A 3.5-Approximation Algorithm for Sorting by Intergenic Transpositions

Andre Rodrigues Oliveira[1(✉)], Géraldine Jean[2], Guillaume Fertin[2], Klairton Lima Brito[1], Ulisses Dias[3], and Zanoni Dias[1]

[1] Institute of Computing, University of Campinas, Campinas, Brazil
{andrero,klairton,zanoni}@ic.unicamp.br
[2] LS2N, UMR CNRS 6004, University of Nantes, Nantes, France
{geraldine.jean,guillaume.fertin}@univ-nantes.fr
[3] School of Technology, University of Campinas, Limeira, Brazil
ulisses@ft.unicamp.br

Abstract. Genome Rearrangements affect large stretches of genomes during evolution. One of the most studied genome rearrangement is the *transposition*, which occurs when a sequence of genes is moved to another position inside the genome. Mathematical models have been used to estimate the evolutionary distance between two different genomes based on genome rearrangements. However, many of these models have focused only on the (order of the) genes of a genome, disregarding other important elements in it. Recently, researchers have shown that considering existing regions between each pair of genes, called *intergenic regions*, can enhance the distance estimation in realistic data. In this work, we study the transposition distance between two genomes, but we also consider intergenic regions, a problem we name Sorting Permutations by Intergenic Transpositions (SbIT). We show that this problem is NP-hard and propose a 3.5-approximation algorithm for it.

Keywords: Genome rearrangements · Intergenic regions · Transpositions · Approximation algorithm

1 Introduction

Genome rearrangements are events that modify genomes by inserting or removing large stretches of DNA sequences, or by changing the order and the orientation of genes inside genomes. A transposition [1] is a rearrangement that swaps the position of two adjacent sequences of genes inside a genome. Another example of genome rearrangement is the reversal [11], that reverses the order and the orientation of a sequence of genes.

We compute the **rearrangement distance** between two genomes by determining the minimum number of events that transform one into another. A **model** $\mathcal{M}$ is a set of genome rearrangements that can be used to calculate the rearrangement distance.

C. Martín-Vide et al. (Eds.): AlCoB 2020, LNBI 12099, pp. 16–28, 2020.
https://doi.org/10.1007/978-3-030-42266-0_2

Algorithms based on the rearrangement distance perform whole-genome comparison and may be used as a tool to infer phylogenetic relationships. The usual method fills a matrix of pairwise distances among genomes that is later used to generate phylogenetic trees [2,12,14]. As for "classical" rearrangements, having a large spectrum of models globally helps better-understanding things.

While in practice it is likely rarely so, if genomes contain no repeated gene and share the same set of n genes, they can be represented as permutations. Without loss of generality, we consider that one of these genomes is the **identity permutation**, i.e., the sorted permutation $\iota = (1\ 2\ \ldots n)$.

The Sorting by Rearrangements Problem thus consists in determining the shortest sequence of events from $\mathcal{M}$ that sorts a permutation π, i.e., that transforms it to ι. Sorting by Rearrangements has been extensively studied in the past. For instance, **Sorting by Transpositions** has been proved NP-hard [6], while the best algorithm so far has an approximation factor of 1.375 [8].

Representing genomes through their gene order (thus, by permutations) implies that information not contained directly in the genes is lost. In particular, in the case of intergenic regions, DNA sequences between the genes are not considered. Recently, some authors argued that incorporating intergenic regions sizes in the models changes the distance estimations, and actually improves them [3,4]. It seems worth investigating models considering both gene order and intergenic sizes.

Results considering intergenic sizes for models with Double-Cut and Join (DCJ) and DCJs along with indels (i.e., insertions and deletions) are known: the former is NP-hard and it has a 4/3-approximation algorithm [9], while the latter is polynomial [7]. In addition to the approximation algorithm, the authors in [7] also developed two exact algorithms: a fixed-parameter tractable algorithm and an integer linear programming formulation. Practical tests from [7] showed that statistical properties of the inferred scenarios using intergenic regions are closer to the true ones than scenarios which do not use them.

Some results considering intergenic sizes with super short operations (i.e., a reversal or a transposition applied to one or two genes of the genome) are known [13]. Using the concept of breakpoints, Brito et al. showed a 4-approximation algorithm (resp. 6-approximation algorithm) for sorting by reversals (resp. reversals and transpositions) when also considering intergenic regions on unsigned permutations [5]. They also showed that both problems are NP-hard.

In this paper, we investigate the transposition distance between genomes that also takes into account intergenic regions, a problem we name **Sorting by Intergenic Transpositions (SbIT)**. Instead of using breakpoints, here we propose a modification of a known graph structure to represent both gene order and intergenic sizes in a single graph. We show that SbIT is NP-hard, and, with the help of this adapted graph structure, we design a 3.5-approximation algorithm.

This work is organized as follows. Section 2 presents some definitions we extensively use throughout this paper. Section 3 presents the graph structure we use to produce our approximation algorithm. Section 4 contains a series of

intermediate lemmas that support our algorithm. Section 5 describes the 3.5-approximation algorithm for SbIT. Section 6 concludes the paper.

2 Basic Definitions

A genome $\mathcal{G}$ is a sequence of n genes denoted by g_i, with $i \in [1..n]$, in which two consecutive genes g_{j-1} and g_j, with $j \in [2..n]$, are separated by a noncoding region called intergenic region, denoted by r_j – that are also present on its extremities (r_1 and r_{n+1}): $\mathcal{G} = r_1, g_1, r_2, g_2, \ldots, r_n, g_n, r_{n+1}$.

Given two genomes of closely related species, for the simplifying purposes of our initial analysis, we expect they will share the same set of genes, which may appear in different orders due to genome rearrangements. Selective pressures tend to conserve genes and not intergenic regions [3]. Therefore, genome rearrangements hardly cut inside genes, whereas cuts appear in intergenic regions.

Our model assumes that (i) no gene is duplicated in a genome and (ii) both genomes share the same set of genes. We assign unique integer numbers in the range $[1..n]$ to each gene and represent them as a permutation. Therefore, the sequence of genes in a genome is modeled by a permutation $\pi = (\pi_1\ \pi_2\ \ldots\ \pi_n)$, $\pi_i \in \mathbb{N}$, $1 \leq \pi_i \leq n$, and $\pi_i \neq \pi_j$ for all $i \neq j$.

We represent intergenic regions by their lengths instead of assigning unique identifiers to each of them, which would be pointless because rearrangements may split intergenic regions several times. The sequence of intergenic regions around n genes is represented as $\breve{\boldsymbol{\pi}} = (\breve{\boldsymbol{\pi}}_1\ \breve{\boldsymbol{\pi}}_2\ \ldots\ \breve{\boldsymbol{\pi}}_{n+1})$, $\breve{\pi}_i \in \mathbb{N}$. Intergenic region $\breve{\pi}_i$ is on the left side of π_i, whereas $\breve{\pi}_{i+1}$ is on the right side.

Our goal is to compute the distance between two genomes: $(\pi, \breve{\pi})$ and $(\sigma, \breve{\sigma})$. We may assign unique labels to genes arbitrarily, so we simplify the definition of our problem by setting σ as the identity permutation ι, such that $\sigma = \iota = (1\ 2\ \ldots\ n)$ and $\breve{\sigma} = \breve{\iota}$, which describes all the information we need to our problem. Therefore, an **instance** of our problem is composed by three elements $(\pi, \breve{\pi}, \breve{\iota})$, such that $\sum_{i=1}^{n+1} \breve{\pi}_i = \sum_{i=1}^{n+1} \breve{\iota}_i$, which guarantees that total intergenic region lengths are conserved.

An **intergenic transposition** is an operation $\rho_{(x,y,z)}^{(i,j,k)}$, $1 \leq i < j < k \leq n+1$, $0 \leq x \leq \breve{\pi}_i$, $0 \leq y \leq \breve{\pi}_j$, $0 \leq z \leq \breve{\pi}_k$, and $\{x, y, z\} \subset \mathbb{N}$. An intergenic transposition acts on instances to generate new ones: $(\pi, \breve{\pi}, \breve{\iota}) \cdot \rho_{(x,y,z)}^{(i,j,k)} = (\pi', \breve{\pi}', \breve{\iota})$, where (i) $\pi' = (\pi_1 \pi_2 \ldots \pi_{i-1}\ \underline{\pi_j \pi_{j+1} \ldots \pi_{k-1}}\ \underline{\pi_i \pi_{i+1} \ldots \pi_{j-1}}\ \pi_k \pi_{k+1} \ldots \pi_n)$, and (ii) $\breve{\pi}' = (\breve{\pi}_1 \ldots \breve{\pi}_{i-1}\ \boxed{\breve{\pi}'_i}\ \underline{\breve{\pi}_{j+1} \ldots \breve{\pi}_{k-1}}\ \boxed{\breve{\pi}'_j}\ \underline{\breve{\pi}_{i+1} \ldots \breve{\pi}_{j-1}}\ \boxed{\breve{\pi}'_k}\ \breve{\pi}_{k+1} \ldots \breve{\pi}_{n+1})$, such that $\breve{\pi}'_i = x + \breve{\pi}_j - y$, $\breve{\pi}'_j = z + \breve{\pi}_i - x$, and $\breve{\pi}'_k = y + \breve{\pi}_k - z$.

As we can see, while $\rho_{(x,y,z)}^{(i,j,k)}$ keeps $\breve{\iota}$ intact, it moves segments from π and $\breve{\pi}$ to other positions and also modifies the contents of three elements from $\breve{\pi}$: it cuts $\breve{\pi}_i$ after first x nucleotides, $\breve{\pi}_j$ after first y nucleotides, and $\breve{\pi}_k$ after first z nucleotides, and rearranges them as defined above. Figure 1 shows examples of instances and the application of an intergenic transposition.

The **intergenic transposition distance** $d_t(\pi, \breve{\pi}, \breve{\iota})$ is the minimum number m of intergenic transpositions $\rho_1, \ldots, \rho_m$ that transform π into ι, and $\breve{\pi}$ into $\breve{\iota}$.

Therefore, $d_t(\pi, \breve{\pi}, \breve{\iota}) = m$ implies a minimal sequence $(\pi, \breve{\pi}, \breve{\iota}) \cdot \rho_1 \cdot \ldots \cdot \rho_m = (\iota, \breve{\iota}, \breve{\iota})$.

Lemma 1. *Sorting by Intergenic Transpositions (SbIT) is NP-hard.*

Proof. The Sorting by Transpositions problem (SbT) has already been proved NP-hard [6]. An instance of this problem consists of a permutation γ and a non-negative integer d. The goal is to determine if its possible to transform γ into ι applying at most d transpositions.

We can reduce all instances of SbT to instances of SbIT by setting $\pi = \gamma$ and $\breve{\pi} = \breve{\iota} = (0\ 0\ ...\ 0)$. Note that it is possible to transform γ into ι applying at most d transpositions if and only if $d_t(\pi, \breve{\pi}, \breve{\iota}) \leq d$. □

From now on, we will refer to intergenic transposition as transposition only.

3 Weighted Cycle Graph

We adapted a graph structure called *breakpoint graph* [1,10] to conveniently represent an instance $(\pi, \breve{\pi}, \breve{\iota})$ in a single graph. This structure allows us to describe algorithms and prove approximation bounds. All definitions we propose here are exemplified in Figs. 2 and 3.

We represent a given instance by a **weighted cycle graph** $G(\pi, \breve{\pi}, \breve{\iota}) = (V, E, w)$, where V is the set $\{-n, \ldots, -2, -1, 1, 2, \ldots, n\} \cup \{0, -(n+1)\}$, E is the set of edges that can be either gray or black, and $w : E \to \mathbb{N}$ is a function mapping edges to values corresponding to intergenic region lengths. The black edge set is $\{e_i = (-\pi_i, +\pi_{i-1}) : 1 \leq i \leq n+1\}$, and $w(e_i) = \breve{\pi}_i$. The gray edge set is $\{e'_i = (+(i-1), -i) : 1 \leq i \leq n+1\}$, and $w(e'_i) = \breve{\iota}_i$. In this definition, we consider $\pi_0 = 0$ and $\pi_{n+1} = n+1$.

The graph can be drawn in many arbitrary ways, but it is more convenient to place its vertices on a horizontal line in the same order as the elements of π, so π_0 (resp. $-\pi_{n+1}$) is the leftmost (resp. rightmost) element of it. In addition,

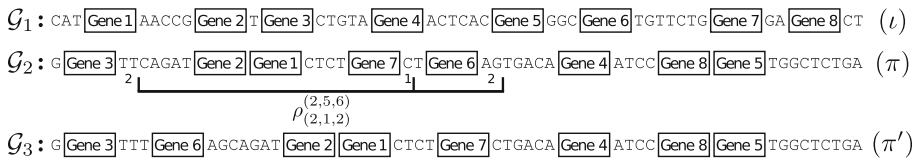

Fig. 1. Two (fictitious) genomes $\mathcal{G}_1$ and $\mathcal{G}_2$ that share 8 genes. We represent $\mathcal{G}_1$ as the identity permutation, which leads to $\mathcal{G}_2$ as the permutation $\pi = (3\ 2\ 1\ 7\ 6\ 4\ 8\ 5)$. We assume that the number of nucleotides between genes are good estimators for intergenic regions lengths. For example, in $\mathcal{G}_1$ before "Gene 1" we have 3 nucleotides, between "Gene 1" and "Gene 2" we have 5, and so on. Thus, $(\pi, \breve{\pi}, \breve{\iota})$ is such that $\pi = (3\ 2\ 1\ 7\ 6\ 4\ 8\ 5)$, $\breve{\pi} = (1\ 7\ 0\ 4\ 2\ 7\ 4\ 0\ 9)$, and $\breve{\iota} = (3\ 5\ 1\ 5\ 6\ 3\ 7\ 2\ 2)$. Genome $\mathcal{G}_3$ represents $(\pi', \breve{\pi}', \breve{\iota}') = (\pi, \breve{\pi}, \breve{\iota}) \cdot \rho^{(2,5,6)}_{(2,1,2)}$, so $\pi' = (3\ 6\ 2\ 1\ 7\ 4\ 8\ 5)$, $\breve{\pi}' = (1\ 3\ 7\ 0\ 4\ 6\ 4\ 0\ 9)$.

for each element $\pi_i \in \pi$, vertex $-\pi_i \in G(\pi, \breve{\pi}, \breve{\iota})$ is drawn to the left of vertex $+\pi_i$. Since black edges relate to π, they are drawn as horizontal lines, and we label the black edge e_i as i. Gray edges are drawn as arcs.

Each vertex in $G(\pi, \breve{\pi}, \breve{\iota})$ has a gray edge and a black edge, which allows a unique decomposition of edges in cycles of alternating colors. Each cycle C with ℓ black edges is represented as a list $(c^1, c^2, \ldots, c^\ell)$ of the labels from its black edges, and to make the notation unique we assume c^1 to be the index of the "rightmost" black edge (i.e., the black edge with the highest label using our default drawing) and we traverse it from right to left. We follow by several definitions regarding cycles.

A cycle is **long** if it has 3 or more black edges; a cycle is **short** if it has 2 black edges; a cycle is **trivial** if it has 1 black edge; a cycle is **non-trivial** if it is either short or long. A non-trivial cycle $C = (c^1, \ldots, c^\ell)$ is **non-oriented** if $c^1, \ldots, c^\ell$ is a decreasing sequence (note that every short cycle C is non-oriented); C is **oriented** otherwise.

Given a non-trivial cycle $C = (c^1, \ldots, c^\ell)$, every pair of black edges e_{c^i} and $e_{c^{(i+1)}}$ with $1 \leq i < \ell$ is called an **open gate** if for any $c^j \in C$ with $j \notin \{i, i{+}1\}$ either $c^j > c^i$ or $c^j < c^{i+1}$, if $c^i > c^{i+1}$, and either $c^j > c^{i+1}$ or $c^j < c^i$ otherwise. Besides, the pair of black edges e_{c^1} and e_{c^ℓ} is called an **open gate** if $c^1 > c^j > c^\ell$ for any $c^j \in C$ with $j \notin \{1, \ell\}$. Note that on every short cycle $C = (c^1, c^2)$ the pair e_{c^1}, e_{c^2} is an open gate.

Bafna and Pevzner [1] showed that for every open gate e_{c^i} and e_{c^j} from C with $c^i > c^j$ there exists another non-trivial cycle D with black edges e_{d^i} and e_{d^j} such that either $c^i > d^i > c^j > d^j$ or $d^i > c^i > d^j > c^j$. In this case, we say that D **closes** this open gate. Figure 2 shows two examples of cycles with open gates and how cycles can close these open gates.

Cycles can be either **balanced** or **unbalanced**. A cycle $C = (c^1, \ldots, c^\ell)$ is **balanced** if $\sum_{i=1}^{\ell} |w(e'_{c^i}) - w(e_{c^i})| = 0$, and is **unbalanced** otherwise. In other words, one cycle is balanced if the sum of weights of gray edges equals the sum of weights of black edges, and it is unbalanced otherwise. An unbalanced cycle $C = (c^1, \ldots, c^\ell)$ is called **positive** if $\sum_{i=1}^{\ell} (w(e'_{c^i}) - w(e_{c^i})) > 0$, and it is called **negative** otherwise. Figure 3 shows an example of a weighted cycle graph and

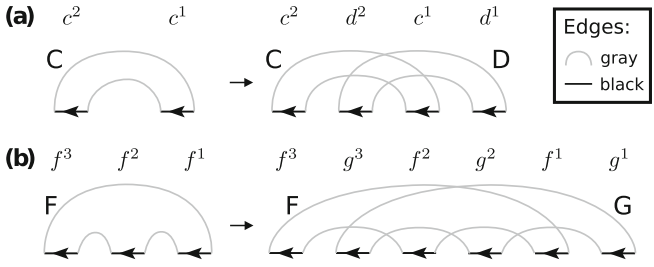

Fig. 2. **(a)** The cycle $C = (c^1, c^2)$ has an open gate (c^1, c^2) closed by $D = (d^1, d^2)$ on the right. **(b)** The cycle $F = (f^1, f^2, f^3)$ has three open gates (f^1, f^2), (f^2, f^3), and (f^3, f^1) closed by $G = (g^1, g^2, g^3)$ on the right.

the application of an intergenic transposition on it, as well as the weighted cycle graph for an instance $(\iota, \check{\iota}, \check{\iota})$.

Let $\boldsymbol{c(\pi, \check{\pi}, \check{\iota})}$, $\boldsymbol{c_b(\pi, \check{\pi}, \check{\iota})}$, and $\boldsymbol{c_u(\pi, \check{\pi}, \check{\iota})}$, denote the number of cycles, balanced cycles, and unbalanced cycles in $G(\pi, \check{\pi}, \check{\iota})$, respectively.

Lemma 2. *The instance $(\iota, \check{\iota}, \check{\iota})$ has two properties that do not occur together in any other instance: (i) $c(\iota, \check{\iota}, \check{\iota}) = n + 1$, and (ii) $c_b(\iota, \check{\iota}, \check{\iota}) = n + 1$. As a consequence, we have that $c_u(\iota, \check{\iota}, \check{\iota}) = 0$ (see Fig. 3(c) for an example).*

Note that Sorting by Intergenic Transpositions is more complicated than the Sorting by Transpositions because increasing the number of cycles is not sufficient - these cycles need to be balanced.

Given a sequence of transpositions $\mathcal{S}_\rho = (\rho_1, \rho_2, \ldots, \rho_k)$, let $\boldsymbol{(\pi, \check{\pi}, \check{\iota}) \cdot \mathcal{S}_\rho}$ denotes $(\pi, \check{\pi}, \check{\iota}) \cdot \rho_1 \cdot \rho_2 \cdot \ldots \cdot \rho_k$ such that ρ_{i+1} is always a transposition for $(\pi, \check{\pi}, \check{\iota}) \cdot \rho_1 \cdot \ldots \cdot \rho_i$ with $1 \leq i < k$. Let $\boldsymbol{\Delta c(\pi, \check{\pi}, \check{\iota}, \mathcal{S}_\rho)} = c((\pi, \check{\pi}, \check{\iota}) \cdot \mathcal{S}_\rho) - c(\pi, \check{\pi}, \check{\iota})$ and $\boldsymbol{\Delta c_b(\pi, \check{\pi}, \check{\iota}, \mathcal{S}_\rho)} = c_b((\pi, \check{\pi}, \check{\iota}) \cdot \mathcal{S}_\rho) - c_b(\pi, \check{\pi}, \check{\iota})$ denote the variation in the number of cycles and balanced cycles, respectively, when $\mathcal{S}_\rho$ is applied to $(\pi, \check{\pi}, \check{\iota})$.

Lemma 3. *$\Delta c(\pi, \check{\pi}, \check{\iota}, \rho) \in \{2, 0, -2\}$ for any transposition ρ.*

Proof. Straightforward from Bafna and Pevzner [1]. □

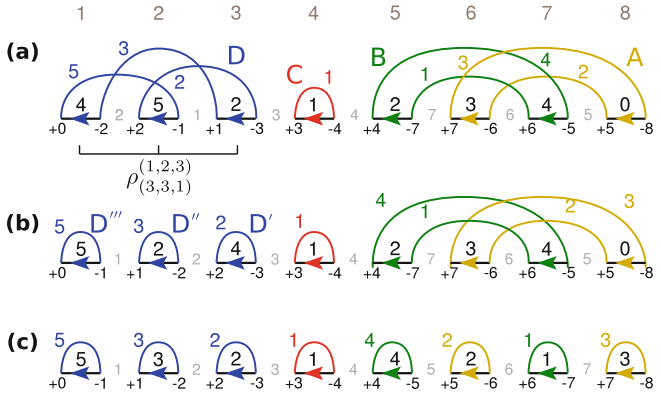

Fig. 3. We color the figure to improve cycle visualization. Black edges are drawn as horizontal lines, and gray edges are drawn as arcs. Arrows over black edges represent how we traverse them, and numbers on the top of the image indicate black edges labels. **(a)** The weighted cycle graph $G(\pi, \check{\pi}, \check{\iota})$ for $\pi = (2\ 1\ 3\ 4\ 7\ 6\ 5)$, $\check{\pi} = (4\ 5\ 2\ 1\ 2\ 3\ 4\ 0)$, and $\check{\iota} = (5\ 3\ 2\ 1\ 4\ 2\ 1\ 3)$. $G(\pi, \check{\pi}, \check{\iota})$ has four cycles: $A = (8, 6)$ is a non-oriented positive short cycle; $B = (7, 5)$ is a non-oriented short negative cycle; $C = (4)$ is a trivial balanced cycle, and $D = (3, 1, 2)$ is an oriented long negative cycle. **(b)** The weighted cycle graph $G((\pi, \check{\pi}, \check{\iota}) \cdot \rho^{(1,2,3)}_{(3,3,1)})$, with $\Delta c(\pi, \check{\pi}, \check{\iota}, \rho) = 2$ and $\Delta c_b(\pi, \check{\pi}, \check{\iota}, \rho) = 1$. The transposition applied to the negative cycle D from **(a)** transformed it into three trivial cycles $D' = (3)$, $D'' = (2)$, and $D''' = (1)$ such that D''' is balanced. **(c)** The weighted cycle graph $G(\iota, \check{\iota}, \check{\iota})$, with $c(\iota, \check{\iota}, \check{\iota}) = c_b(\iota, \check{\iota}, \check{\iota}) = n + 1$, and $c_u(\iota, \check{\iota}, \check{\iota}) = 0$.

Lemma 4. *$\Delta c_b(\pi, \breve{\pi}, \breve{\iota}, \rho) \leq 2$ for any transposition ρ.*

Proof. From Lemma 3 we know that we can increase the number of cycles by at most 2. In this scenario, one cycle C is split in three by a single transposition ρ. If C is balanced, the best we can expect is that ρ creates three balanced cycles, so $\Delta c_b(\pi, \breve{\pi}, \breve{\iota}, \rho) = 2$. Otherwise, at least one of the resulting cycles shall be unbalanced too, since weights of black edges of the three cycles sum up to a value that is different from the sum of weights of gray edges. Therefore, the best we expect is that the other two cycles are balanced, so $\Delta c_b(\pi, \breve{\pi}, \breve{\iota}, \rho) = 2$. □

Theorem 5. $d_t(\pi, \breve{\pi}, \breve{\iota}) \geq \frac{n+1-c_b(\pi,\breve{\pi},\breve{\iota})}{2}$.

Proof. By Lemma 2 we know that $c_b(\iota, \breve{\iota}, \breve{\iota}) = n + 1$. Therefore, our goal is to increase the number of cycles from $c_b(\pi, \breve{\pi}, \breve{\iota})$ to $n+1$; since this number increases by at most 2 for each transposition (Lemma 4), the lemma follows. □

4 Preliminary Results

This section presents properties and lemmas to support the 3.5-approximation presented in Sect. 5.

Let e_{c^x} and e_{c^y} be two arbitrary black edges in the same cycle $C = (c^1, \ldots, c^\ell)$, with $\{x, y\} \subset [1..\ell]$. We define the function $f : E \times E \to \mathbb{Z}$ as $f(e_{c^x}, e_{c^y}) = \sum_{i=x \pmod \ell)+1}^{y-1} [w(e'_{c^i}) - w(e_{c^i})] + w(e'_{c^x})$.

In other words, given the path P of black and gray edges that goes from e_{c^x} to e_{c^y}, $f(e_{c^x}, e_{c^y})$ computes the sum of weights of gray edges in P minus the weights of black edges from P – excluding the black edges e_{c^x} and e_{c^y}. Observe that we traverse the first black edge in the cycle from right to left, which means that the path that goes from e_{c^x} to e_{c^y} is different from the path that goes from e_{c^y} to e_{c^x}.

Note that $f(e_{c^x}, e_{c^y}) + f(e_{c^y}, e_{c^x}) - w(e_{c^x}) - w(e_{c^y})$ indeed computes the sum of weights of all gray edges minus the sum of weights of all black edges. Therefore, a cycle C is balanced if $f(e_{c^x}, e_{c^y}) + f(e_{c^y}, e_{c^x}) - w(e_{c^x}) - w(e_{c^y}) = 0$, positive if $f(e_{c^x}, e_{c^y}) + f(e_{c^y}, e_{c^x}) - w(e_{c^x}) - w(e_{c^y}) > 0$, and negative if $f(e_{c^x}, e_{c^y}) + f(e_{c^y}, e_{c^x}) - w(e_{c^x}) - w(e_{c^y}) < 0$.

We follow by presenting several lemmas, that will later be combined to prove the correctness of our 3.5-approximation algorithm. Let us first present the ideas behind them. Due to space constraints, proofs of Lemmas 6–13 are omitted. However, it can be seen in Fig. 4 how (and in which case) each of these lemmas is applied, as explained in detail right after the lemmas.

The next two lemmas deal with non-trivial negative cycles. Lemmas 6 and 8 show that it is always possible to increase balanced cycles by applying one transposition on negative cycles that are non-oriented and oriented, respectively.

Lemma 6. *Let C be a non-trivial non-oriented negative cycle. There is a transposition that increases the number of balanced cycles by one.*

Lemma 7. *Let $C = (c^1, \ldots, c^k)$ be a non-trivial oriented cycle. If C is not positive, there is a triple (c^x, c^y, c^z) with $c^x > c^z > c^y$ and $1 \leq x < y < z \leq k$ such that $0 \leq f(e_{c^x}, e_{c^y}) \leq w(c^x) + w(c^y)$, or $0 \leq f(e_{c^y}, e_{c^z}) \leq w(c^y) + w(c^z)$, or $0 \leq f(e_{c^z}, e_{c^x}) \leq w(c^z) + w(c^x)$.*

Lemma 8. *Let C be an oriented long negative cycle. There is a transposition that increases the number of balanced cycles by 1 and the number of cycles by 2.*

Now let us explain how to deal with trivial negative cycles. We use Lemma 9 as an intermediary step to the correctness of Lemma 10, that shows how many transpositions are needed to transform trivial negative cycles into trivial balanced.

Lemma 9. *It is possible to make any redistribution of weights of three distinct black edges e_{b^1}, e_{b^2}, and e_{b^3} using two transpositions.*

Lemma 10. *Let G be a weighted cycle graph in which all negative cycles are trivial. Then, there are two transpositions that increase the number of balanced cycles by 2.*

The next three lemmas dealing with non-trivial balanced cycles. Lemmas 11, 12 and 13 show that it is possible to increase balanced cycles by applying transpositions on oriented long balanced cycles, non-oriented long balanced cycles, and short balanced cycles, respectively.

Lemma 11. *Let C be an oriented long balanced cycle. Then it is possible to increase the number of balanced cycles by two after at most three transpositions.*

Lemma 12. *Let C be a non-oriented long balanced cycle in a graph with no oriented cycles. It is possible to increase the number of balanced cycles by four after at most seven transpositions.*

Lemma 13. *Let G be a graph with no long cycles such that all cycles are balanced. If G has short cycles it is possible to increase the number of balanced cycles by two after two transpositions.*

In Fig. 4(a) the blue cycle $A = (6, 4, 2)$ is non-oriented and negative, so we can apply Lemma 6 using the positive cycle $B = (7, 5)$, and the intergenic transposition $\rho^{(2,6,7)}_{(5,4,1)}$ generates in Fig. 4(b) the trivial balanced cycle $C = (2)$, and the non-trivial cycle $D = (7, 5, 3, 6)$, that in this case is negative and oriented. We then can use Lemma 8 on D, and the transposition $\rho^{(3,6,7)}_{(2,1,0)}$ generates in Fig. 4(c) the trivial balanced cycle $E = (3)$, the trivial cycle $F = (7)$, and the short cycle $G = (6, 4)$. Note that F is negative and G is balanced. At this stage, there is no long cycle, and we can use Lemma 10 on the trivial negative cycle F. This lemma requires a positive cycle to interact with the trivial negative, and $H = (8)$ is the only positive cycle. Since H is also trivial, we need to borrow a black edge of a balanced cycle to apply the transposition, so let us use the short balanced cycle G. Two consecutive transpositions on these black edges (see

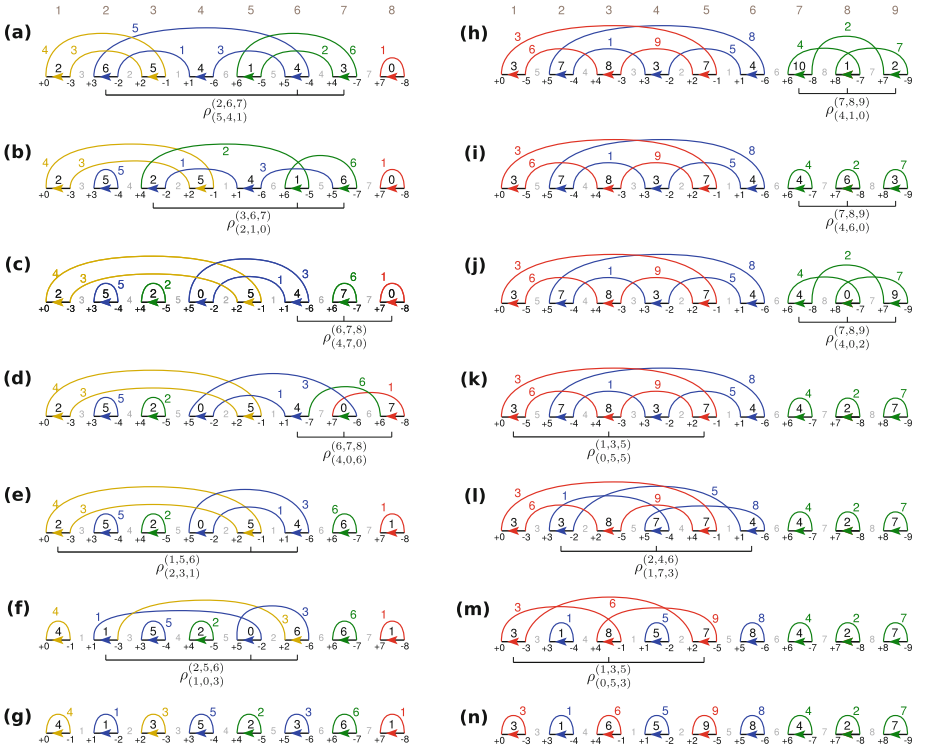

Fig. 4. Black egdes are drawn as horizontal lines, and their labels are on the top of the figure. Gray edges are drawn as arcs. **(a)–(g)** shows a sequence of intergenic transpositions that sorts the instance $(\pi, \breve{\pi}, \breve{\iota})$ with $\pi = (3\ 2\ 1\ 6\ 5\ 4\ 7)$, $\breve{\pi} = (2, 6, 5, 4, 1, 4,\ 3, 0)$, and $\breve{\iota} = (4, 1, 3, 5, 2, 3, 6, 1)$ using Lemmas 6–10 and 13. **(h)–(n)** shows a sequence of intergenic transpositions that sorts the instance $(\pi, \breve{\pi}, \breve{\iota})$ with $\pi = (5\ 4\ 3\ 2\ 1\ 6\ 8\ 7), \breve{\pi} = (3, 7, 8, 3, 7, 4, 10, 1, 2)$, and $\breve{\iota} = (3, 1, 6, 5, 9, 8, 4, 2, 7)$ using Lemmas 11 and 12. (Color figure online)

Fig. 4(c) and (d)) generate two balanced cycles $I = (7)$ and $J = (8)$ in Fig. 4(e), without modifying G. The weighted cycle graph in Fig. 4(e) has only balanced cycles, and they are either short or trivial. In this case we can use Lemma 13 that applies two intergenic transpositions to two non-oriented short cycles, in this case $G = (6, 4)$ and $K = (5, 1)$. The two consecutive transpositions applied to G and K (see Fig. 4(e) and (f)) generate four balanced cycles, increasing the number of balanced cycles by 2, and completing the sorting process – the weighted cycle graph in Fig. 4(g) has only balanced cycles.

In Fig. 4(h) the green cycle $A' = (9, 7, 8)$ is oriented, and since it is also balanced we can use Lemma 11. The three transpositions in Fig. 4(h–j) breaks A' into three balanced cycles. In Fig. 4(k) we have only balanced non-oriented cycles, and we can use Lemma 12 that applies a transposition followed by Lemma 11

twice. The first transposition is applied over cycle $B' = (5, 3, 1)$ transforming the blue cycle in Fig. 4(k) into an oriented balanced cycle $C' = (6, 2, 4)$. The first application of Lemma 11 breaks C' into three balanced cycles (in this case, one transposition is sufficient) and also transforms the non-oriented cycle B' into the oriented cycle $B'' = (5, 1, 3)$ (from Fig. 4(l) to (m)). The second application of Lemma 11 breaks B'' into three balanced cycles (using again only one transposition), completing the sorting process – Fig. 4(g) has only balanced cycles.

5 The 3.5-Approximation Algorithm

Algorithm 1 focuses on applying transpositions that increase the number of balanced cycles, following a sequence of steps using Lemmas 6, 8, 10, 11, 12, and 13.

Algorithm 1. a 3.5-approximation algorithm for SbIT.

Data: an instance $(\pi, \breve{\pi}, \breve{\iota})$.
Result: a sequence $\rho_1, \rho_2, \ldots, \rho_m$ such that $(\pi, \breve{\pi}) \cdot \rho_1 \cdot \rho_2 \cdot \ldots \cdot \rho_m = (\iota, \breve{\iota})$.

```
1  sequence ← ∅
2  while (π, π̆, ῐ) ≠ (ι, ῐ, ῐ) do
3      G ← G(π, π̆, ῐ)
4      if there exists an oriented cycle C in G that is either balanced or negative
         then
5          if C is negative then
6              S_ρ ← transposition from Lemma 8
7          else S_ρ ← transpositions from Lemma 11
8      else if there exists a negative cycle C in G that is either non-oriented or
         trivial then
9          if C is non-oriented then
10             S_ρ ← transposition from Lemma 6
11         else S_ρ ← transpositions from Lemma 10
12     else
13         if there exists a long balanced cycle C in G then
14             S_ρ ← transpositions from Lemma 12
15         else S_ρ ← transpositions from Lemma 13
16     (π, π̆, ῐ) ← (π, π̆, ῐ) · S_ρ
17     sequence.append(S_ρ)
18 return sequence
```

Let us briefly show the correctness of Algorithm 1, i.e., it stops and reaches $(\iota, \breve{\iota}, \breve{\iota})$.

While $(\pi, \breve{\pi}, \breve{\iota}) \neq (\iota, \breve{\iota}, \breve{\iota})$ we have that one of the following must be true: (i) there is an oriented cycle. in this scenario we break this cycle if it is balanced or negative on lines 5 and 7; (ii) there is a negative cycle (considering that they are not oriented): we create balanced cycles if it is non-oriented or trivial on lines 9

and 11; and (iii) cycles are all balanced, and there is no oriented cycles. If there is a long cycle we break it at line 13, and we break short cycles at line 15.

Note that we did not care about the positive oriented cycles: they become either balanced or negative before the algorithm uses (iii), and will be handled in (i) at some point. If the algorithm reaches (iii) then all cycles are balanced since any negative cycle is handled by (i) and (ii).

Concerning the complexity of Algorithm 1, the loop of lines 2–17 iterates up to $m = n+1$ times. Since each time the algorithm applies one of those lemmas, it increases the number of balanced cycles by at least one. Finding which lemma to use (and at which positions the transposition takes place) requires $O(n^2)$ time. Thus, the overall complexity of Algorithm 1 is $O(n^3)$.

Now let us discuss about the approximation factor Algorithm 1 guarantees. Note that some of the steps of the algorithm require more than one transposition, so the approximation factor will be computed as follows:

Definition 14. *Let $\mathcal{S}_\rho = (\rho_1, \rho_2, \ldots, \rho_d)$ be a sequence of transpositions such that $(\pi, \breve{\pi}, \breve{\iota}) \cdot \mathcal{S}_\rho = (\sigma, \breve{\sigma}, \breve{\iota})$. By Lemma 4, $\mathcal{S}_\rho$ creates up to $2d$ balanced cycles. Therefore, the approximation factor is at most $\frac{\mathbf{2d}}{\boldsymbol{c_b(\sigma,\breve{\sigma},\breve{\iota}) - c_b(\pi,\breve{\pi},\breve{\iota})}}$.*

The following lemma shows that each step of Algorithm 1 guarantees an approximation factor of 3.5 or less, which leads to the 3.5-approximation algorithm we propose.

Lemma 15. *Algorithm 1 has an approximation factor of* 3.5.

Proof. We use the formula from Definition 14 to calculate the approximation factor of each step.

- Step using Lemma 8 creates at least one new balanced cycle using one transposition, which leads to the maximum approximation $\frac{2}{1} = 2$.
- Step using Lemma 11 creates at least two new balanced cycles using up to three transpositions, so its maximum approximation factor is $\frac{6}{2} = 3$.
- Step using Lemma 6 creates a new balanced cycles using one transposition, and its approximation is $\frac{2}{1} = 2$.
- Step using Lemma 10 it creates two new balanced cycles using two transpositions, so its approximation is $\frac{4}{2} = 2$.
- Step using Lemma 12 creates four new balanced cycle using up to seven transpositions, and it follows that the maximum approximation is $\frac{14}{4} = 3.5$.
- Step using Lemma 13 creates two new balanced cycles using two transpositions, so it follows that its approximation is $\frac{4}{2} = 2$. □

6 Conclusion

We adapted the breakpoint graph to represent both gene order and intergenic sizes, and investigated properties of this new graph structure during a sorting process. As a result, we were able to design an approximation algorithm for

the Sorting by Intergenic Transpositions. We also show that this problem is NP-Hard.

As future works, one can explore a problem where the probability of an intergenic region being affected by transpositions is related to its size, i.e., when genome rearrangements are more likely to cut the genome on bigger intergenic regions. One can also investigate the use of reversals and transpositions on signed permutations along with intergenic regions.

Acknowledgments. This work was supported by the National Council for Scientific and Technological Development - CNPq (grants 400487/2016-0, 425340/2016-3, and 140466/2018-5), the São Paulo Research Foundation - FAPESP (grants 2013/08293-7, 2015/11937-9, 2017/12646-3, and 2017/16246-0), the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, and the CAPES/COFECUB program (grant 831/15).

References

1. Bafna, V., Pevzner, P.A.: Sorting by transpositions. SIAM J. Discrete Math. **11**(2), 224–240 (1998). https://doi.org/10.1137/S089548019528280X
2. Belda, E., Moya, A., Silva, F.J.: Genome rearrangement distances and gene order phylogeny in γ-proteobacteria. Mol. Biol. Evol. **22**(6), 1456–1467 (2005). https://doi.org/10.1093/molbev/msi134
3. Biller, P., Guéguen, L., Knibbe, C., Tannier, E.: Breaking good: accounting for fragility of genomic regions in rearrangement distance estimation. Genome Biol. Evol. **8**(5), 1427–1439 (2016). https://doi.org/10.1093/gbe/evw083
4. Biller, P., Knibbe, C., Beslon, G., Tannier, E.: Comparative genomics on artificial life. In: Beckmann, A., Bienvenu, L., Jonoska, N. (eds.) CiE 2016. LNCS, vol. 9709, pp. 35–44. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40189-8_4
5. Brito, K.L., Jean, G., Fertin, G., Oliveira, A.R., Dias, U., Dias, Z.: Sorting by genome rearrangements on both gene order and intergenic sizes. J. Comput. Biol. **27**(2), 156–174 (2020). https://doi.org/10.1089/cmb.2019.0293
6. Bulteau, L., Fertin, G., Rusu, I.: Sorting by transpositions is difficult. SIAM J. Comput. **26**(3), 1148–1180 (2012). https://doi.org/10.1137/110851390
7. Bulteau, L., Fertin, G., Tannier, E.: Genome rearrangements with indels in intergenes restrict the scenario space. BMC Bioinform. **17**(S14), 225–231 (2016). https://doi.org/10.1186/s12859-016-1264-6
8. Elias, I., Hartman, T.: A 1.375-approximation algorithm for sorting by transpositions. IEEE/ACM Trans. Comput. Biol. Bioinform. **3**(4), 369–379 (2006). https://doi.org/10.1109/TCBB.2006.44
9. Fertin, G., Jean, G., Tannier, E.: Algorithms for computing the double cut and join distance on both gene order and intergenic sizes. Algorithm Mol. Biol. **12**(16), 1–11 (2017). https://doi.org/10.1186/s13015-017-0107-y
10. Hannenhalli, S., Pevzner, P.A.: Transforming men into mice (polynomial algorithm for genomic distance problem). In: Proceedings of the 36th Annual IEEE Symposium Foundations of Computer Science (FOCS 1995), pp. 581–592. IEEE Computer Society Press, Washington, DC (1995). https://doi.org/10.1109/SFCS.1995.492588

11. Kececioglu, J.D., Sankoff, D.: Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. Algorithmica **13**, 180–210 (1995). https://doi.org/10.1007/BF01188586
12. Lin, Y., Rajan, V., Moret, B.M.E.: TIBA: a tool for phylogeny inference from rearrangement data with bootstrap analysis. Bioinformatics **28**(24), 3324–3325 (2012). https://doi.org/10.1093/bioinformatics/bts603
13. Oliveira, A.R., Jean, G., Fertin, G., Dias, U., Dias, Z.: Super short operations on both gene order and intergenic sizes. Algorithm Mol. Biol. **14**(21), 1–17 (2019). https://doi.org/10.1186/s13015-019-0156-5
14. Wang, L.S., Warnow, T., Moret, B.M.E., Jansen, R.K., Raubeson, L.A.: Distance-based genome rearrangement phylogeny. J. Mol. Evol. **63**(4), 473–483 (2006). https://doi.org/10.1007/s00239-005-0216-y