Algorithms for Computing the Family-Free Genomic Similarity under DCJ

Diego P. Rubert[1], Gabriel L. Medeiros[1], Edna A. Hoshino[1], Marília D. V. Braga[2], Jens Stoye[2], and Fábio V. Martinez[1,⋆]

[1] Faculdade de Computação, Universidade Federal de Mato Grosso do Sul, Campo Grande, MS, Brazil, (diego,gabriel_medeiros,eah,fhvm)@facom.ufms.br,
[2] Faculty of Technology and Center for Biotechnology (CeBiTec), Bielefeld University, Bielefeld, Germany, (mbraga,stoye)@cebitec.uni-bielefeld.de

Abstract. The genomic similarity is a large-scale measure for comparing two given genomes. In this work we study the (NP-hard) problem of computing the genomic similarity under the DCJ model in a setting that does not assume that the genes of the compared genomes are grouped into gene families. This problem is called family-free DCJ similarity. Here we propose an exact ILP algorithm to solve it, we show its APX-hardness, and we present three combinatorial heuristics, with computational experiments comparing their results to the ILP. Experiments on simulated datasets show that the proposed heuristics are very fast and even competitive with respect to the ILP algorithm for some instances.

Keywords: Genome rearrangement, Double-cut-and-join, Family-free genomic similarity

1 Introduction

A central question in comparative genomics is the elucidation of similarities and differences between genomes. Local and global measures can be employed. A popular set of global measures is based on the number of genome rearrangements necessary to transform one genome into another one [23]. Genome rearrangements are large scale mutations, changing the number of chromosomes and/or the positions and orientations of DNA segments. Examples of such rearrangements are inversions, translocations, fusions, and fissions.

As a first step before such a comparison can be performed, some preprocessing is required. The most common method, adopted for about 20 years [23, 24], is to base the analysis on the order of conserved DNA syntenic segments across different genomes and group homologous segments into *families*. This setting is said to be *family-based*. Without duplicate segments, i.e., with the additional restriction that at most one representative of each family occurs in any genome, several polynomial time algorithms have been proposed to compute genomic distances and similarities [5, 6, 9, 16, 27]. However, when duplicates are allowed,

⋆ Corresponding author.

problems become more intricate and many presented approaches are NP-hard [1–3, 11, 12, 24, 26].

Although family information can be obtained by accessing public databases or by direct computing, data can be incorrect, and inaccurate families could be providing support to erroneous assumptions of homology between segments [15]. Thus, it is not always possible to classify each segment unambiguously into a single family, and an alternative to the family-based setting was proposed recently [10]. It consists of studying genome rearrangements without prior family assignment, by directly accessing the pairwise similarities between DNA segments of the compared genomes. This approach is said to be *family-free* (FF).

The *double cut and join* (DCJ) operation, that consists of cutting a genome in two distinct positions and joining the four resultant open ends in a different way, represents most of large-scale rearrangements that modify genomes [27]. In this work we are interested in the problem of computing the overall similarity of two given genomes in a family-free setting under the DCJ model. This problem is called FFDCJ similarity. The complexity of computing the FFDCJ similarity was proven to be NP-hard [20], while the counterpart problem of computing the FFDCJ distance was already proven to be APX-hard [20]. In the remainder of this paper, after preliminaries and a formal definition of the NP-hard FFDCJ similarity problem, we first present an exact ILP algorithm to solve it. We then show the APX-hardness of the FFDCJ similarity problem and present three combinatorial heuristics, with computational experiments comparing their results to the ILP for datasets simulated by a framework for genome evolution.

2 Preliminaries

Each segment (often called *gene*) g of a genome is an oriented DNA fragment and its two distinct *extremities* are called *tail* and *head*, denoted by g^t and g^h, respectively. A genome is composed of a set of chromosomes, each of which can be circular or linear and is a sequence of genes. Each one of the two extremities of a linear chromosome is called a *telomere*, represented by the symbol $\circ$. An *adjacency* in a chromosome is then either the extremity of a gene that is adjacent to a telomere, or a pair of consecutive gene extremities. As an example, observe that the adjacencies 5^h, 5^t2^t, 2^h4^t, 4^h3^t, 3^h6^t, 6^h1^h and 1^t can define a linear chromosome. Another representation of the same linear chromosome, flanked by parentheses for the sake of clarity, would be $(\circ\ -5\ 2\ 4\ 3\ 6\ -1\ \circ)$, in which the genes preceded by the minus sign $(-)$ have reverse orientation.

A *double cut and join* or DCJ operation applied to a genome A is the operation that cuts two adjacencies of A and joins the separated extremities in a different way, creating two new adjacencies. For example, a DCJ acting on two adjacencies pq and rs would create either the adjacencies pr and qs, or the adjacencies ps and qr (this could correspond to an inversion, a reciprocal translocation between two linear chromosomes, a fusion of two circular chromosomes, or an excision of a circular chromosome). In the same way, a DCJ acting on two adjacencies pq and r would create either pr and q, or p and qr (in this case,

the operation could correspond to an inversion, a translocation, or a fusion of a circular and a linear chromosome). For the cases described so far we can notice that for each pair of cuts there are two possibilities of joining. There are two special cases of a DCJ operation, in which there is only one possibility of joining. The first is a DCJ acting on two adjacencies p and q, that would create only one new adjacency pq (that could represent a circularization of one or a fusion of two linear chromosomes). Conversely, a DCJ can act on only one adjacency pq and create the two adjacencies p and q (representing a linearization of a circular or a fission of a linear chromosome).

In the remainder of this section we extend the notation introduced in [20]. In general we consider the comparison of two distinct genomes, that will be denoted by A and B. Respectively, we denote by $\mathcal{A}$ the set of genes in genome A, and by $\mathcal{B}$ the set of genes in genome B.

2.1 Adjacency Graph and Family-Based DCJ Similarity

In most versions of the family-based setting the two genomes A and B have the same content, that is, $\mathcal{A} = \mathcal{B}$. When in addition there are no duplicates, that is, when there is exactly one representative of each family in each genome, we can easily build the *adjacency graph* of genomes A and B, denoted by $AG(A, B)$ [6]. It is a bipartite multigraph such that each partition corresponds to the set of adjacencies of one of the two input genomes, and an edge connects the same extremities of genes in both genomes. In other words, there is a one-to-one correspondence between the set of edges in $AG(A, B)$ and the set of gene extremities. Since the graph is bipartite and vertices have degree one or two, the adjacency graph is a collection of paths and even cycles. An example of an adjacency graph is presented in Fig. 1.

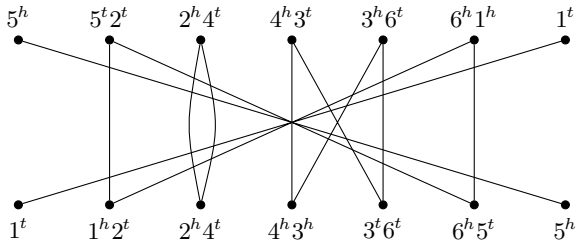

Fig. 1. The adjacency graph for the genomes $A = \{(\circ\ -5\ 2\ 4\ 3\ 6\ -1\ \circ)\}$ and $B = \{(\circ\ 1\ 2\ 4\ -3\ 6\ 5\ \circ)\}$.

It is well known that a DCJ operation that modifies $AG(A, B)$ by increasing either the number of even cycles by one or the number of odd paths by two decreases the DCJ distance between genomes A and B [6]. This type of DCJ

operation is said to be *optimal*. Conversely, if we are interested in a DCJ similarity measure between A and B, rather than a distance measure, then it should be increased by such an optimal DCJ operation. This suggests that a formula for a DCJ similarity between two genomes should correlate to the number of connected components (in the following just *components*) of the corresponding adjacency graph.

Moreover, when the genomes A and B are identical, their corresponding adjacency graph is a collection of 2-cycles and 1-paths [6]. This should correspond to the maximum value of our DCJ similarity measure. We know that an optimal operation can always be applied to adjacencies that belong to one of the two genomes and to one single component of $AG(A, B)$, until the graph becomes a collection of 2-cycles and 1-paths. In other words, each component of the graph can be *sorted*, that is, converted into a collection of 2-cycles and 1-paths independently of the other components. Furthermore, it is known that each of the following components – an even cycle with $2d + 2$ edges, or an odd path with $2d + 1$ edges, or an even path with $2d$ edges – can be sorted with exactly d optimal DCJ operations. This suggests that the three listed components should have equivalent weights in the DCJ similarity formula. However, we should also take into consideration that, for the same d, components with more edges should actually have a higher weight.

Let $\mathcal{P}$, $\mathcal{I}$, and $\mathcal{C}$ represent the sets of components in $AG(A, B)$ that are even paths, odd paths and cycles, respectively. We have the following formula for the family-based DCJ similarity:

$$\begin{aligned} \mathrm{s}_{\mathrm{DCJ}}(A, B) &= \sum_{C \in \mathcal{P}} \left(\frac{|C|}{|C|+2} \right) + \sum_{C \in \mathcal{I}} \left(\frac{|C|}{|C|+1} \right) + \sum_{C \in \mathcal{C}} \left(\frac{|C|}{|C|} \right) \\ &= \sum_{C \in \mathcal{P}} \left(\frac{|C|}{|C|+2} \right) + \sum_{C \in \mathcal{I}} \left(\frac{|C|}{|C|+1} \right) + c, \end{aligned} \tag{1}$$

where c is the number of cycles in $AG(A, B)$. In Fig. 1 the DCJ similarity is $\mathrm{s}_{\mathrm{DCJ}}(A, B) = 2\left(\frac{1}{2}\right) + 3 = 4$. Observe that $\mathrm{s}_{\mathrm{DCJ}}(A, B)$ is a positive value, upper bounded by n, where $n = |\mathcal{A}| = |\mathcal{B}|$.

The formula to compute $\mathrm{s}_{\mathrm{DCJ}}(A, B)$ in Equation (1) is actually the family-based version of the family-free DCJ similarity defined in [20], as we will see in the following subsections.

2.2 Gene Similarity Graph

In the family-free setting, each gene in each genome is represented by a unique (signed) symbol, thus $\mathcal{A} \cap \mathcal{B} = \emptyset$ and the cardinalities $|\mathcal{A}|$ and $|\mathcal{B}|$ may be distinct. Let a be a gene in A and b be a gene in B, then their *normalized gene similarity* is given by some value $\sigma(a, b)$ such that $0 \leq \sigma(a, b) \leq 1$.

We can represent the gene similarities between the genes of genome A and the genes of genome B with respect to σ in the so called *gene similarity graph* [10],

denoted by $GS_\sigma(A, B)$. This is a weighted bipartite graph whose partitions $\mathcal{A}$ and $\mathcal{B}$ are the sets of (signed) genes in genomes A and B, respectively. Furthermore, for each pair of genes (a, b) such that $a \in \mathcal{A}$ and $b \in \mathcal{B}$, if $\sigma(a, b) > 0$ then there is an edge e connecting a and b in $GS_\sigma(A, B)$ whose weight is $\sigma(e) := \sigma(a, b)$. An example of a gene similarity graph is given in Fig. 2.

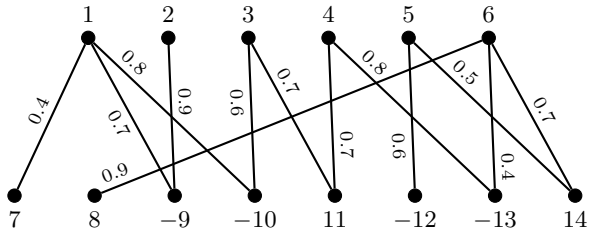

Fig. 2. Representation of a gene similarity graph for two unichromosomal linear genomes $A = \{(\circ\ 1\ 2\ 3\ 4\ 5\ 6\ \circ)\}$ and $B = \{(\circ\ 7\ 8\ -9\ -10\ 11\ -12\ -13\ 14\ \circ)\}$.

2.3 Weighted Adjacency Graph

The *weighted adjacency graph* $AG_\sigma(A, B)$ of two genomes A and B has a vertex for each adjacency in A and a vertex for each adjacency in B. For a gene a in A and a gene b in B with gene similarity $\sigma(a, b) > 0$ there is one edge e^h connecting the vertices containing the two heads a^h and b^h and one edge e^t connecting the vertices containing the two tails a^t and b^t. The weight of each of these edges is $\sigma(e^h) = \sigma(e^t) = \sigma(a, b)$. Differently from the simple adjacency graph, the weighted adjacency graph cannot be easily decomposed into cycles and paths, since its vertices can have degree greater than 2. As an example, the weighted adjacency graph corresponding to the gene similarity graph of Fig. 2 is given in Fig. 3.

2.4 Reduced Genomes

Let A and B be two genomes and let $GS_\sigma(A, B)$ be their gene similarity graph. Now let $M = \{e_1, e_2, \ldots, e_n\}$ be a matching in $GS_\sigma(A, B)$ and denote by $w(M) = \sum_{e_i \in M} \sigma(e_i)$ the weight of M, that is the sum of its edge weights. Since the endpoints of each edge $e_i = (a, b)$ in M are not saturated by any other edge of M, we can unambiguously define the function $\ell^M(a) = \ell^M(b) = i$ to relabel each vertex in A and B [20]. The *reduced genome* A^M is obtained by deleting from A all genes not saturated by M, and renaming each saturated gene a to $\ell^M(a)$, preserving its orientation (sign). Similarly, the reduced genome B^M is obtained by deleting from B all genes that are not saturated by M, and renaming each saturated gene b to $\ell^M(b)$, preserving its orientation. Observe that the set of genes in A^M and in B^M is $\mathcal{G}(M) = \{\ell^M(g) : g \text{ is saturated by the matching } M\} = \{1, 2, \ldots, n\}$.

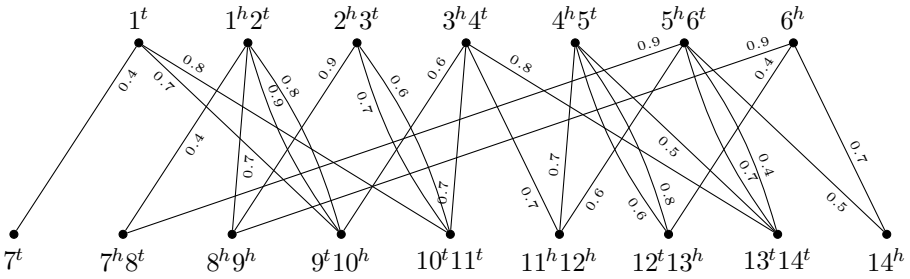

Fig. 3. The weighted adjacency graph $AG_\sigma(A, B)$ for two unichromosomal linear genomes $A = \{(\circ\ 1\ 2\ 3\ 4\ 5\ 6\ \circ)\}$ and $B = \{(\circ\ 7\ 8\ {-9}\ {-10}\ 11\ {-12}\ {-13}\ 14\ \circ)\}$.

2.5 Weighted Adjacency Graph of Reduced Genomes

Let A^M and B^M be the reduced genomes for a given matching M of $GS_\sigma(A, B)$. The weighted adjacency graph $AG_\sigma(A^M, B^M)$ can be obtained from $AG_\sigma(A, B)$ by deleting all edges that are not elements of M and relabeling the adjacencies according to ℓ^M. Vertices that have no connections are then also deleted from the graph. Another way to obtain the same graph is building the adjacency graph of A^M and B^M and adding weights to the edges as follows. For each gene i in $\mathcal{G}(M)$, both edges $i^t i^t$ and $i^h i^h$ inherit the weight of edge e_i in M, that is, $\sigma(i^t i^t) = \sigma(i^h i^h) = \sigma(e_i)$. Consequently, the graph $AG_\sigma(A^M, B^M)$ is also a collection of paths and even cycles and differs from $AG(A^M, B^M)$ only by the edge weights.

Observe that, for each edge $e \in M$, we have two edges of weight $\sigma(e)$ in $AG_\sigma(A^M, B^M)$, thus $w(AG_\sigma(A^M, B^M)) = 2\, w(M)$. Examples of weighted adjacency graphs of reduced genomes are shown in Fig. 4.

2.6 The Family-Free DCJ Similarity

For a given matching M in $GS_\sigma(A, B)$, a first formula for the weighted DCJ (wDCJ) similarity s_σ of the reduced genomes A^M and B^M was proposed in [10] only considering the cycles of $AG_\sigma(A^M, B^M)$. Following, this definition was modified and extended in [20], in order to consider the normalized total weight of all components of the weighted adjacency graph. Let $\mathcal{P}$, $\mathcal{I}$, and $\mathcal{C}$ represent the sets of components in $AG_\sigma(A^M, B^M)$ that are even paths, odd paths and cycles, respectively, and $w(C) = \sum_{e \in C} \sigma(e)$ is the sum of the weights of all the edges in a component C. Then the wDCJ similarity s_σ is given by the following formula [20]:

$$s_\sigma(A^M, B^M) = \sum_{C \in \mathcal{P}} \left(\frac{w(C)}{|C|+2} \right) + \sum_{C \in \mathcal{I}} \left(\frac{w(C)}{|C|+1} \right) + \sum_{C \in \mathcal{C}} \left(\frac{w(C)}{|C|} \right) \tag{2}$$

Observe that, when the weights of all edges in M are equal to 1, this formula is equivalent to the one in Equation (1).

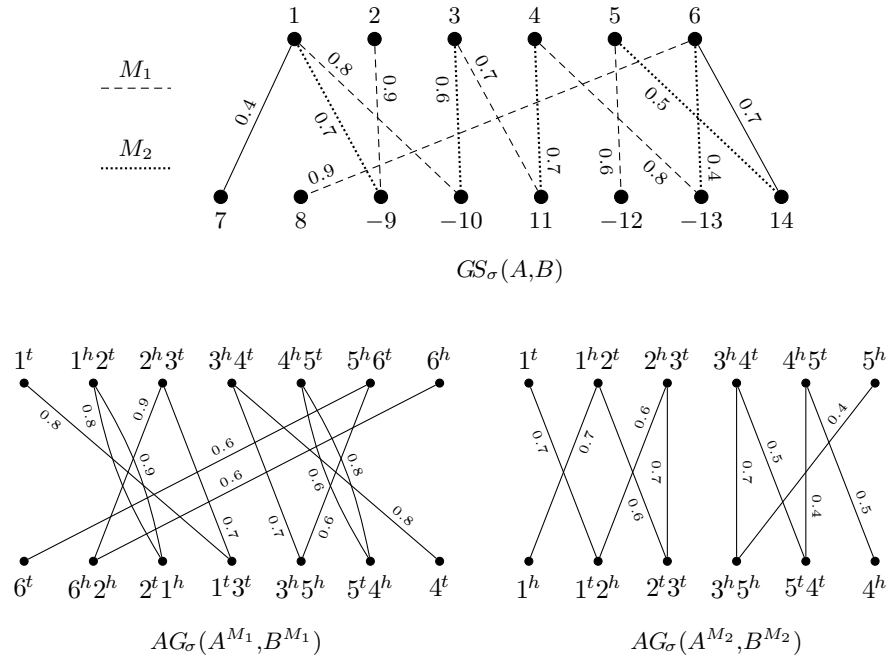

Fig. 4. Considering, as in Fig. 2, the genomes $A = \{(\circ\ 1\ 2\ 3\ 4\ 5\ 6\ \circ)\}$ and $B = \{(\circ\ 7\ 8\ -9\ -10\ 11\ -12\ -13\ 14\ \circ)\}$, let M_1 (dashed edges) and M_2 (dotted edges) be two distinct maximal matchings in $GS_\sigma(A, B)$, shown in the upper part. The two resulting weighted adjacency graphs $AG_\sigma(A^{M_1}, B^{M_1})$, that has two cycles and two even paths, and $AG_\sigma(A^{M_2}, B^{M_2})$, that has two odd paths, are shown in the lower part.

The goal now is to compute the family-free DCJ similarity, i.e., to find a matching in $GS_\sigma(A, B)$ that maximizes s_σ. However, although $s_\sigma(A^M, B^M)$ is a positive value upper bounded by $|M|$, the behaviour of the wDCJ similarity does not correlate with the size of the matching, since smaller matchings, that possibly discard gene assignments, can lead to higher wDCJ similarities [20]. For this reason, the wDCJ similarity function is restricted to *maximal matchings* only, ensuring that no pair of genes with positive gene similarity score is simply discarded, even though it might decrease the overall wDCJ similarity. We then have the following optimization problem:

Problem FFDCJ-SIMILARITY(A, B): Given genomes A and B and their gene similarities σ, calculate their family-free DCJ similarity

$$\mathrm{s}_{\mathrm{FFDCJ}}(A, B) = \max_{M \in \mathbb{M}} \{s_\sigma(A^M, B^M)\}, \tag{3}$$

where $\mathbb{M}$ is the set of all maximal matchings in $GS_\sigma(A, B)$.

Problem FFDCJ-SIMILARITY is NP-hard [20]. One can directly correlate this problem to the adjacency similarity problem, where the goal is to maximize the number of preserved adjacencies between two given genomes [1]. However, since there the objective is to maximize the number of cycles of length 2, even an approximation for the adjacency similarity problem is not a good algorithm for the FFDCJ-SIMILARITY problem, where cycles of higher lengths are possible in the solution [22].

2.7 Capping Telomeres

A very useful preprocessing to $AG_\sigma(A, B)$ is the *capping* of telomeres, a general technique for simplifying algorithms that handle genomes with linear chromosomes, commonly used in the context of family-based settings [16, 25, 27]. Given two genomes A and B with i and j linear chromosomes, respectively, for each vertex representing only one extremity we add a *null extremity* τ to it (e.g., 1^t of Fig. 4 becomes $\tau 1^t$). Furthermore, in order to add the same number of null extremities to both genomes, $|2j - 2i|$ *null adjacencies* $\tau\tau$ (composed of two null extremities) are added to genome A, if $i < j$, or to genome B, if $j < i$. Finally, for each null extremity of a vertex in A we add to $AG_\sigma(A, B)$ a *null edge* with weight 0 to each null extremity of vertices in B. Consequently, after capping of telomeres the graph $AG_\sigma(A, B)$ has no vertex of degree one. Notice that, if before the capping p was a path of weight w connecting telomeres in $AG_\sigma(A, B)$, then after the capping p will be part of a cycle closed by null extremities with normalized weight $\frac{w}{|p|+1}$ if p is an odd path, or of normalized weight $\frac{w}{|p|+2}$ if p is an even path. In any of the two cases, the normalized weight is consistent with the wDCJ similarity formula in Equation (2).

3 An Exact Algorithm

In order to exactly compute the family-free DCJ similarity between two given genomes, we propose an integer linear program (ILP) formulation that is similar to the one for the family-free DCJ distance given in [20]. It adopts the same notation and also uses an approach to solve the maximum cycle decomposition problem as in [26].

Let A and B be two genomes, let $G = GS_\sigma(A, B)$ be their gene similarity graph, and let X_A and X_B be the extremity sets (including null extremities) with respect to A and B for the capped adjacency graph $AG_\sigma(A, B)$, respectively. The weight $w(e)$ of an edge e in G is also denoted by w_e. For the ILP formulation, an extension $H = (V_H, E_H)$ of the capped weighted adjacency graph $AG_\sigma(A, B)$ is defined such that $V_H = X_A \cup X_B$ and $E_H = E_m \cup E_a \cup E_s$ has three types of edges: (i) *matching edges* that connect two extremities in different extremity sets, one in X_A and the other in X_B, if they are null extremities or there exists an edge connecting these genes in G; the set of matching edges is denoted by E_m; (ii) *adjacency edges* that connect two extremities in the same extremity set if they form an adjacency; the set of adjacency edges is denoted by E_a; and (iii)

self edges that connect two extremities of the same gene in an extremity set; the set of self edges is denoted by E_s. Matching edges have weights defined by the normalized gene similarity σ, all adjacency and self edges have weight 0. Notice that any edge in G corresponds to two matching edges in H.

The description of the ILP follows. For each edge in H, we create a binary variable x_e to indicate whether e will be in the final solution. We require first that each adjacency edge be chosen:

$$x_e = 1, \qquad \forall\ e \in E_a.$$

Now we rename each vertex in H such that $V_H = \{v_1, v_2, \ldots, v_k\}$ with $k = |V_H|$. We require that each of these vertices be adjacent to exactly one matching or self edge:

$$\sum_{e=v_r v_t \in E_m \cup E_s} x_e = 1, \forall\ v_r \in X_A, \quad \text{and} \quad \sum_{e=v_r v_t \in E_m \cup E_s} x_e = 1, \forall\ v_t \in X_B.$$

Then, we require that the final solution be valid, meaning that if one extremity of a gene in A is assigned to an extremity of a gene in B, then the other extremities of these two genes have to be assigned as well:

$$x_{a^h b^h} = x_{a^t b^t}, \qquad \forall\ ab \in E_G.$$

We also require that the matching be maximal. This can easily be ensured if we guarantee that at least one of the vertices connected by an edge in the gene similarity graph be chosen, which is equivalent to not allowing both of the corresponding self edges in the weighted adjacency graph be chosen:

$$x_{a^h a^t} + x_{b^h b^t} \leq 1, \qquad \forall\ ab \in E_G.$$

To count the number of cycles, we use the same strategy as described in [26]. For each vertex v_i we define a variable y_i that labels v_i such that

$$0 \leq y_i \leq i, \qquad 1 \leq i \leq k.$$

We also require that adjacent vertices have the same label, forcing all vertices in the same cycle to have the same label:

$$y_i \leq y_j + i \cdot (1 - x_e),\ \forall\ e = v_i v_j \in E_H,$$
$$y_j \leq y_i + j \cdot (1 - x_e),\ \forall\ e = v_i v_j \in E_H.$$

We create a binary variable z_i, for each vertex v_i, to verify whether y_i is equal to its upper bound i:

$$i \cdot z_i \leq y_i, \qquad 1 \leq i \leq k.$$

Since all the y_i variables in the same cycle have the same label but a different upper bound, only one of the y_i can be equal to its upper bound i. This means that z_i is 1 if the cycle with vertex i as representative is used in a solution.

Now, let $L = \{2j : j = 1, \ldots, n\}$ be the set of possible cycle lengths in H, where $n := \min(|A|, |B|)$. We create the binary variable x_{ei} to indicate whether e is in i, for each $e \in E_H$ and each cycle i. We also create the binary variable x_{ei}^{ℓ} to indicate whether e belongs to i and the length of cycle i is ℓ, for each $e \in E_H$, each cycle i, and each $\ell \in L$.

We require that if an edge e belongs to a cycle i, then it can be true for only one length $\ell \in L$. Thus,

$$\sum_{\ell \in L} x_{ei}^{\ell} \leq x_{ei}, \qquad \forall\ e \in E_H \text{ and } 1 \leq i \leq k. \tag{4}$$

We create another binary variable z_i^{ℓ} to indicate whether cycle i has length ℓ. Then $\ell \cdot z_i^{\ell}$ is an upper bound for the total number of edges in cycle i of length ℓ:

$$\sum_{e \in E_M} x_{ei}^{\ell} \leq \ell \cdot z_i^{\ell}, \qquad \forall\ \ell \in L \text{ and } 1 \leq i \leq k.$$

The length of a cycle i is given by $\ell \cdot z_i^{\ell}$, for $i = 1, \ldots, k$ and $\ell \in L$. On the other hand, it is the total amount of matching edges e in cycle i. That is,

$$\sum_{\ell \in L} \ell \cdot z_i^{\ell} = \sum_{e \in E_m} x_{ei}, \qquad 1 \leq i \leq k.$$

We have to ensure that each cycle i must have just one length:

$$\sum_{\ell \in L} z_i^{\ell} = z_i, \qquad 1 \leq i \leq k.$$

Now we create the binary variable y_{ri} to indicate whether the vertex v_r is in cycle i. Thus, if $x_{ei} = 1$, i.e., if the edge $e = v_r v_t$ in H is chosen in cycle i, then $y_{ri} = 1 = y_{ti}$ (and $x_e = 1$ as well). Hence,

$$\left.\begin{array}{l} x_{ei} \leq x_e, \\ x_{ei} \leq y_{ri}, \\ x_{ei} \leq y_{ti}, \\ x_{ei} \geq x_e + y_{ri} + y_{ti} - 2, \end{array}\right\} \quad \forall\ e = v_r v_t \in E_H \text{ and } 1 \leq i \leq k. \tag{5}$$

Since y_r is an integer variable, we associate y_r to the corresponding binary variable y_{ri}, for any vertex v_r belonging to cycle i:

$$y_r = \sum_{i=1}^{r} i \cdot y_{ri}, \qquad \forall\ v_r \in V_H.$$

Furthermore, we must ensure that each vertex v_r may belong to at most one cycle:

$$\sum_{i=1}^{r} y_{ri} \leq 1, \qquad \forall\ v_r \in V_H.$$

Finally, we set the objective function as follows:

$$\text{maximize} \quad \sum_{i=1}^{k} \sum_{\ell \in L} \sum_{e \in E_m} \frac{w_e x_{ei}^{\ell}}{\ell}.$$

Note that, with this formulation, we do not have any path as a component. Therefore, the objective function above is exactly the family-free DCJ similarity $s_{\text{FFDCJ}}(A, B)$ as defined in Equations (2) and (3).

Notice that the ILP formulation has $O(N^4)$ variables and $O(N^3)$ constraints, where $N = |A| + |B|$. The number of variables is proportional to the number of variables x_{ei}^{ℓ}, and the number of constraints is upper bounded by (4) and (5).

4 APX-hardness and Heuristics

In this section we first state that problem FFDCJ-SIMILARITY is APX-hard and then provide a lower bound for the approximation ratio.

Theorem 1. FFDCJ-SIMILARITY *is APX-hard and cannot be approximated with approximation ratio better than* $22/21 = 1.0476\ldots$*, unless* $P = NP$.

Proof. See Appendix A.

We now propose three heuristic algorithms to compute the family-free DCJ similarity of two given genomes: a greedy-like heuristic collecting the best density cycles in the weighted adjacency graph, a greedy-like heuristic collecting cycles of increasing lengths in the weighted adjacency graph, and a heuristic that tries to collect sets of cycles of increasing lengths with maximum total density in the weighted adjacency graph by a weighted maximum independent set (WMIS) algorithm.

The algorithms select disjoint cycles in the capped $AG_\sigma(A, B)$, inducing a matching M in $GS_\sigma(A, B)$. In addition to being disjoint, the selected cycles must also be consistent: we say that two edges in $AG_\sigma(A, B)$ are *consistent* if one connects the head and the other connects the tail of the same pair of genes, or if they connect extremities of distinct genes in both genomes. Otherwise they are *inconsistent*. A set of edges, in particular a cycle, is consistent if it has no pair of inconsistent edges. A set of cycles is consistent if the union of all of their edges is consistent.

All three heuristics have a common adjustment step: the deletion of blocking genes, that works as follows. After some iterations of cycle selection that increase the matching M, one or more genes may become blocking and thus must be deleted. This happens when the algorithms either find no cycle, or find some cycles but they are all inconsistent with previous selections, having however genes in $GS_\sigma(A, B)$ unsaturated by M. We call them *blocking genes*. Whenever this occurs, we can find and delete blocking genes by (i) finding genes in $GS_\sigma(A, B)$ having all neighbors saturated by M and thus blocking or (ii) finding a vertex set $S \subseteq \mathcal{A}$ or $S \subseteq \mathcal{B}$ such that, for the set of neighbors $N(S)$ of

vertices in S, we have $|S| > |N(S)|$ (Hall's theorem), then choosing $|S| - |N(S)|$ genes in S as blocking. After deleting a blocking gene g, $AG_\sigma(A, B)$ must be adjusted accordingly, removing edges corresponding to extremities g^t or g^h of g and "merging" the two vertices that represented these extremities. At the end of the three algorithms we have a matching M, and the union of selected cycles is equivalent to $AG_\sigma(A^M, B^M)$.

The three heuristics have an initial step where all cycles of the weighted adjacency graph are generated (see Step 4 of each one). Thus, the running time of the heuristics is potentially exponential in the number of vertices of the weighted adjacency graph. In Sect. 5, these three heuristics will be compared to the exact ILP algorithm from Sect. 3 regarding quality and running time.

4.1 Best Density

The best density heuristic is shown in Algorithm 1 (Greedy-Density). Its first step is to generate all cycles in the weighted adjacency graph based on the gene similarity graph of two given genomes. Cycles are arranged in order decreasing by their densities, i.e., the weight divided by the squared length. Then, consistent cycles are collected following this criterion and the family-free DCJ similarity is computed on these collected cycles and remaining components.

Algorithm 1 Greedy-Density(A, B, σ)

Input: genomes A and B, gene similarity function σ
Output: a family-free DCJ similarity between A and B

1: $M := \emptyset$; $\mathcal{C} := \emptyset$.
2: Build the gene similarity graph $GS_\sigma(A, B)$.
3: Build the capped weighted adjacency graph $AG_\sigma(A, B)$.
4: List all cycles C of $AG_\sigma(A, B)$ in decreasing order of their density $w(C)/|C|^2$.
5: While it is possible, select the best density consistent cycle C that is also consistent with all cycles in $\mathcal{C}$ and add it to $\mathcal{C}$, let $AG_\sigma(A, B) := AG_\sigma(A, B) \setminus C$, update M by adding the new gene connections induced by C.
6: Find and delete blocking genes, returning to Step 4 if there are genes in $GS_\sigma(A, B)$ unsaturated by M.
7: Return $s_\sigma(A, B) = \sum_{C \in \mathcal{C}} \left(\frac{w(C)}{|C|}\right)$

Step 4 is the core of Greedy-Density, where cycles are obtained by a procedure based on Johnson's algorithm [18,19]. Although the number of cycles may be exponential in the size of the input graph, the implemented algorithm restricts the size of cycles found and is good enough for bipartite graphs and for many experiments, as presented in Sect. 5.

4.2 Best Length

The best length heuristic is shown in Algorithm 2 (Greedy-Length). As in the best density heuristic, the first step is to generate all cycles in the weighted

adjacency graph based on the gene similarity graph of the two given genomes. However, cycles are arranged in order increasing by their lengths, where ties are broken by selecting cycles with greater density. Similar to above, consistent cycles are collected following this criterion, and the family-free DCJ similarity is computed on these collected cycles and remaining components.

Algorithm 2 GREEDY-LENGTH(A, B, σ)

Input: genomes A and B, gene similarity function σ
Output: a family-free DCJ similarity between A and B

1: $M := \emptyset$; $\mathcal{C} := \emptyset$.
2: Build the gene similarity graph $GS_\sigma(A, B)$.
3: Build the capped weighted adjacency graph $AG_\sigma(A, B)$.
4: List all cycles C of $AG_\sigma(A, B)$ in increasing order of their lengths $|C|$.
5: Iterate over the list of cycles as follows. Select consistent cycles of length 2, while its is possible. Then, select consistent cycles of length 4. And so on, until there are no more cycles. Let C be the cycle selected at each iteration. Add C to $\mathcal{C}$, let $AG_\sigma(A, B) := AG_\sigma(A, B) \setminus C$, update M by adding the new gene connections induced by C.
6: Find and delete blocking genes, returning to Step 4 if there are genes in $GS_\sigma(A, B)$ unsaturated by M.
7: Return $s_\sigma(A, B) = \sum_{C \in \mathcal{C}} \left(\frac{w(C)}{|C|}\right)$

Again, Step 4 is the core of the algorithm and it is implemented based on Johnson's algorithm [18, 19], but we first find and select cycles of length 2, then of length 4, and so on.

4.3 Best Length with Weighted Maximum Independent Set

The best length heuristic with WMIS is shown in Algorithm 3 (GREEDY-WMIS) and is a variation of GREEDY-LENGTH. Instead of selecting cycles of greater density for a fixed length, this algorithm selects the greatest amount of cycles for a fixed length by a WMIS algorithm. The heuristic builds a *cycle graph* where each vertex is a cycle of $AG_\sigma(A, B)$, the weight of a vertex is the density of the cycle it represents and two vertices are adjacent if the cycles they represent are inconsistent. The heuristic tries to find next an independent set with the greatest weight in the cycle graph. Since this graph is not d-claw-free for any fixed d, the WMIS algorithm [7] does not guarantee any fixed ratio.

5 Experimental Results

Experiments for the ILP and our heuristics were conducted on an Intel i7-4770 3.40GHz machine with 16 GB of memory. In order to do so, we produced simulated datasets by the Artificial Life Simulator (ALF) [14]. Gurobi Optimizer

Algorithm 3 GREEDY-WMIS(A, B, σ)

Input: genomes A and B, gene similarity function σ
Output: a family-free DCJ similarity between A and B

1: $M := \emptyset$; $\mathcal{C} := \emptyset$.
2: Build the gene similarity graph $GS_\sigma(A, B)$.
3: Build the capped weighted adjacency graph $AG_\sigma(A, B)$.
4: List all cycles C of $AG_\sigma(A, B)$ in increasing order of their lengths $|C|$.
5: Iterate over the list of cycles as follows. Select a set of consistent cycles of length 2 trying to maximize the sum of densities by a WMIS algorithm. Then, repeat for consistent cycles of length 4. And so on, until there are no more cycles. Let $\mathcal{C}'$ be the set of cycles selected at each iteration. Add $\mathcal{C}'$ to $\mathcal{C}$, let $AG_\sigma(A, B) := AG_\sigma(A, B) \backslash \mathcal{C}'$, update M by adding the new gene connections induced by $\mathcal{C}'$.
6: Find and delete blocking genes, returning to Step 4 if there are genes in $GS_\sigma(A, B)$ unsaturated by M.
7: Return $s_\sigma(A, B) = \sum_{C \in \mathcal{C}} \left(\frac{w(C)}{|C|}\right)$

7.0 was set to solve ILP instances with default parameters, time limit of 1800 seconds and 4 threads, and the heuristics were implemented in C++.

We generated datasets with 10 genome samples each, running pairwise comparisons between all genomes in the same dataset. Each dataset has genomes of sizes around 25, 50 or 1000 (the latter used only for running the heuristics), generated based on a sample from the tree of life with 10 leaf species and PAM distance of 100 from root to the deepest leaf. Gamma distribution with parameters $k = 3$ and $\theta = 133$ was used for gene length distribution. For amino acid evolution we used the WAG substitution model with default parameters and the preset of Zipfian indels with rate 0.00005. Regarding genome level events, we allowed gene duplications and gene losses with rate 0.002, and reversals and translocations with rate 0.0025, with at most 3 genes involved in each event. To test different proportions of genome level events, we also generated simulated datasets with 2- and 5-fold increase for reversal and translocation rates.

Results are summarized in Table 1. Each dataset is composed of 10 genomes, totaling 45 comparisons of pairs per dataset. Rate $r = 1$ means the default parameter set for genome level events, while $r = 2$ and $r = 5$ mean the 2- and 5-fold increase of rates. For the ILP the table shows the average time for instances for which the optimal solution was found, the number of instances for which the optimizer did not find the optimal solution after time limit and, for the latter class of instances, the average relative gap between the best solution found and the upper bound found by the solver, given by $(\frac{\text{upper bound}}{\text{best solution}} - 1) \times 100$. For heuristics, the running time for all instances of sizes 25 and 50 was negligible, therefore the table shows only the average relative gap between the solution found and the upper bound given by the ILP solver (if any).

Results clearly show the average relative gap of heuristics increases proportionally to the rate of reversals and translocations. This is expected, as higher mutation rates often result in higher normalized weights on longer cycles, thus the association of genes with greater gene similarity scores will be subject to the

Table 1. Results of experiments for simulated genomes

	ILP			Greedy-Density	Greedy-Length	Greedy-wmis
	Time (s)	Not finished	Gap (%)	Gap (%)	Gap (%)	Gap (%)
25 genes, $r = 1$	19.50	0	–	5.03	5.84	5.97
25 genes, $r = 2$	84.60	2	69.21	30.77	43.57	43.00
25 genes, $r = 5$	49.72	0	–	43.83	55.38	55.38
50 genes, $r = 1$	445.91	7	19.56	18.74	19.36	18.90
50 genes, $r = 2$	463.50	29	38.12	65.41	66.52	64.78
50 genes, $r = 5$	330.88	29	259.72	177.58	206.60	206.31

selection of longer cycles. Interestingly, for some larger instances the relative gap for heuristics is very close to the values obtained by the ILP solver, suggesting the use of heuristics may be a good alternative for some classes of instances or could help the solver finding lower bounds quickly. It is worth noting that the Greedy-Density heuristic found solutions with gap smaller than 1% for 38% of the instances with 25 genes.

In a single instance (25 genes, $r = 2$), the gap between the best solution found and the upper bound was much higher for the ILP solver and for the heuristics. This instance in particular is precisely the one with the largest number of edges in $GS_\sigma(A, B)$ in the dataset. This may indicate that a moderate increase in degree of vertices (1.3 on average to 1.8 in this case) may result in much harder instances for the solver and heuristics, as after half of the time limit the solver attained no significant improvement on solutions found, and the heuristics returned solutions with a gap even higher.

Although we have no upper bounds for comparing the results of our heuristics for genome sizes around 1000, the algorithms are still very fast. The average running times are 0.30, 15.11 and 12.16 seconds for Greedy-Density, Greedy-Length and Greedy-wmis, respectively, showing nevertheless little difference on results. However, in 25% of the instances with $r = 5$, the solutions provided by the heuristics varied between 10% and 24%, the best of which were given by Greedy-Density. That is probably because, instead of prioritizing shorter cycles, Greedy-Density attempts to balance both normalized weight and length of the selected cycles. The average running times for the instances with $r = 5$ are 2.35, 97.28 and 102.67 seconds for Greedy-Density, Greedy-Length and Greedy-wmis, respectively.

To better understand how cycles scale, we generated 5-fold instances with 100, 500, 1000, 5000, and 10000 genes, running the Greedy-Density heuristic for these instances and counting different cycle lengths. The running time was 0.008, 0.667, 1.98, 508 and 2896 seconds, respectively, on average. Results (Fig. 5) show that most of the cycles found are of short lengths compared to the genome sizes, providing some insight on why heuristics are fast despite having to enumerate a number of cycles that could be exponential. Besides, even the maximum number of longer cycles found for any instance is reasonably small.

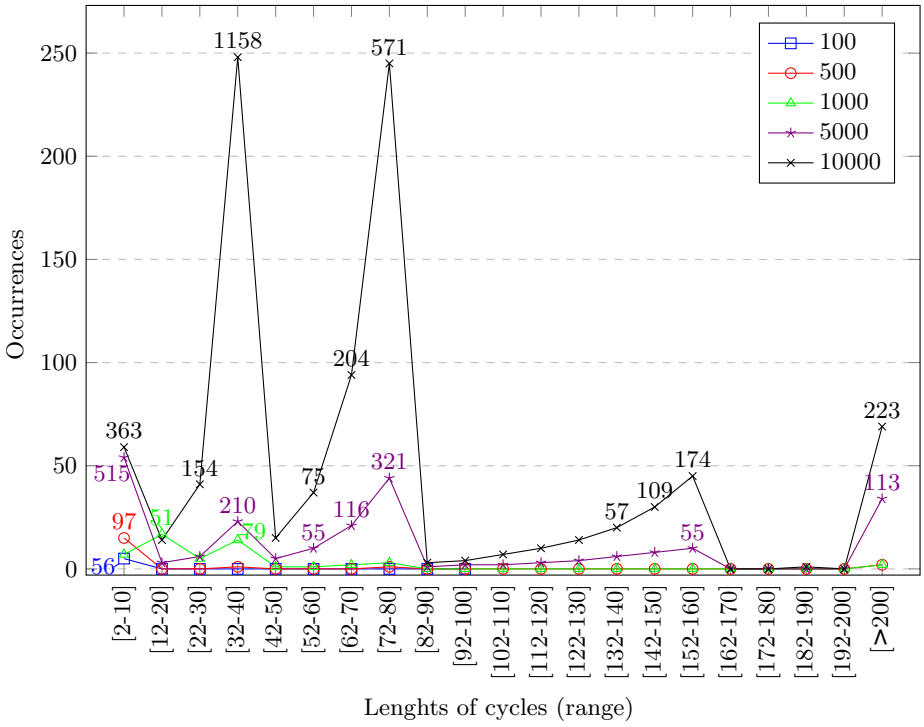

Fig. 5. Average count by lengths of cycles for the Greedy-Density heuristic for instances with $r = 5$ and genome sizes of 100, 500, 1000, 5000, and 10000 genes. Numbers above marks denote the maximum number of cycles for a pair of genomes in an instance (only for values greater than 50). As seen, the number of cycles may be exponential, therefore the heuristic implementation finds cycles of lengths up to 10, then up to 20, and so on. Moreover, when finding cycles of lengths up to 20, the algorithm does not try to find cycles composed by adjacencies in $AG_\sigma(A, B)$ already covered by shorter cycles chosen previously. The same occurs for longer lengths.

6 Conclusion

In this paper we studied the family-free DCJ similarity, which is a large-scale rearrangement measure for comparing two given genomes. We first presented formally the (NP-hard) problem of computing the family-free DCJ similarity. Then, we proposed an exact ILP algorithm to solve it. Following, we showed the APX-hardness of the family-free DCJ similarity problem and presented three combinatorial heuristics, with computational experiments comparing their results to the ILP. Results show that while the ILP program is fast and accurate for smaller instances, the Greedy-Density heuristic is probably the best choice for general use on larger instances.

One drawback of the function s_{FFDCJ} as defined in Equation (3) is that distinct pairs of genomes might give family-free DCJ similarity values that cannot be

compared easily, because the value of s_{FFDCJ} varies between 0 and $|M|$, where M is the matching giving rise to s_{FFDCJ}. Therefore some kind of normalization would be desirable. A simple approach could be to divide s_{FFDCJ} by the size of the smaller genome, because this is a trivial upper bound for $|M|$. Moreover, it can be applied as a simple postprocessing step, keeping all theoretical results of this paper valid. A better normalization, however, might be to divide by $|M|$ itself. An analytical treatment here seems more difficult, though. Therefore we leave this as an open problem for future work.

Acknowledgments

We would like to thank Pedro Feijão and Daniel Doerr for helping us with hints on how to get the simulated data for our experiments.

References

1. Angibaud, S., Fertin, G., Rusu, I., Thévenin, A., Vialette, S.: Efficient tools for computing the number of breakpoints and the number of adjacencies between two genomes with duplicate genes. Journal of Computational Biology 15(8), 1093–1115 (2008)
2. Angibaud, S., Fertin, G., Rusu, I., Thévenin, A., Vialette, S.: On the approximability of comparing genomes with duplicates. Journal of Graph Algorithms and Applications 13(1), 19–53 (2009)
3. Angibaud, S., Fertin, G., Rusu, I., Vialette, S.: A pseudo-boolean framework for computing rearrangement distances between genomes with duplicates. Journal of Computational Biology 14(4), 379–393 (2007)
4. Ausiello, G., Protasi, M., Marchetti-Spaccamela, A., Gambosi, G., Crescenzi, P., Kann, V.: Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edn. (1999)
5. Bafna, V., Pevzner, P.: Genome rearrangements and sorting by reversals. In: Proc. of FOCS 1993. pp. 148–157 (1993)
6. Bergeron, A., Mixtacki, J., Stoye, J.: A unifying view of genome rearrangements. In: Proc. of WABI 2006. LNBI, vol. 4175, pp. 163–173 (2006)
7. Berman, P.: A d/2 approximation for maximum weight independent set in d-claw free graphs. In: Algorithm Theory - SWAT 2000: 7th Scandinavian Workshop on Algorithm Theory Bergen. pp. 214–219 (2000)
8. Berman, P., Karpinski, M.: On some tighter inapproximability results. In: International Colloquium on Automata, Languages, and Programming. pp. 200–209. Springer (1999)
9. Braga, M.D.V., Willing, E., Stoye, J.: Double cut and join with insertions and deletions. Journal of Computational Biology 18(9), 1167–1184 (2011)
10. Braga, M.D.V., Chauve, C., Dörr, D., Jahn, K., Stoye, J., Thévenin, A., Wittler, R.: The potential of family-free genome comparison. In: Chauve, C., El-Mabrouk, N., Tannier, E. (eds.) Models and Algorithms for Genome Evolution, chap. 13, pp. 287–307. Springer, London (2013)

11. Bryant, D.: The complexity of calculating exemplar distances. In: Sankoff, D., Nadeau, J.H. (eds.) Comparative Genomics, pp. 207–211. Kluwer Academic Publishers, Dortrecht (2000)
12. Bulteau, L., Jiang, M.: Inapproximability of (1,2)-exemplar distance. IEEE/ACM Trans. Comput. Biol. Bioinf. 10(6), 1384–1390 (2013)
13. Crescenzi, P.: A short guide to approximation preserving reductions. In: Proceedings of Computational Complexity. Twelfth Annual IEEE Conference. pp. 262–273 (1997)
14. Dalquen, D.A., Anisimova, M., Gonnet, G.H., Dessimoz, C.: Alf – a simulation framework for genome evolution. Molecular Biology and Evolution 29(4), 1115 (2012)
15. Dörr, D., Thévenin, A., Stoye, J.: Gene family assignment-free comparative genomics. BMC Bioinformatics 13(Suppl 19), S3 (2012)
16. Hannenhalli, S., Pevzner, P.: Transforming men into mice (polynomial algorithm for genomic distance problem). In: Proc. of FOCS 1995. pp. 581–592 (1995)
17. Håstad, J.: Some optimal inapproximability results. Journal of the ACM (JACM) 48(4), 798–859 (2001)
18. Hawick, K.A., James, H.A.: Enumerating circuits and loops in graphs with self-arcs and multiple-arcs. Tech. Rep. CSTN-013, Massey University (2008)
19. Johnson, D.: Finding all the elementary circuits of a directed graph. SIAM Journal on Computing 4(1), 77–84 (1975)
20. Martinez, F.V., Feijão, P., Braga, M.D.V., Stoye, J.: On the family-free DCJ distance and similarity. Algorithms for Molecular Biology 10, 13 (2015)
21. Raman, V., Ravikumar, B., Rao, S.S.: A simplified np-complete maxsat problem. Information Processing Letters 65(1), 1 – 6 (1998)
22. Rubert, D.P., Feijão, P., Braga, M.D.V., Stoye, J., Martinez, F.V.: Approximating the DCJ distance of balanced genomes in linear time. Algorithms for Molecular Biology 12, 3 (2017)
23. Sankoff, D.: Edit distance for genome comparison based on non-local operations. In: Proc. of CPM 1992. LNCS, vol. 644, pp. 121–135 (1992)
24. Sankoff, D.: Genome rearrangement with gene families. Bioinformatics 15(11), 909–917 (1999)
25. Shao, M., Lin, Y.: Approximating the edit distance for genomes with duplicate genes under DCJ, insertion and deletion. BMC Bioinformatics 13(Suppl 19), S13 (2012)
26. Shao, M., Lin, Y., Moret, B.: An exact algorithm to compute the DCJ distance for genomes with duplicate genes. In: Proc. of RECOMB 2014. LNBI, vol. 8394, pp. 280–292 (2014)
27. Yancopoulos, S., Attie, O., Friedberg, R.: Efficient sorting of genomic permutations by translocation, inversion and block interchanges. Bioinformatics 21(16), 3340–3346 (2005)

A Proof of APX-hardness and Approximation Ratio Lower Bound

For the APX-hardness proof of problem FFDCJ-SIMILARITY, we first give some definitions based on [13]. Thereby we restrict ourselves to maximization problems and feasible solutions.

Given an instance x of an optimization problem P and a solution y of x, $\mathrm{val}(x, y)$ denotes the value of y, which is a positive integer measure of y. The function val, also referred to as objective function, must be computable in polynomial time. The value of an optimal solution (which maximizes the objective function) is defined as $\mathrm{opt}(x)$. Thus, the *performance ratio* of y with respect to x is defined as:

$$R_P(x, y) = \frac{\mathrm{opt}(x)}{\mathrm{val}(x, y)}. \tag{6}$$

Given two optimization problems P and P', let f be a polynomial-time computable function that maps an instance x of P into an instance $f(x)$ of P', and let g be a polynomial-time computable function that maps a solution y for the instance $f(x)$ of P' into a solution $g(x, y)$ of P. A *reduction* is a pair (f, g). A reduction from P to P' is frequently denoted by $P \leq P'$, and we say that P is *reduced* to P'. A reduction $P \leq P'$ *preserves membership* in a class $\mathcal{C}$ if $P' \in \mathcal{C}$ implies $P \in \mathcal{C}$. An *approximation-preserving* reduction preserves membership in either APX, PTAS, or both classes. The *strict reduction*, which is the simplest type of approximation-preserving reduction, preserves membership in both APX and PTAS classes and must satisfy the following condition:

$$R_P(x, g(x, y)) \leq R_{P'}(f(x), y). \tag{7}$$

We consider the following optimization problem, to be used within the proof of Theorem 1 below:

> **Problem** MAX-2SAT3(ϕ): Given a 2-CNF formula (i.e., with at most 2 literals per clause) $\phi = \{C_1, \ldots, C_m\}$ with n variables $X = \{x_1, \ldots, x_n\}$, where each variable appears in at most 3 clauses, find an assignment that satisfies the largest number of clauses.

The formula ϕ as defined above is called a 2SAT3 formula. MAX-2SAT3 [4, 8] is a special case of MAX-2SATB (also known as B-OCC-MAX-2SAT), where each variable occurs in at most B clauses for some B, which in turn is a restricted version of MAX-2SAT [21].

Theorem 1. FFDCJ-SIMILARITY *is APX-hard and cannot be approximated with approximation ratio better than* $22/21 = 1.0476\ldots$*, unless* $P = NP$.

Proof (Theorem 1, first part). We give a strict reduction (f, g) from MAX-2SAT3 to FFDCJ-SIMILARITY, showing that

$$R_{\text{MAX-2SAT3}}(\phi, g(f(\phi), \gamma)) \leq R_{\text{FFDCJ-SIMILARITY}}(f(\phi), \gamma),$$

for any instance ϕ of MAX-2SAT3 and solution γ of FFDCJ-SIMILARITY with instance $f(\phi)$. Since variables occurring only once imply their clauses and others to be trivially satisfied, we consider only clauses that are not trivially satisfied in their instance. Similar for clauses containing literals x_i and $\overline{x_i}$, for some variable x_i.

(Function f.) We show progressively how to build $GS_\sigma(A, B)$ and define genes and their sequences in chromosomes of A and B. For each variable x_i occurring three times, let Cx_i^1, Cx_i^2 and Cx_i^3 be *aliases* for the clauses where x_i occurs (notice that a clause composed of two literals has two aliases). We define a *variable component* $\mathcal{C}_i$ adding vertices (genes) x_i^1, x_i^2 and x_i^3 to $\mathcal{A}$, vertices (genes) Cx_i^1, Cx_i^2 and Cx_i^3 to $\mathcal{B}$, and edges $ex_i^j = (Cx_i^j, x_i^j)$ and $e\overline{x_i}^j = (Cx_i^j, x_i^k)$ for $j \in \{1, 2, 3\}$ and $k = (j + 1) \bmod 3 + 1$. An edge ex_i^j ($e\overline{x_i}^j$) has weight 1 (0) if the literal x_i ($\overline{x_i}$) belongs to the clause Cx_i^j. Edges in the variable component $\mathcal{C}_i$ form a cycle of length 6 (Fig. 6). Variable components for variables occurring two times are defined in a similar manner. Genomes are $A = \{(x_i^j)$ for each occurrence j of each variable $x_i \in X\}$ and $B = \{(Cx_i^j) : Cx_i^j$ is an alias to a clause in ϕ with only one literal$\} \cup \{(Cx_i^j\ Cx_{i'}^{j'}) : Cx_i^j$ and $Cx_{i'}^{j'}$ are aliases to the same clause in $\phi\}$.

The function f as defined here maps an instance ϕ of MAX-2SAT3 (a 2-CNF formula) to an instance $f(\phi)$ of FFDCJ-SIMILARITY (genomes A and B and $GS_\sigma(A, B)$) and is clearly polynomial. Besides, since all chromosomes are circular, the corresponding weighted adjacency graph $AG_\sigma(A, B)$ (or $AG_\sigma(A^M, B^M)$ for some matching M) is a collection of cycles only.

Now, notice that any maximal matching in $GS_\sigma(A, B)$ covers all genes in both $\mathcal{A}$ and $\mathcal{B}$, inducing in $AG_\sigma(A, B)$ only cycles of length 2, composed by (genes in) chromosomes (x_i^j) and $(Cx_i^{j'})$, or cycles of length 4, composed by chromosomes (x_i^j), (x_k^l) and $(Cx_i^{j'}\ Cx_k^{l'})$.

Define the *normalized weight* of cycle C as $\mu(C) = w(C)/|C|$. In this transformation, each cycle C is such that $\mu(C) = 0, 0.5$ or 1. A cycle C such that $\mu(C) > 0$ is a *helpful cycle* and represents a clause satisfied by one or two literals ($\mu(C) = 0.5$ or $\mu(C) = 1$, respectively). See an example in Fig. 7.

In this scenario, however, a solution of FFDCJ-SIMILARITY with performance ratio r could lead to a solution of MAX-2SAT3 with ratio $2r$, since the total normalized weight for two cycles C_1 and C_2 with $\mu(C_1) = \mu(C_2) = 0.5$ (two clauses satisfied by one literal each) is the same for one cycle C with $\mu(C) = 1.0$ (one clause satisfied by two literals). Therefore, achieving the desired ratio requires some modifications in f. It is not possible to make these two types of cycles have the same weight, but it suffices to get close enough.

We introduce special genes into the genomes called *extenders*. For some p even, for each edge $ex_i^j = (Cx_i^j, x_i^j)$ of weight 1 in $GS_\sigma(A, B)$ we introduce p extenders $\alpha_1, \ldots, \alpha_p$ into A (as a consequence, they are also introduced into $\mathcal{A}$) and p extenders $\alpha_{p+1}, \ldots, \alpha_{2p}$ into B (each ex_i^j of weight 1 has its own set of extenders). Edge ex_i^j is replaced by edges (Cx_i^j, α_1) with weight 1 (which we consider equivalent to ex_i^j) and (α_{p+1}, x_i^j) with weight 0, and edges (α_k, α_{p+k}) with weight

0 are added to $GS_\sigma(A,B)$ for each $1 \leq k \leq p$ (extenders α_1 and α_{p+1} are now part of the variable component $\mathcal{C}_i$). Regarding new chromosomes in genomes A and B, A is updated to $A \cup \{(\alpha_1\ -\alpha_p)\} \cup \{(\alpha_k\ -\alpha_{k+1}) : k \in \{2,4,\ldots,p-2\}\}$ and B to $B \cup \{(\alpha_k\ -\alpha_{k+1}) : k \in \{p+1,p+3,\ldots,2p-1\}\}$. By this construction, which is still polynomial, the path from x_i^{jt} to Cx_i^{jt} in $AG_\sigma(A,B)$ is extended from 1 to $1+p$ edges, from $\{(x_i^{jt}, Cx_i^{jt})\}$ to $\{(x_i^{jt},\alpha_p^t),(\alpha_{p+1}^t,\alpha_2^t),(\alpha_3^t,\alpha_{p+2}^t),(\alpha_{p+3}^t,\alpha_4^t),\ldots,(\alpha_1^t,Cx_i^{jt})\}$. The same occurs for the path from x_i^{jh} to Cx_i^{jh} (see Fig. 8). Now, cycles in $AG_\sigma(A,B)$ induced by edges of weight 0 in $GS_\sigma(A,B)$ have normalized weight 0, cycles previously with normalized weight 1 are extended and have normalized weight $\frac{1}{1+p}$, and cycles previously with normalized weight 0.5 are extended and have normalized weight $\frac{1}{2+p}$. Notice that, for a sufficiently large p, $\frac{1}{1+p}$ is quite close to $\frac{1}{2+p}$, hence the problem of finding the maximum similarity in this graph is very similar to finding the maximum number of helpful cycles.

(Function g.) By the structure of variable components in $GS_\sigma(A,B)$, and since solutions of FFDCJ-SIMILARITY are restricted to maximal matchings only, any solution γ for $f(\phi)$ is a matching that covers only edges ex_i^j or $e\overline{x_i}^j$ for each variable component $\mathcal{C}_i$. For a $\mathcal{C}_i$, if edges ex_i^j ($e\overline{x_i}^j$) are in the solution then the variable x_i is assigned to true (false), inducing in polynomial time an assignment for each $x_i \in X$ and therefore a solution $g(f(\phi),\gamma)$ to MAX-2SAT3. A clause is satisfied if vertices (or the only vertex) corresponding to its aliases are in a helpful cycle.

(Approximation ratio.) Given $f(\phi)$ and a feasible solution γ of FFDCJ-SIMILARITY with the maximum number of helpful cycles, denote by c' the number of helpful cycles in γ. Notice that c' is also the maximum number of satisfied clauses of MAX-2SAT3, that is, the value of an optimal solution for MAX-2SAT3 for any instance ϕ, denoted here by $\mathrm{opt}_{\text{2SAT3}}(\phi)$. Thus, $c' = \mathrm{opt}_{\text{2SAT3}}(\phi)$.

To achieve the desired ratio we must establish some properties and relations between the parameters of MAX-2SAT3 and FFDCJ-SIMILARITY and set some parameters to specific values.

Let $n := |A| = |B|$ before extenders are added. We choose for p (the number of extenders added for each edge of weight 1 in $GS_\sigma(A,B)$) the value $2n$ and define $\omega = \frac{1}{2+p} = \frac{1}{2+2n}$ and

$$\varepsilon = \frac{1}{1+p} - \frac{1}{2+p} = \frac{1}{4n^2+6n+2},$$

which implies that $\omega + \varepsilon = \frac{1}{p+1}$. Thus, it is easy to see that $n\varepsilon < \omega$, i.e.,

$$\varepsilon < \frac{\omega}{n} < 1. \tag{8}$$

If $\mathrm{opt}_{\text{SIM}}(f(\phi))$ denotes the value of an optimal solution for FFDCJ-SIMILARITY with instance $f(\phi)$ and c^* denotes the number of helpful cycles in an optimal solution of FFDCJ-SIMILARITY, then we have immediately that

$$\frac{\mathrm{opt}_{\text{SIM}}(f(\phi))}{\omega+\varepsilon} \leq c^* \leq \frac{\mathrm{opt}_{\text{SIM}}(f(\phi))}{\omega}. \tag{9}$$

Besides that

$$0 \leq c^* \leq n, \tag{10}$$

and

$$c^*\omega \leq \text{opt}_{\text{SIM}}(f(\phi)) \leq c^*(\omega + \varepsilon). \tag{11}$$

Thus, we have

$$\begin{aligned} c^*(\omega + \varepsilon) &= c^*\omega + c^*\varepsilon \\ &< c^*\omega + \frac{c^*\omega}{n} && (12) \\ &\leq c^*\omega + 1 \cdot \omega && (13) \\ &= c^*\omega + \omega, && (14) \end{aligned}$$

where (12) comes from (8) and (13) is valid due to (10).

Now, let c^r be the number of helpful cycles given by an approximate solution for the FFDCJ-SIMILARITY with approximation ratio r. Then,

$$R_{\text{MAX-2SAT3}}(\phi, g(f(\phi), \gamma)) = \frac{\text{opt}_{\text{2SAT3}}(\phi)}{c^r} = \frac{c'}{c^r} \leq r,$$

where the last inequality is given by Proposition 2 below. This concludes the first part of the proof. □

Proposition 1. *Let c' be the number of helpful cycles in a feasible solution of* FFDCJ-SIMILARITY *with the greatest number of helpful cycles possible. Let c^* be the number of helpful cycles in an optimal solution of* FFDCJ-SIMILARITY*. Then,*

$$c' = c^*.$$

Proof. Since c' is the greatest number of helpful cycles possible, it is immediate that $c^* \leq c'$.

Let us now show that $c^* \geq c'$. Suppose for a moment that $c^* < c'$. Since c^* and c' are integers, this implies that $c^* + 1 \leq c'$, i.e.,

$$c^* \leq c' - 1. \tag{15}$$

Let $\mathcal{C}'$ be the set of cycles with c' cycles, i.e., with the maximum number of helpful cycles possible. Let $\mu(\mathcal{C}') := \sum_{C \in \mathcal{C}'} \mu(C) = \sum_{C \in \mathcal{C}'} w(C)/|C|$. Then

$$\begin{aligned} \mu(\mathcal{C}') \geq c'\omega &= (c' - 1)\omega + \omega \\ &\geq c^*\omega + \omega && (16) \\ &> c^*(\omega + \varepsilon) && (17) \\ &\geq \text{opt}_{\text{SIM}}(f(\phi)), && (18) \end{aligned}$$

where (16) follows from (15), (17) comes from (14), and (18) is valid due to (11). It means that $\mu(\mathcal{C}') > \text{opt}_{\text{SIM}}(f(\phi))$, which is a contradiction.

Therefore, $c' = c^*$. □

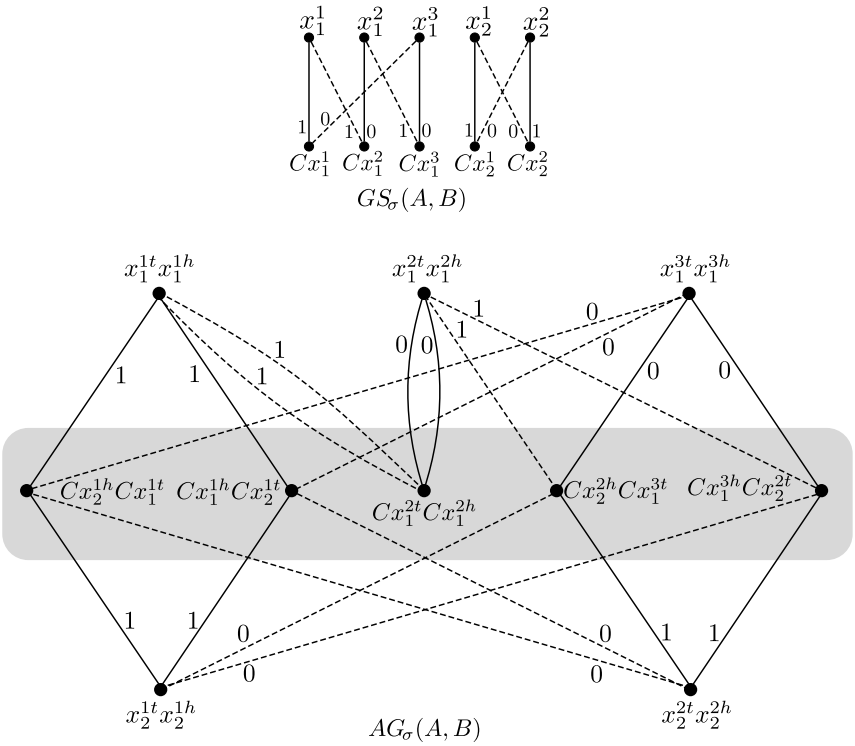

Fig. 6. $GS_\sigma(A,B)$ and $AG_\sigma(A,B)$ for genomes $A = \{(x_1^1),(x_1^2),(x_1^3),(x_2^1),(x_2^2)\}$ and $B = \{(Cx_1^1\ Cx_2^1),(Cx_1^2),(Cx_1^3\ Cx_2^2)\}$ given by function f (Theorem 1) applied to 2SAT(3) clauses $C_1 = (x_1 \vee x_2)$, $C_2 = (\overline{x_1})$ and $C_3 = (\overline{x_1} \vee x_2)$. In $GS_\sigma(A,B)$, solid edges correspond to ex_i^j and dashed edges correspond to $e\overline{x_i}^j$. In $AG_\sigma(A,B)$, shaded region corresponds to genes of genome B, and solid (dashed) edges correspond to solid (dashed) edges of $GS_\sigma(A,B)$.

Proposition 2. *Let c^r be the number of helpful cycles given by an approximate solution for* FFDCJ-SIMILARITY *with approximation ratio r. Let c' be the same as defined in Proposition 1. Then,*

$$c^r \geq \frac{c'}{r}.$$

Proof. Given an instance $f(\phi)$ of FFDCJ-SIMILARITY, let γ^r be an approximate solution of $f(\phi)$ with performance ratio r, i.e., $\text{val}(f(\phi),\gamma^r) \geq \frac{\text{opt}_{\text{SIM}}(f(\phi))}{r}$. Let c^r

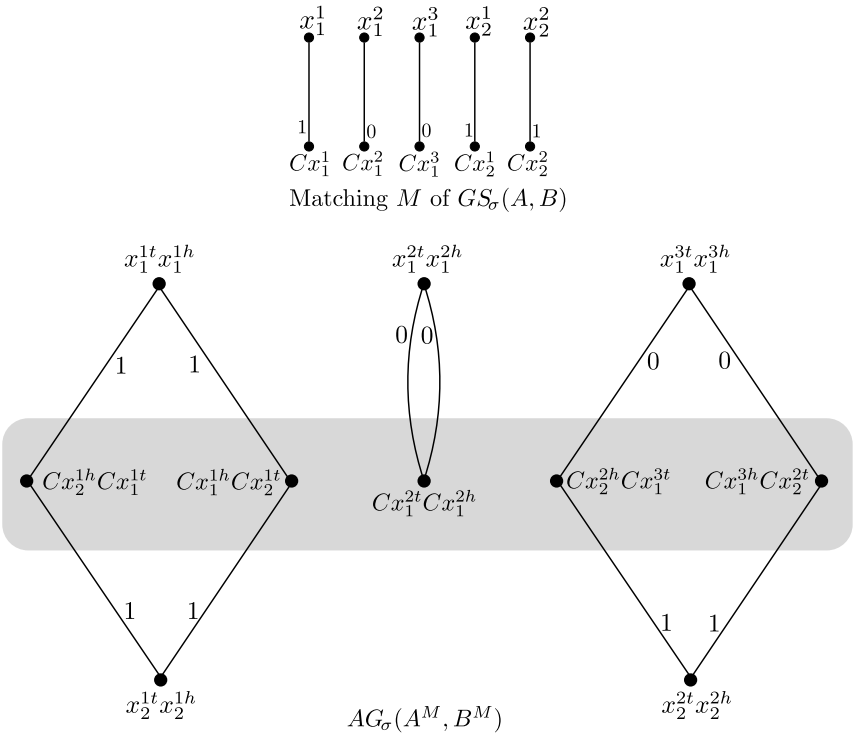

Fig. 7. A matching M of $GS_\sigma(A, B)$ and cycles induced by M in and $AG_\sigma(A^M, B^M)$ for genomes of Fig. 6. This solution of FFDCJ-SIMILARITY represents clauses C_1 and C_3 of MAX-2SAT3 satisfied.

be the number of helpful cycles of γ^r. Then

$$
\begin{aligned}
c^r &\geq \frac{\left(\frac{\mathrm{opt}_{\mathrm{SIM}}(f(\phi))}{r}\right)}{\omega + \epsilon} \\
&> \frac{\mathrm{opt}_{\mathrm{SIM}}(f(\phi))}{r(\omega + \omega/n)} && (19) \\
&= \frac{\mathrm{opt}_{\mathrm{SIM}}(f(\phi))}{r\omega} \cdot \frac{n}{n+1} \\
&\geq \frac{c'\omega}{r\omega} \cdot \frac{n}{n+1} && (20) \\
&= \frac{c'}{r} \cdot \left(1 - \frac{1}{n+1}\right) \\
&= \frac{c'}{r} - \frac{c'}{r(n+1)} \\
&\geq \frac{c'}{r} - 1 && (21)
\end{aligned}
$$

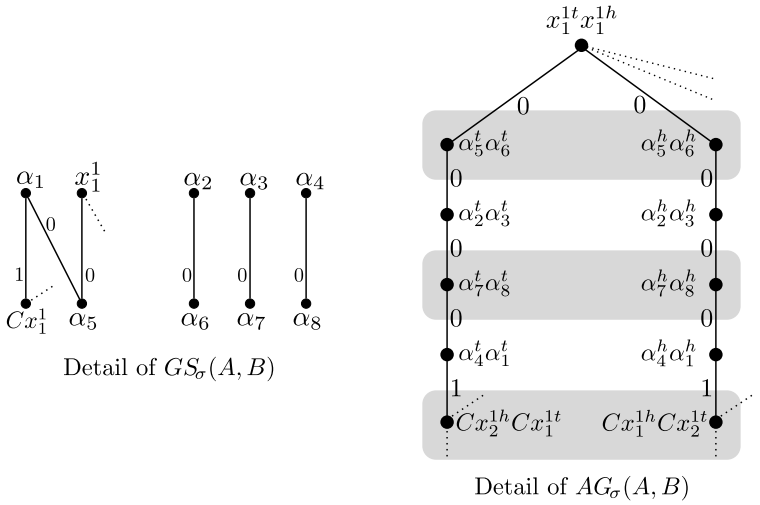

Fig. 8. Detail of graphs $GS_\sigma(A,B)$ and $AG_\sigma(A,B)$ for genomes of Fig. 6 including extenders for edge (x_1^1, Cx_1^1) for $p = 4$. Shaded region corresponds to genes of genome B. Extending all edges of weight 1 and selecting the matching of Fig. 7, this helpful cycle (only half of it is in this figure) would have normalized weight $\frac{4}{4(p+1)} = \frac{1}{p+1} = \frac{1}{5} = 0.2$.

where (19) follows from (8), (20) is valid from (11) and Proposition 1. Then, from (21) we know that $c^r > \frac{c'}{r} - 1$ and, since c^r is an integer number, the result follows.

□

We now continue with the proof of Theorem 1.

Proof (Theorem 1, second part). First, notice that if a problem is APX-hard, the existence of a PTAS for it implies P = NP. Since a strict reduction preserves membership in the class PTAS, finding a PTAS for FFDCJ-SIMILARITY implies a PTAS for every APX-hard problem and P = NP. A PTAS for FFDCJ-SIMILARITY would also imply an approximation ratio better than $2012/2011 = 1.0005\ldots$, unless P = NP. This follows immediately from the reduction in Theorem 1 with $R_{\text{MAX-2SAT3}} = R_{\text{FFDCJ-SIMILARITY}}$ and the fact that MAX-2SAT3 is shown in [8] to be NP-hard to approximate within a factor of $2012/2011 - \varepsilon$ for any $\varepsilon > 0$.

However, our result is slightly stronger. Notice particularly that the reduction MAX-2SAT3 $\leq$ FFDCJ-SIMILARITY from the first part of the proof can be trivially extended to MAX-2SAT $\leq$ FFDCJ-SIMILARITY by extending variable components to arbitrary sizes. This increases the lower bound to $22/21 = 1.0476\ldots$ [17]. □