Discrete Applied Mathematics 274 (2020) 26–34

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

On the computational complexity of closest genome problems

Luís Felipe I. Cunha [a,*], Pedro Feijão [b], Vinícius F. dos Santos [c], Luis Antonio B. Kowada [d], Celina M.H. de Figueiredo [a]

[a] Universidade Federal do Rio de Janeiro, Brazil
[b] Simon Fraser University, Canada
[c] Universidade Federal de Minas Gerais, Brazil
[d] Universidade Federal Fluminense, Brazil

ARTICLE INFO

Article history:
Received 13 April 2017
Received in revised form 22 January 2019
Accepted 2 April 2019
Available online 27 April 2019

Keywords:
Closest problem
Single-cut-or-join
Breakpoint
Block-interchange

ABSTRACT

Genome rearrangements are events where large blocks of DNA exchange places during evolution. The analysis of these events is a promising tool for understanding evolutionary genomics. Many pairwise rearrangement distances have been proposed, based on finding the minimum number of rearrangement events to transform one genome into the other, using some predefined operation. When more than two genomes are considered, we have the more challenging problem of rearrangement-based phylogeny reconstruction. One important problem is the Closest Genome Problem (CGP), that aims to find, for a given distance notion, a genome that minimizes the maximum distance to any other, which can be seen as finding a genome in the center of all others. The Hamming Closest String Problem (Hamming-CSP) was already studied and settled to be NP-complete. In this paper, we show that the CGP is NP-complete for well-known genome rearrangement distances, such as the single-cut-or-join, the breakpoint and the block interchange.

1. Introduction

Several genome rearrangement metrics, defined as minimum edit distance problems, are combinatorial challenges with many applications in computational biology [16] and have received much attention in recent years. Among the studied distances are the Cayley, reversal, transposition, block-interchange and breakpoint distances for the case where genomes are represented as permutations [2,6,9]. Genomes have also other models, more suited for representing multichromosomal genomes. Regarding this approach, the double-cut-and-join (DCJ) [10] and single-cut-or-join (SCJ) [8] distances are two commonly used metrics.

When more than two genomes are given as input, a relevant problem is the Median Genome Problem, where we want to find a solution genome that minimizes the sum of the distances between the solution and all others of the input. There are many approaches regarding the problem of finding ancestral genomes. When considering metrics, the Breakpoint and the DCJ median problems are NP-complete [10,15], but the SCJ median problem is polynomially solvable. Haghighi and Sankoff [12] proved that regarding the breakpoint metric (they also conjectured that regarding the double-cut-and-join metric), a tendency for medians is to fall on or to be close to one of the input orders, which contain no information for the phylogeny reconstruction.

* Corresponding author.
E-mail addresses: lfignacio@cos.ufrj.br (L.F.I. Cunha), pfeijao@sfu.ca (P. Feijão), viniciussantos@dcc.ufmg.br (V.F. dos Santos), luis@ic.uff.br (L.A.B. Kowada), celina@cos.ufrj.br (C.M.H. de Figueiredo).

https://doi.org/10.1016/j.dam.2019.04.002
0166-218X/

Since in some cases medians give no new information, an alternative problem is the Closest Genome Problem (CGP), that aims to find a genome that minimizes the maximum distance to any other, which can be seen as finding a genome in the center of all others, i.e. a genome corresponding to the *radius* of the input set.

Lanctot et al. [14] studied the CGP for strings regarding the Hamming distance, and settled that this problem is NP-complete. Popov [17] studied a version on permutations regarding the Cayley metric, and showed that the Cayley-CGP is NP-complete. To the best of our knowledge, the CGP has not been studied for other polynomially solvable distance problems. We focus only on distance problems that are polynomially solvable, since if the distance problem is NP-complete for a given metric M, then M-CGP is also NP-complete.

In this paper, we consider the Closest Genome Problem with respect to different genome models and rearrangement metrics. For multichromosomal genomes, the SCJ distance is a simple model with the advantage of simplifying problems that are NP-hard under other models, such as the genome median and small parsimony, for which a solution can be found in polynomial time [8]. Somewhat surprisingly, we show that SCJ-CGP is an NP-complete problem. For the breakpoint and block-interchange distances, which are well-known metrics where genomes are modeled as permutations, we show that Breakpoint-CGP and the Block interchange-CGP are NP-complete.

This paper is organized as follows: in Section 2 we report the Hamming Closest String Problem, which is the basis of our NP-completeness reductions, and the metrics we deal with in this paper; in Section 3 we prove NP-completeness, where in Section 3.1 we prove that SCJ-CGP is NP-complete; in Section 3.2 we prove that Breakpoint-CGP is NP-complete and in Section 3.3 we prove that Block-interchange-CGP is NP-complete; and finally in Section 4 we discuss some open questions for further work about complexity and approximation algorithms on the closest and the median problems.

2. Preliminaries

Input genomes can be uni- or multichromosomal, and chromosomes might be linear or circular. To model multichromosomal genomes, some definitions are needed. Let $G = \{1, \ldots, n\}$ be a set of *genes*, representing an oriented segment of DNA, and let $E = \bigcup_{i \in G} \{i^h, i^t\}$ be the set of *gene extremities*, representing the head and tail of each gene. An *adjacency* is an unoriented pair of extremities of a same gene or of distinct genes. A *general genome*, or simply a *genome*, is represented by a set of adjacencies where the tail and head of each gene appear at most once. An extremity that is not connected to any other is a *telomere*. For instance, for a gene set $G = \{1, 2, 3, 4\}$, $A = \{1^h2^t, 2^h3^h, 4^h4^t\}$ represents a genome, composed of two chromosomes, one linear and one circular. Note that extremities 1^t and 3^t are telomeres and therefore are unambiguously omitted from the genome representation.

Genome rearrangements are events where large blocks of DNA exchange places during evolution. For some genome rearrangement models, we may consider genomes as strings or as permutations, which are particular strings, as we see next.

An *alphabet* Σ is a non empty set of letters, and a *string* over Σ is a finite sequence of letters of Σ. The *Hamming distance of two strings* of the same length s and σ denoted $d_H(s, \sigma)$ is defined as the number of mismatched positions between s and σ. The *Hamming distance of a string* s of length m denoted $d_H(s)$ is the Hamming distance of s and $\iota = 0^m$.

A *permutation* of length n is a particular string with a unique occurrence of each letter, since it is a bijection from the set $\{1, 2, \ldots, n\}$ onto itself $\pi = [\boldsymbol{\pi(0)}\pi(1)\pi(2)\cdots\pi(n)\boldsymbol{\pi(n+1)}]$, such that $\boldsymbol{\pi(0)} = 0$ and $\boldsymbol{\pi(n+1)} = n + 1$. The operations we consider will never act on $\boldsymbol{\pi(0)}$ nor $\boldsymbol{\pi(n+1)}$.

The *union* of the permutations α and β of lengths n and m, respectively, is the permutation π constructed by the juxtaposition of α and β, $\pi = [\mathbf{0}\ \alpha(1)\ \alpha(2)\ \ldots\alpha(n)\ \beta(1) + n\ \beta(2) + n\ \ldots\ \beta(m) + n\ \boldsymbol{n + m + 1}]$. For instance the permutation [**0** 1 2 3 4 7 6 5 8 **9**] is the union of [**0** 1 2 3 4 **5**] and [**0** 3 2 1 4 **5**].

Given a metric M and $d_M(p, \pi)$ the distance between permutations p and π with respect to the metric M, the *distance of a permutation* π of length m denoted $d_M(\pi)$ is the distance between π and the *identity permutation* $\iota = [\mathbf{0}\ 1\ 2\ \cdots\ n\ \boldsymbol{n\ +\ 1}]$.

Closest genome problem. The Metric-M Closest Genome Problem is defined as follows:

> Metric-M Closest Genome Problem (M-CGP)
> INPUT: Set of genomes $\{g_1, g_2, \ldots, g_\ell\}$, and a non-negative integer f.
> QUESTION: Is there a genome σ such that $\max_{i=1,\ldots,\ell} d_M(g_i, \sigma) \leq f$?

If there is a positive answer for M-CGP, then a *solution of M-CGP* is any genome σ that satisfies $\max_{i=1,\ldots,\ell} d_M(g_i, \sigma) \leq f$.

The CGP was considered through the version of strings with respect to the Hamming distance, it was proposed by Lanctot et al. [14] as the *closest string problem* (CSP), defined by given a set of strings $\{s_1, s_2, \ldots, s_\ell\}$ over an alphabet Σ of length m each, and a non-negative integer f, one wants to find a string σ of length m such that $\max_{i=1,\ldots,\ell} d_H(s_i, \sigma) \leq f$.

The CSP was proposed in the context of some biological problems, such as: discovering potential drug targets, creating diagnostic probes, universal primers or unbiased consensus sequences. All these problems reduce to the task of finding a pattern that, with some error, occurs in one set of strings (closest string problem). Lanctot et al. [14] also proved that the Hamming-CSP is NP-complete for a binary alphabet.

Given a set of genomes (general genomes, strings or permutations), the Closest Genome Problem aims to find a solution genome that minimizes the maximum distance between the solution and all other input genomes. The metric of distances depends on the context of the problem.

Metrics. In this work, we consider the distances of *single-cut-or-join* on general genomes, and of *breakpoint* and *block-interchange* on permutations, for which these distance problems were proposed in the context of genome rearrangements. Similarly to the Hamming distance on strings with respect to the CSP problem, which was motivated by applications in bioinformatics, the metrics studied in the present paper are much studied by the bioinformatics community [6,8,9].

The *single-cut-or-join* is an operation that transforms one genome into another one by a *cut* or a *join* to transform a uni- or multichromosomal genome into another one. A *cut* breaks a pair of consecutive elements, by breaking a linear chromosome in two, or cutting a circular chromosome in a linear one. A *join* is the inverse operation of a cut, merging two free extremities.

The *breakpoint distance* is the number of consecutive elements in one permutation that are not consecutive in another one. Note that on the breakpoint distance we do not apply any operation to transform a permutation into another one.

The *block-interchange* operation transforms one permutation into another one by exchanging two blocks, and generalizes the *transposition* operation (whose blocks are consecutive) and the *Cayley* operation (whose blocks are unitary). The *block-interchange distance of two permutations* is the minimum number of block-interchanges to transform one permutation into another one, and the *block-interchange distance of a permutation* π is the block-interchange distance between π and ι.

Relationship between distances and between closest problems. Note that a block-interchange generalizes a transposition and generalizes also a Cayley operation. Nevertheless, with respect to the distance problem, general operations do not imply the same computational complexity of more particular operations. For instance, with respect to the block-interchange distance, it can be computed in polynomial time [6], whereas the transposition distance is an NP-complete problem [4], and the Cayley distance is a polynomial problem.

On the other hand, if a distance problem is NP-complete, then the closest problem for the same operation is also NP-complete. Indeed, by considering the input set of permutations with two permutations π, ι such that $\pi \neq \iota$ and we ask for a permutation with distance for a metric M at most d for each, we can see the distance as the closest problem with a particular instance. Since, by the triangular inequality, it is necessary that $d_M(\pi, \iota) \leq 2d$.

Although the Transposition-CGP is NP-complete, if we consider some particular input sets then we can determine a closest permutation in polynomial time. For instance, let us consider the *toric equivalence*, which is an equivalence relation between permutations, where permutations in the same class have the same transposition distance [7]. Given a permutation for which we know its transposition distance (let us say it is equal to d), hence we know the distances of any other permutation and the distance of any pair of permutations in the same toric class. Therefore, if any $d' \geq d$ is the input radius asked in the problem, then the identity permutation is a solution permutation. Another example is to consider $n \leq 15$, once we know the exact distance of any permutation as discussed in [7] (let us say d is the maximum distance of any permutation). Therefore, if any $d' \geq d$ is chosen to be the radius in the CGP, then the identity permutation is a solution.

We review next how to obtain in polynomial time the distances between general genomes, and between permutations based on the breakpoints and on the reality and desire diagram.

The single-cut-or-join distance. The *single-cut-or-join (SCJ) distance* between two genomes A and B is the minimum number of cuts and joins necessary to transform A in B where a cut breaks an adjacency and a join glues two separate extremities. This distance can be calculated with $d_{SCJ}(A, B) = |A \backslash B| + |B \backslash A|$ [8].

The breakpoint distance. An *adjacency in a permutation* (resp. a *reverse adjacency in a permutation*) π with respect to δ is a pair (a, e) of consecutive elements in π such that (a, e) (resp. the pair (e, a)) is also consecutive in δ. If a pair of consecutive elements is neither an adjacency nor a reverse adjacency, then (a, e) is called a *breakpoint*, and we denote $b(\pi, \delta)$ the number of breakpoints of π with respect to δ. Hence, the breakpoint distance between π and δ is $d_{BP}(\pi, \delta) = b(\pi, \delta)$.

The block-interchange distance. Bafna and Pevzner [2] proposed a useful graph, the reality and desire diagram, which allowed non-trivial bounds on the transposition distance [2], and also provided, as established by Christie [6], the exact block-interchange distance.

Given a permutation π of length n, the *reality and desire diagram* $G(\pi)$ of π, is a multigraph $G(\pi) = (V, R \cup D)$, where $V = \{0, -1, +1, -2, +2, \ldots, -n, +n, -(n+1)\}$, each element of π corresponds to two vertices and we also include the vertices labeled by 0 and $-(n+1)$, and the edges are partitioned into two sets: the reality edges R and the desire edges D. The *reality edges* represent the adjacency between the elements on π, that is $R = \{(+\pi(i), -\pi(i+1)) \mid i = 1, \ldots, n-1\} \cup \{(0, -\pi(1)), (+\pi(n), -(n+1))\}$; and the *desire edges* represent the adjacency between the elements on ι, that is $D = \{(+i, -(i+1)) \mid i = 0, \ldots, n\}$. Fig. 1 illustrates the reality and desire diagram of a permutation.

As a direct consequence of the construction of this graph, every vertex in $G(\pi)$ has degree 2, so $G(\pi)$ can be partitioned into disjoint *cycles*. We say that a cycle in π has length k, or that it is a k-cycle, if it has exactly k reality edges (or, equivalently, k desire edges). Hence, the identity permutation of length n has $n+1$ cycles of length 1. We denote $C(G(\pi))$ the number of cycles in $G(\pi)$.

After applying a block-interchange $b\ell$ in a permutation π, the number of cycles $C(G(\pi))$ changes in such a way that: $C(G(\pi b\ell)) = C(G(\pi)) + x$, such that $x \in \{-2, 0, 2\}$. The block-interchange $b\ell$ is thus classified as an *x-move* for π.

Christie [6] proved for the block-interchange operation the existence of a 2-move for any permutation, which says that the number of cycles yields the exact block-interchange distance:

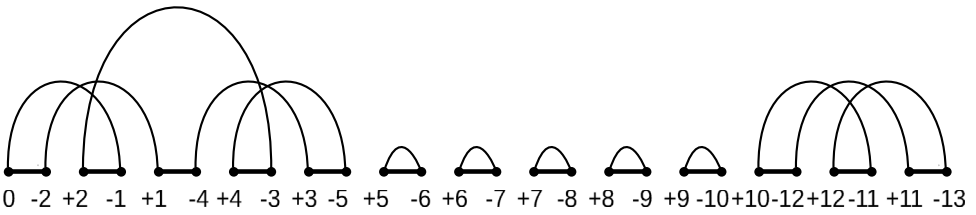

Fig. 1. The reality and desire diagram of the permutation [**0** 2 1 4 3 5 6 7 8 9 10 12 11 **13**], where bold edges are reality edges, and the others are the desire edges.

Theorem 2.1 (*[6]*). *The block-interchange distance of a permutation π of length n is* $d_{BI}(\pi) = \frac{(n+1)-C(G(\pi))}{2}$.

On the other hand, by allowing only the particular case of the transposition operation, a 2-move is not always possible to be found. We say a transposition *affects* a cycle if the extremities of the two blocks of the transposition eliminate a reality edge of a cycle and create another edge. This new edge may increase, decrease, or keep the number of cycles.

Bafna and Pevzner [2] showed conditions of a cycle for a transposition to be an x-move. If a transposition t is a -2-move, then t affects three distinct cycles. However, if a transposition t is a 0-move or a 2-move, then t affects at least two elements of the same cycle [2]. We shall see that, when considering the block-interchange operation for the CGP, it is useful to apply 0-move transpositions.

An interesting transformation in a permutation is the reduction, since the permutation obtained after the transformation preserves the block-interchange distance. The *reduced permutation* of π, denoted $gl(\pi)$, is the permutation whose reality and desire diagram $G(gl(\pi))$ is equal to $G(\pi)$ without the cycles of length 1, and has its vertices relabeled accordingly. For instance the reduced permutation corresponding to the permutation in Fig. 1 is [**0** 2 1 4 3 5 7 6 **8**]. Christie [6] proved an important equality.

Theorem 2.2 (*[6]*). *The block-interchange distances of a permutation π and its reduced permutation $gl(\pi)$ satisfy* $d_{BI}(\pi) = d_{BI}(gl(\pi))$.

3. NP-completeness for CGP

Next, we present reductions from Hamming-CSP to CGP. We apply transformations from a generic instance of Hamming-CSP to particular instances of the M-CGP with respect to the single-cut-or-join, the breakpoint, and the block-interchange metrics. In each one, we establish a relationship between the Hamming distance of binary strings and the distance on the corresponding metric. We refer to Section 2 for definitions and properties of: Hamming distances between two strings s and σ, $d_H(s, \sigma)$, and of a string s, $d_H(s)$; single-cut-or-join distance between two genomes A and B, $d_{SCJ}(A, B)$; breakpoint distances between two permutations π and δ, $d_{BP}(\pi, \delta)$, and of a permutation π, $d_{BP}(\pi)$; and block-interchange distances between two permutations p and π, $d_{BI}(p, \pi)$, and of a permutation π, $d_{BI}(\pi)$.

3.1. Single-cut-or-join–CGP *is* NP-*complete*

For any string s of length m on $\Sigma = \{0, 1\}$, let $g(s)$ represent a genome on the gene set $G = \{1, \ldots, m\}$ obtained next in Algorithm 1.

Algorithm 1: $Genome_{SCJ}(s)$

input : Binary string s of length m.
output: Genome $g(s)$
1 each occurrence of 0 in position k corresponds to the telomeres k^h and k^t.
2 each occurrence of 1 in position k corresponds to the adjacency $k^h k^t$.

For instance, for $s = 1011$, we have that $g(s) = \{1^h 1^t, 3^h 3^t, 4^h 4^t\}$ (2^h and 2^t are telomeres, omitted in the genome set). Therefore, for any two strings p and r of length m, it follows that $d_H(p, r) = d_{SCJ}(g(p), g(r))$, since for each position k where p and r are different, there is exactly one adjacency in $(g(p) \backslash g(r)) \cup (g(r) \backslash g(p))$.

Lemma 3.1. *Given a set of strings $s_1, s_2, \ldots, s_\ell$ over alphabet $\Sigma = \{0, 1\}$ of length m each, and a non-negative integer d, there is a Hamming closest string σ of length m such that* $\max_{i=1,\ldots,\ell} d_H(s_i, \sigma) \leq d$ *if, and only if, there is a SCJ closest genome π such that* $\max_{i=1,\ldots,\ell} d_{SCJ}(g(s_i), \pi) \leq d$.

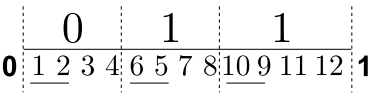

Fig. 2. Permutation β_s = [**0** 1 2 3 4 6 5 7 8 10 9 11 12 **13**] where $s = 011$, obtained from Algorithm 2, with the breakpoint distance $d_{BP}(\beta_s) = 4$, since the breakpoints are (4, 6), (5, 7), (8, 10) and (9, 11). The Hamming distance of s is $d_H(s) = \frac{d_{BP}(\beta_s)}{2} = 2$.

Proof. ($\Rightarrow$) Since $d_H(p, r) = d_{SCJ}(g(p), g(r))$ for any two strings p and r of length m, it follows that the genome $\pi = g(\sigma)$ satisfies $\max_{i=1,\ldots,\ell} d_{SCJ}(g(s_i), \pi) \leq d$.

($\Leftarrow$) On the other hand, if there is a *SCJ* closest genome π with maximum distance d from all genomes in the set $\{g(s_1), \ldots, g(s_\ell)\}$, then let π' be the genome where we remove any adjacency from π that is not on the form $k^h k^t$.

Clearly, $d_{SCJ}(g(s_i), \pi') \leq d_{SCJ}(g(s_i), \pi)$ for $i = 1, \ldots, \ell$, and π' is also a *SCJ* closest genome. Then, the string σ with a 1 on every position k where an adjacency $k^h k^t$ is present on π', and 0 otherwise, $\max_{i=1,\ldots,\ell} d_H(s_i, \sigma) \leq d$, and is a Hamming closest string with maximum distance d. □

Lemma 3.1 implies the following result.

Theorem 3.1. *The* Single-cut-or-join-CGP *is* NP*-complete.*

3.2. Breakpoint–CGP *is* NP*-complete*

Firstly, we apply Algorithm 2 that transforms an arbitrary binary string s of length m into a particular permutation β_s of length $4m$.

Algorithm 2: $Permut_{BP}(s)$

input : Binary string s of length m.
output: Permutation β_s

1 each occurrence of 0 in position i corresponds to the elements $4i-3$, $4i-2$, $4i-1$ and $4i$ in positions $4i-3$, $4i-2$, $4i-1$, $4i$, respectively.
2 each occurrence of 1 in position i corresponds to the elements $4i-3$, $4i-2$, $4i-1$ and $4i$ in positions $4i-2$, $4i-3$, $4i-1$, $4i$, respectively.

Note that any permutation obtained in Algorithm 2 is constructed by successive unions of [**0** 1 2 3 4 **5**] and [**0** 2 1 3 4 **5**]. Fig. 2 illustrates the construction of a permutation for a given string with respect to Algorithm 2.

Next, we establish the following relationship between the Hamming distance of an input string and the Breakpoint distance of its output permutation obtained from Algorithm 2.

Lemma 3.2. *Given β_s the permutation obtained from a binary string s by Algorithm 2, the Breakpoint distance of β_s is $d_{BP}(\beta_s) = 2d_H(s)$.*

Proof. Each element 1 of the binary string yields an exchange between two consecutive elements, hence we are creating exactly two breakpoints. □

Now, we show how a solution for the Hamming-CSP implies a solution for the Breakpoint-CPP, and vice versa.

Lemma 3.3. *Given a set of k permutations obtained by Algorithm 2, there is a breakpoint closest permutation with max distance equal to $2d$ if, and only if, there is a Hamming closest string with max distance equal to d.*

Proof. ($\Rightarrow$) If β' can be built by Algorithm 2 for some input string s', then, by Lemma 3.2, s' is a closest string.

Otherwise (similar to Lemma 3.7), given any solution permutation we search from the left to the right to find the first position where the corresponding element is different from the one intended to be by the algorithm. In each case, we transform to a new permutation with a longer prefix agreeing with the algorithm output, without increasing the distance to every input permutation. By repeating this process a string agreeing with the algorithm output can be found and, by Lemma 3.2, a string with maximum distance equal to d can be constructed.

We consider each case of a position not agreeing with an element intended to be by Algorithm 2.

- position $4i-3$: We call $a = 4i-3$ and $b = 4i-2$ the possible elements that could be in this position.
  - If a and b are not consecutive to the right of the position $4i-3$, then there is a universal breakpoint, *i.e.* a breakpoint with respect to all input permutations, on the right of a or on the right of b. In this case we apply a flipping from the position $4i-3$ until such universal breakpoint.

- If a and b appear consecutive to the right of the position $4i-3$, then there is a universal breakpoint. If such breakpoint is on the right of a (in the case of b, a) or on the right of b (in the case of a, b), then we apply a flipping from the position $4i-3$ until such universal breakpoint.
  From now on, we consider a and b consecutive, but there is a breakpoint on the left of such pair.
- If there is a universal breakpoint on the right of the pair a, b (or b, a), then we apply a transposition such that the first block starts at position $4i-3$ and ends at the element before the pair a, b (or b, a), and the second block ends at the universal breakpoint.
- If there is not a universal breakpoint on the right of the pair a, b (or b, a), then every consecutive pair after the position $4i-3$ has difference at most 2, since any pair of consecutive elements is also a pair in an input permutation, for which by Algorithm 2 such property holds. Let x, y, z be the elements in positions $4i-3, 4i-2, 4i-1$, respectively. Since the a, b (or b, a) are both less than x, hence the pair $x-1, x+1$ appears consecutive and on the right of a, b (or b, a), otherwise such difference would be greater than 2, since x is in position $4i-3$.
  - If the transposition putting x between $x-1$ and $x+1$ does not create any breakpoint, then we apply such transposition. The resulted permutation is approximating to our intended solution, since the element $4i-3$ is becoming closer to the intended position.
  - If such transposition putting x between $x-1$ and $x+1$ creates a breakpoint with respect to some input permutation, but it only happens when x, y is an adjacency in such input permutation. In this case, $y = x-2$ or $y = x+2$. Without loss of generality, we assume $y = x-2$. Since the difference between two consecutive elements is at most 2, hence $x-3$ is also on the right of a, b and is consecutive to $x-1$. So, we apply a transposition to put x between $x-1$ and $x+1$, and afterwards we apply a transposition to put $y = x-2$ between $x-3$ and $x-1$. Now, we prove that in this case the number of breakpoints does not increase for no one permutation. For any input permutation, after the transpositions we have at most 3 breakpoints: one on the left of z, another one on the left or the right of x, and another one on the left or the right of y. Before the transpositions we have at least 3 breakpoints: by the construction, one between $x-3$ and $x-1$ or between $x-1$ and $x+1$. Similarly, we have at least one breakpoint between x and y, or between y and z (if yz is not a universal breakpoint, then $z = x-4$, and for the same reason $x, x-2, x-4$ cannot be all adjacencies). Moreover, we have a third breakpoint, on the left of x. Therefore, for any input permutation, the number of breakpoints does not increase after these transpositions.

- position $4i-2$. If $4i-3$ is already correct, we apply a flipping from the element in position $4i-2$ until the element $4i-2$, this operation always decreases the number of breakpoints, since the element $4i-3$ is adjacent to the element $4i-2$ with respect to all input permutations. If after such operation we have created a breakpoint after the position where the $4i-2$ is, then there is no problem, since we are removing one breakpoint in putting $4i-3$ adjacent to $4i-2$. Hence, we are not increasing the number of breakpoints.
- position $4i-1$. If there is a universal breakpoint on the right of the element $4i-1$, then we apply a flipping and the number of breakpoints does not increase. Otherwise, the element $4i$ is adjacent on the right of $4i-1$, hence we apply a transposition putting the $4i-1, 4i$ in the correct positions. In the worst case the number of breakpoints is the same, since we are creating a breakpoint between the elements before $4i-1$ and after the element $4i$, but creating an adjacency between $4i-2$ and $4i-1$.
- position $4i$. In this case we can apply a flipping from the position $4i$ until the position where the element $4i$ is. For the same reason of the second case, the number of breakpoints is not increasing.

($\Leftarrow$) Given a solution string s, we obtain the associated permutation β_s given by Algorithm 2. By Lemma 3.2 we have the solution s regarding the Hamming-CSP corresponding to the permutation β_s with the value of max distance equal to $2d$. □

Lemma 3.3 implies the following result.

Theorem 3.2. *The* Breakpoint-CGP *is* NP*-complete.*

3.3. Block-interchange-CGP *is* NP*-complete*

Firstly, we apply Algorithm 3 that transforms an arbitrary binary string s of length m into a particular permutation λ_s of length $2m$.

Note that any permutation obtained in Algorithm 3 is constructed by successive unions of [**0** 1 2 **3**] and [**0** 2 1 **3**]. Fig. 3 illustrates the construction of a permutation for a given string with respect to Algorithm 3.

Lemma 3.4. *Given a string s of length m and the permutation λ_s of length $2m$ obtained in Algorithm 3, then the reduced permutation $gl(\lambda_s)$ has length n', where $2d_H(s) \leq n' \leq 3d_H(s) - 1$.*

Algorithm 3: $Permut_{Bl}(s)$

input : Binary string s of length m
output: Permutation λ_s
1 each occurrence of 0 in position i corresponds to the elements $2i-1$ and $2i$ in positions $2i-1$ and $2i$, respectively.
2 each occurrence of 1 in position i corresponds to the elements $2i-1$ and $2i$ in positions $2i$ and $2i-1$, respectively.

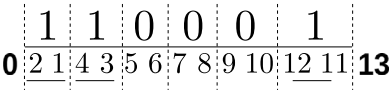

Fig. 3. Permutation $\lambda_s = [\mathbf{0}\ 2\ 1\ 4\ 3\ 5\ 6\ 7\ 8\ 9\ 10\ 12\ 11\ \mathbf{13}]$ obtained from Algorithm 3 – its reality and desire diagram is in Fig. 1 – where $s = 110001$, with reduced permutation $gl(\lambda_s) = [\mathbf{0}\ 2\ 1\ 4\ 3\ 5\ 7\ 6\ \mathbf{8}]$, and number of cycles of its reality and desired diagram $C(G(gl(\lambda_s))) = 2$. The Hamming distance of s is equal to the Block-interchange distance of λ_s.

Proof. If the string s is $s = 1^{d_H(s)}0^{m-d_H(s)}$ (or $s = 0^{m-d_H(s)}1^{d_H(s)}$), then to obtain the reduced permutation of λ_s we remove $2(m - d_H(s))$ elements. Therefore, the associated permutation has length $n' = 2m - 2(m - d_H(s)) = 2d_H(s)$. On the other hand, if s is $s = (01)^{\frac{m}{2}}$ (or $s = (10)^{\frac{m}{2}}$), then each adjacency $2i-1, 2i$ is removed to obtain the reduced permutation, excepted the first adjacency 1,2 (or the last one $2m-1, 2m$), for which both elements are removed. So, the length of the reduced permutation is $n' = 2d_H(s) + d_H(s) - 1$. Since these are the cases of maximum and minimum number of 0s adjacent, hence they correspond to the minimum and maximum lengths of the associated permutations, respectively. □

Lemma 3.5. *If λ_s is a permutation obtained in Algorithm 3 and $gl(\lambda_s)$ its reduced permutation of length $2d_H(s) + x$, for $0 \le x \le d_H(s) - 1$, then $C(G(gl(\lambda_s))) = x + 1$.*

Proof. There exist x contiguous sequences of bits 0 in s, and a sequence of bits 0 is between two contiguous sequences of bits 1. It implies a cycle in the reality and desire diagram for each contiguous sequence of bits 1. □

Next, we establish the key equality between the Hamming distance of an input string and the block-interchange distance of its output permutation obtained from Algorithm 3.

Lemma 3.6. *Given λ_s the permutation obtained from a binary string s by Algorithm 3, the block-interchange distance of λ_s is equal to the Hamming distance of s, $d_{Bl}(\lambda_s) = d_H(s)$.*

Proof. Since $d_{Bl}(\lambda_s) = d_{Bl}(gl(\lambda_s))$, from Lemma 3.5 we have that $d_{Bl}(gl(\lambda_s)) = \frac{2d_H(s)+x+1-(x+1)}{2}$, which implies $d_{Bl}(\lambda_s) = d_H(s)$. □

Now, we show how a solution for the Hamming-CSP implies a solution for the block-interchange-CPP, and vice versa.

Lemma 3.7. *Given a set of k permutations obtained by Algorithm 3, there is a block-interchange closest permutation with max distance at most d if, and only if, there is a Hamming closest string with max distance equal to d.*

Proof. ($\Rightarrow$) If λ' can be built by Algorithm 3 for some input string s', then, by Lemma 3.6, s' is a closest string.

Otherwise, given a solution permutation, we search from the left to the right to find the first position where the corresponding element is different from the one intended to be by the algorithm, which can be a position $2i-1$ or a position $2i$. In each case, we transform to a new permutation with a longer prefix agreeing with the algorithm output, without increasing the distance to any input permutation.

Hence, we apply transpositions on the solution permutation to obtain a new one. To guarantee that the distance of this new permutation and every one of the input is not increasing, we show in each case, the worst operation is a 0-move with respect to every input permutation, since such transposition affects elements of either the same cycle in the reality and desire diagram, or it creates new adjacencies.

By repeating this process, a string agreeing with the algorithm output can be found and, by Lemma 3.6, a string with maximum distance equal to d can be constructed.

1. in position $2i-1$ there is a correct element, but in position $2i$ there is not the element $2i-1$ nor the element $2i$, either.

Given any solution permutation, the elements until the position $2i-1$ are correct, but an element $a > 2i$ is in position $2i$. Let a' be an adjacency of a with respect to all input permutations. Let us assume without loss of generality $i = 2$. Since if $i > 2$, then all elements between 1 and $2i-2$ are already before the position $2i-1$. Hence, we consider a solution permutation $[1\,2\,3\,a\,\ldots\,4\,\ldots]$, such that it can be either: (i) $[1\,2\,3\,a\,\ldots\,4\,\ldots\,a'\ldots]$, or (ii) $[1\,2\,3\,a\,\ldots\,a'\,\ldots\,4\ldots]$. Therefore, we compare this solution to any kind of input permutation.

If the solution is (i), we obtain $[1\,2\,3\,\underline{a\,\ldots}\,\underline{4\,\ldots\,a'}\ldots] \rightarrow [1\,2\,3\,4\,\ldots\,a'\,a\,\ldots]$. For each possible input permutation, we justify below that the distance between the new permutation and each input does not increase.

- Case 1. If an input has the adjacency a, a', then:

  Subcase 1.1: $[1\,2\,3\,4\,\ldots\,a\,a'\,\ldots]$, we are creating one cycle of length 1, by creating the adjacency 3, 4. Hence, such transposition is at least a 0-move;

  Subcase 1.2: $[1\,2\,4\,3\,\ldots\,a\,a'\,\ldots]$, the elements a and a' are in the same cycle. Hence, such transposition is at least a 0-move;

  Subcase 1.3: $[2\,1\,3\,4\,\ldots\,a\,a'\,\ldots]$, we are creating one cycle of length 1, by creating the adjacency a, a'. Hence, such transposition is at least a 0-move, similar the Subcase 1.1;

  Subcase 1.4: $[2\,1\,4\,3\,\ldots\,a\,a'\,\ldots]$, the elements 0, 1, 2, 3, 4 are in the same cycle. Hence, such transposition is at least a 0-move, since it affects the elements 3, 4.

- Case 2. If an input has the adjacency a', a, then:

  Subcase 2.1: $[1\,2\,3\,4\,\ldots\,a'\,a\,\ldots]$, we are creating two cycles of length 1, by creating the adjacencies 3, 4 and a', a;

  Subcase 2.2: $[1\,2\,4\,3\,\ldots\,a'\,a\,\ldots]$, we are creating one cycle of length 1, by creating the adjacency a', a. Hence, such transposition is at least a 0-move;

  Subcase 2.3: $[2\,1\,3\,4\,\ldots\,a'\,a\,\ldots]$, we are creating one cycle of length 1, by creating the adjacency 3, 4. Hence, such transposition is at least a 0-move, similar the Subcases 1.1 and 1.3;

  Subcase 2.4: $[2\,1\,4\,3\,\ldots\,a'\,a\,\ldots]$, we are creating one cycle of length 1, by creating the adjacency a', a. Hence, such transposition is at least a 0-move.

If the solution is (ii), we obtain $[1\,2\,3\,\underline{a\,\ldots 4\,\ldots\,b}\,c\ldots] \rightarrow [1\,2\,3\,4\,\ldots\,b\,a\,\ldots]$, for the pair b, c being a universal breakpoint, i.e. a breakpoint with respect to all input permutations. Note the existence of such breakpoint, since the element a is before the element 4 in the permutation and there is some place at the right of 4 where a should be. For each possible input permutation we justify below that the distance between the new permutation and each input does not increase. In Subcases 1.1, 2.1, 1.3, 2.3 we are creating one cycle of length 1, by the adjacency 3, 4. Hence, such transposition is at least a 0-move; The remaining cases the transposition applied is between elements of same cycle.

2. in position $2i-1$ there is not the element $2i-1$ nor the element $2i$, either.

In this case we assume the solution is $[1\,2\,a\,\ldots\,3\,\ldots 4\ldots]$. We consider the inputs with the adjacency 3, 4 or the adjacency 4, 3, so we do not need to deal with the case of the solution $[1\,2\,a\,\ldots\,4\,\ldots 3\ldots]$.

In all cases $[1\,2\,4\,3\,\ldots]$, $[1\,2\,3\,4\,\ldots]$, $[2\,1\,4\,3\,\ldots]$, and $[2\,1\,3\,4\,\ldots]$, the elements 2, 4 are in the same cycle. Hence, any transposition affecting such elements that fix element 4 after 2 is at least a 0-move.

($\Leftarrow$) Given a solution string s, we obtain the associated permutation λ_s given by Algorithm 3. By Lemma 3.6 we have the solution s regarding the Hamming-CSP corresponding to the permutation λ_s with the same value of max distance d. □

Lemma 3.7 implies the following result.

Theorem 3.3. *The* Block-interchange-CGP *is* NP*-complete.*

4. Further work

This paper describes the complexity of the Closest Genome Problem with respect to well-known metrics. Table 1 summarizes the state of the art of the computational complexity of the distance, closest and median problems with respect to seven well-known metrics. We find in the second column the complexity status for the closest problem with respect to the metrics studied in this paper (single-cut-or-join, breakpoint and block-interchange), for the Cayley, reversal, transposition, and for the *double-cut-and-join* (DCJ), which is another well-known metric studied in the context of comparative genomics [10] closely related to the single-cut-or-join, but so far not considered with respect to the closest problem.

Despite the hardness to decide the CGP with respect to the metrics shown in the present paper, some interesting questions arise.

How to improve the 2*-approximation algorithm that computes all pairwise distances?* By the triangular inequality, a necessary condition for a string to be the center with radius at most d is that the distance of any pair in the input string set must be at most $2d$. Hence, if such condition is true for a given input string set, then any string of the input is a solution with approximation ratio 2 of an optimal solution. Since the two considered metrics admit polynomial algorithms to compute the distances, one question would be to lower the approximation ratio.

Table 1
Computational complexity of the distance, closest and median problems. Highlighted cells are the contributions in the present paper. We conjecture that Closest Problem is NP-complete for DCJ and Reversal (signed) distances.

	Distance	Closest	Median
DCJ	Polynomial	Open	NP-complete
SCJ	Polynomial	NP-complete	Polynomial
Breakpoint	Polynomial	NP-complete	Polynomial
Block-interchange	Polynomial	NP-complete	Open
Cayley	Polynomial	NP-complete	Open
Reversal (signed)	Polynomial	Open	NP-complete
Transposition	NP-complete	NP-complete	NP-complete

What can be further said about the KENDALL-τ MEDIAN PROBLEM*?* A related problem is the MEDIAN PROBLEM, where we ask for the solution string/permutation that minimizes the sum of the distances between the solution and the input string/permutation set. The HAMMING MEDIAN STRING PROBLEM is a polynomial problem [11], but regarding permutations the breakpoint [15], transposition [1], and reversal [5] MEDIAN PERMUTATION PROBLEMS are NP-complete. For the DCJ, Tannier et al. [19] proved that DCJ Median problem is NP-complete, and Feijão and Meidanis [8] proved that the SCJ Median problem is polynomial, as described in Table 1.

The Kendall-τ operation, also known as the *bubble sort*, is an exchange between two consecutive elements in a permutation. Hence, the Block-interchange, as well as the Transposition and the Cayley, are all generalizations of the Kendall-τ operation. The KENDALL-τ MEDIAN PROBLEM is known to be NP-complete, but its complexity is open when the input set has three permutations [3].

Note that the relationship between the closest and the median problems often appears in classical combinatorial optimization problems. The closest problem is a min–max problem and the median problem is a min-sum problem. For instance, in graph theory, given two sets of vertices, there are the min–max disjoint path and the min-sum disjoint path problems. These path problems were considered in several papers, where they are polynomial or NP-complete, according to distinct classes of graphs [13,18]. Hence, it is interesting to investigate and contrast the computational complexity of the closest and the median problems for several metrics of distances.

Acknowledgments

The authors thank the anonymous referees for the diligent reading and suggestions. This study was financed in part by the *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior — Brasil (CAPES)* — Finance Code 001, by the *Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro — Brasil (FAPERJ)*, and by *Conselho Nacional de Desenvolvimento Científico e Tecnológico — Brasil (CNPq)*.

References

[1] M. Bader, The transposition median problem is NP-complete, Theoret. Comput. Sci. 412 (2011) 1099–1110.
[2] V. Bafna, P.A. Pevzner, Sorting by transpositions, SIAM J. Discrete Math. 11 (1998) 224–240.
[3] G. Blin, M. Crochemore, S. Hamel, S. Vialette, Medians of an odd number of permutations, Pure Math. Appl. 21 (2) (2010) 161–175.
[4] L. Bulteau, G. Fertin, I. Rusu, Sorting by transpositions is difficult, SIAM J. Discrete Math. 26 (3) (2012) 1148–1180.
[5] A. Caprara, The reversal median problem, INFORMS J. Comput. 15 (2003) 93–113.
[6] D.A. Christie, Sorting permutations by block-interchange, Inform. Process. Lett. 60 (1996) 165–169.
[7] L.F.I. Cunha, L.A.B. Kowada, R.A. Hausen, C.M.H. Figueiredo, Advancing the transposition distance and the diameter through lonely permutations, SIAM. J. Discrete Math. 27 (4) (2013) 1682–1709.
[8] P. Feijão, J. Meidanis, SCJ: a breakpoint-like distance that simplifies several rearrangement problems, IEEE/ACM Trans. Comput. Biol. Bioinf. 8 (1318) (2011).
[9] G. Fertin, A. Labarre, I. Rusu, E. Tannier, S. Vialette, Combinatorics of Genome Rearrangements, The MIT Press, 2009.
[10] R. Friedberg, A.E. Darling, S. Yancopoulos, Genome rearrangement by the double cut and join operation, Methods Mol. Bio. 285 (452) (2008) 385.
[11] J. Gramm, R. Niedermeier, P. Rossmanith, Fixed-parameter algorithms for CLOSEST STRING and related problems, Algorithmica 37 (2003) 25–42.
[12] M. Haghighi, D. Sankoff, Medians seek the corners, and other conjectures, BMC Bioinformatics 13 (Suppl 19) (2012) S5.
[13] Y. Kobayashi, C. Sommer, On shortest disjoint paths in planar graphs, Discrete Optim. 7 (2010) 234–245.
[14] J.K. Lanctot, M. Li, B. Ma, S. Wang, L. Zhang, Distinguishing string selection problems, Inf. Comput. 185 (1) (2003) 41–55.
[15] I. Pe'er, R. Shamir, The Median Problems for Breakpoints are NP-Complete, Technical report TR98-071, The Electronic Colloquium on Computational Complexity, 1998.
[16] P.A. Pevzner, Computational Molecular Biology: An Algorithmic Approach, The MIT Press, 2000.
[17] V.Y. Popov, Multiple genome rearrangement by swaps and by element duplications, Theoret. Comput. Sci. 385 (2007) 115–126.
[18] N. Robertson, P.D. Seymour, An outline of a disjoint paths algorithm, in: Paths, Flows, and VLSI-Layout, Springer-Verlag, 1990, pp. 267–292.
[19] E. Tannier, C. Zheng, D. Sankoff, Multichromosomal median and halving problems under different genomic distances, BMC Bioinformatics 10 (2009) 120.