summation in Theorem 4 for the number of scenarios between distantly related species, we developed a scheme that is always guaranteed to use a specified amount of memory, but sometimes ends up producing upper and lower bounds on the number of scenarios instead of the exact values. This scheme produced the exact result for 5 out of the 20 instances, namely, the primates (chimpanzee, marmoset, orangutan, and rhesus) and the horse. For the other instances, we obtained upper and lower bounds on the number of optimal scenarios, using the monotonicity of $S_p(k, l)$ and multinomial coefficients containing l with respect to the l variable, and utilizing up to $N = 10^9$ memory cells, as described in the Appendix. On the other hand, computing the number of intermediates proved easy even for the farthest pairs of genomes. In Table 1 we list the number of scenarios (or an interval containing it) and the number of intermediates for all the instances.

Comparing the larger instances of the chloroplast data set with the ones from the human, cat and mouse genomes, we note that linear, multichromosomal genomes tend to have more scenarios than circular, unichromosomal ones. This may be due to the fact that each path in the breakpoint graph adds another summation in the scenario formula, consisting of many products. In contrast, each cycle only adds one product to existing terms. The number of intermediates, on the other hand, is likely to be less variable between instances with the same distance, since the formulas for intermediates in paths and cycles are very similar.

Table 1. Rank distance, number of scenarios, and number of intermediates between the human genome and the genomes of 20 Eutherian animals, listed in order of distance.

Genome	d	Scenarios	Intermediates
Chimpanzee	27	6.54×10^{11}	2.46×10^{4}
Orangutan	53	6.03×10^{38}	1.29×10^{10}
Rhesus	150	1.21×10^{138}	1.45×10^{28}
Marmoset	204	3.99×10^{250}	3.13×10^{43}
Horse	225	1.63×10^{135}	1.31×10^{51}
Dog	304	$[10^{432}, 10^{471}]$	6.37×10^{70}
Pig	318	$[10^{463}, 10^{479}]$	1.61×10^{73}
White rhino	328	$[10^{546}, 10^{588}]$	1.36×10^{84}
Elephant	336	$[10^{583}, 10^{609}]$	8.56×10^{86}
Cattle	383	$[10^{537}, 10^{579}]$	7.39×10^{83}
Pika	385	$[10^{647}, 10^{710}]$	2.89×10^{98}
Goat	393	$[10^{548}, 10^{588}]$	5.30×10^{85}
Tenrec	407	$[10^{700}, 10^{778}]$	1.71×10^{105}
Shrew	487	$[10^{876}, 10^{999}]$	1.02×10^{128}
Mouse	509	$[10^{830}, 10^{980}]$	2.44×10^{131}
Manatee	519	$[10^{1050}, 10^{1101}]$	3.12×10^{142}
Guinea pig	640	$[10^{1130}, 10^{1360}]$	5.85×10^{166}
Chicken	736	$[10^{1217}, 10^{1550}]$	6.38×10^{193}
Opossum	778	$[10^{1295}, 10^{1590}]$	4.01×10^{204}
Rat	788	$[10^{1251}, 10^{1476}]$	2.30×10^{189}

6 Conclusion

In this paper we opened the doors for the exploration of the solution space of the rank distance problem. We demonstrated that there is no recombination between components of the breakpoint graph in any optimal rank sorting scenario. We then gave a formula for the number of optimal sorting scenarios for co-tailed genomes, and presented a general algorithm for counting the number of scenarios.

We also presented a formula for the number of intermediates between two genomes. In addition, we constructed a bijection that provides a simple way to uniformly sample intermediates. Sampling intermediate genomes is the next step in the study of the solution space and can be very helpful in future applications.

References

1. Bergeron, A., Mixtacki, J., Stoye, J.: A unifying view of genome rearrangements. Algorithms in Bioinformatics pp. 163–173 (2006)
2. Bourque, G., Pevzner, P.A.: Genome-scale evolution: reconstructing gene orders in the ancestral species. Genome Research **12**(1), 26–36 (2002)
3. Braga, M.D.V., Willing, E., Stoye, J.: Genomic distance with DCJ and indels (2010)
4. Braga, M.D., Stoye, J.: The solution space of sorting by DCJ. Journal of Computational Biology **17**(9), 1145–1165 (2010)

5. Compeau, P.E.C.: DCJ-Indel sorting revisited. Algorithms for Molecular Biology **8**(1), 6 (2013)
6. Cosner, M.E., Raubeson, L.A., Jansen, R.K.: Chloroplast DNA rearrangements in Campanulaceae: phylogenetic utility of highly rearranged genomes. BMC Evolutionary Biology **4**(1), 1–17 (2004)
7. Feijão, P.: Reconstruction of ancestral gene orders using intermediate genomes. BMC Bioinformatics **16**(14), S3 (Oct 2015)
8. Feijão, P., Mane, A., Chauve, C.: A tractable variant of the single cut or join distance with duplicated genes. In: Meidanis, J., Nakhleh, L. (eds.) Comparative Genomics. pp. 14–30. Springer International Publishing, Cham (2017)
9. Feijao, P., Meidanis, J.: SCJ: a breakpoint-like distance that simplifies several rearrangement problems. IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB) **8**(5), 1318–1329 (2011)
10. Feijão, P., Meidanis, J.: Extending the algebraic formalism for genome rearrangements to include linear chromosomes. IEEE/ACM Transactions on Computational Biology and Bioinformatics **10**(4), 819–831 (2013)
11. Hannenhalli, S., Pevzner, P.A.: Transforming men into mice (polynomial algorithm for genomic distance problem). In: Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on. pp. 581–592. IEEE (1995)
12. Jamshidpey, A., Jamshidpey, A., Sankoff, D.: Sets of medians in the non-geodesic pseudometric space of unsigned genomes with breakpoints. BMC Genomics **15**(6), S3 (2014)
13. Kim, J., Farré, M., Auvil, L., Capitanu, B., Larkin, D.M., Ma, J., Lewin, H.A.: Reconstruction and evolutionary history of eutherian chromosomes. Proceedings of the National Academy of Sciences **114**(27), E5379–E5388 (2017)
14. Larget, B., Kadane, J.B., Simon, D.L.: A Bayesian approach to the estimation of ancestral genome arrangements. Molecular Phylogenetics and Evolution **36**(2), 214–223 (2005)
15. Lubell, D.: A short proof of Sperner's lemma. Journal of Combinatorial Theory **1**(2), 299 (1966)
16. Miklós, I., Kiss, S.Z., Tannier, E.: Counting and sampling SCJ small parsimony solutions. Theoretical Computer Science **552**, 83–98 (2014)
17. Ouangraoua, A., Bergeron, A.: Combinatorial structure of genome rearrangements scenarios. Journal of Computational Biology **17**(9), 1129–1144 (2010)
18. Palmer, J.D., Herbon, L.A.: Plant mitochondrial DNA evolved rapidly in structure, but slowly in sequence. Journal of Molecular Evolution **28**(1), 87–97 (1988)
19. Pevzner, P., Tesler, G.: Genome rearrangements in mammalian evolution: lessons from human and mouse genomes. Genome Research **13**(1), 37–45 (2003)
20. Shao, M., Lin, Y., Moret, B.: Sorting genomes with rearrangements and segmental duplications through trajectory graphs. BMC Bioinformatics **14**(15), S9 (Oct 2013)
21. Sperner, E.: Ein satz über untermengen einer endlichen menge. Mathematische Zeitschrift **27**(1), 544–548 (Dec 1928)
22. Tannier, E., Zheng, C., Sankoff, D.: Multichromosomal median and halving problems under different genomic distances. BMC Bioinformatics **10**(1), 120 (2009)
23. Yancopoulos, S., Attie, O., Friedberg, R.: Efficient sorting of genomic permutations by translocation, inversion and block interchange. Bioinformatics **21**(16), 3340–3346 (2005)
24. Zanetti, J.P.P., Biller, P., Meidanis, J.: Median approximations for genomes modeled as matrices. Bulletin of Mathematical Biology **78**(4), 786–814 (Apr 2016)