A Fast and Exact Algorithm for the Exemplar Breakpoint Distance

Mingfu Shao(✉) and Bernard M.E. Moret

Laboratory for Computational Biology and Bioinformatics, EPFL, Lausanne, Switzerland
{mingfu.shao,bernard.moret}@epfl.ch

Abstract. A fundamental problem in comparative genomics is to compute the distance between two genomes. For two genomes without duplicate genes, we can easily compute a variety of distance measures in linear time, but the problem is NP-hard under most models when genomes contain duplicate genes. Sankoff proposed the use of exemplars to tackle the problem of duplicates genes and gene families: each gene family is represented by a single gene (the exemplar for that family), chosen so as to optimize some metric. Unfortunately, choosing exemplars is itself an NP-hard problem. In this paper, we propose a very fast and exact algorithm to compute the exemplar breakpoint distance, based on new insights in the underlying structure of genome rearrangements and exemplars. We evaluate the performance of our algorithm on simulation data and compare its performance to the best effort to date (a divide-and-conquer approach), showing that our algorithm runs much faster and scales much better. We also devise a new algorithm for the generalized breakpoint distance problem, which can then be applied to assign orthologs. We compare our algorithm with the state-of-the-art method MSOAR by assigning orthologs among five well annotated mammalian genomes, showing that our algorithm runs much faster and is slightly more accurate than MSOAR.

Keywords: Exemplar · Breakpoint distance · ILP · Orthology assignment

1 Introduction

The combinatorics and algorithmics of genomic rearrangements have been the subject of much research in comparative genomics since the problem was formulated in the 1990s [1]. Perhaps the most fundamental problem is the computation of some distance measure between two genomes. When the two genomes being compared have no duplicate genes, we have linear-time algorithms for most of these distance problems, such as the breakpoint distance [2], the inversion distance [3], and the DCJ distance [4,5].

However, gene duplications are widespread events and have long been recognized as a major driving force of evolution [6,7]. To compare two genomes

T. Przytycka (Ed.): RECOMB 2015, LNBI 9029, pp. 309–322, 2015.
DOI: 10.1007/978-3-319-16706-0_31

with duplicate genes, Sankoff [8] formulated the exemplar distance problem: select exactly one gene in each gene family in each genome, such that some measure between the two resulting genomes (without duplicate genes) is minimized. Unfortunately, the exemplar distance problem is NP-hard even under the simplest distance measures, breakpoint and inversion distances [9]; moreover, the exemplar breakpoint distance cannot be approximated in polynomial time unless P = NP [10,11].

Several exact algorithms have been proposed for these problems. In his original paper, Sankoff [8] gave a first branch-and-bound algorithm for the exemplar breakpoint and inversion distances. Nguyen *et al.* gave a much faster divide-and-conquer approach for the exemplar breakpoint distance [12], while Angibaud *et al.* [13] gave an integer programming formulation of the exemplar breakpoint distance.

In this paper, we propose a fast and exact algorithm for the exemplar breakpoint distance problem. In Section 2, we formally state the problem using the equivalent definition of shared adjacencies (rather than breakpoints). In Section 3, we describe our algorithm, which consists of an integer linear program formulation and two specific techniques, one is to add a collection of new constraints to the ILP formulation and thereby reduce its search space, and the other is to identify optimal substructures and thereby reduce the size of the instance. In Section 4, we evaluate the efficiency of these two techniques and compare our full algorithm with the divide-and-conquer algorithm of Nguyen *et al.* through simulation. In Section 5, we formulate the generalized breakpoint distance, which naturally infers a subset of orthologs under a parsimonious assumption. We then design an algorithm for it by iteratively applying the exact algorithm, and compare its performance with MSOAR [14] among five genomes.

2 Problem Statement

We model each genome as a set chromosomes and each chromosome as a (linear or circular) list of genes. Each gene is represented by a signed (+ or −) symbol. For each linear chromosome in the genome, we always add one *capping* gene, represented by 0, to each of its two ends. Given a chromosome, we can reverse the list of symbols and switch all the signs, which will result in the same chromosome. For example, an equivalent representation of a linear chromosome $(0, g_1, g_2, \cdots, g_m, 0)$ is $(0, -g_m, \cdots, -g_2, -g_1, 0)$.

Genes in the input genomes are grouped into *gene families*. We place all of the capping "genes" into a single gene family of their own, denoted by f_0. In this paper, all the non-capping genes from the same family are represented by the same symbol (the one used to represent their gene family), plus a distinct subscript to distinguish them. Given a genome G, we denote by $\mathcal{A}(G)$ the set of all the gene families in G. We define $\widetilde{\mathcal{A}}(G) = \mathcal{A}(G) \setminus \{f_0\}$. For a gene family $f \in \mathcal{A}(G)$, we use $F(G, f)$ to denote the set of genes in G that come from f. Given a genome G, we can delete all but one gene for each non-capping gene

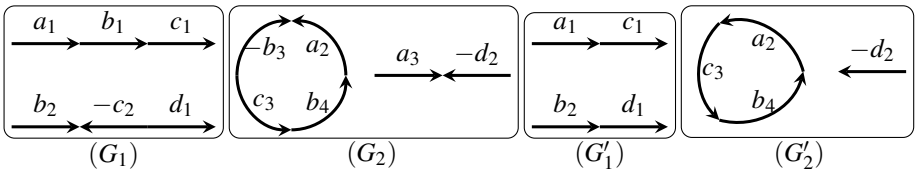

Fig. 1. G_1 contains two linear chromosomes, and G_2 contains one linear and one circular chromosomes. $\mathcal{A}(G_1) = \mathcal{A}(G_2) = \{0, a, b, c, d\}$. G'_1 and G'_2 are two exemplars of G_1 and G_2, respectively. There are two PSSAs, $q_1 = \langle (a_1, c_1), (a_2, c_3) \rangle$ and $q_2 = \langle (d_1, 0), (0, -d_2) \rangle$, between G'_1 and G'_2. which means that $s(G'_1, G'_2) = 2$. We have $\mathcal{P}(G_1, G_2) = \{\langle [0, a_1], [0, a_3] \rangle, \langle [a_1, c_1], [a_2, c_3] \rangle, \langle [b_1, c_1], [b_4, c_3] \rangle, \langle [b_2, -c_2], [-b_3, c_3] \rangle, \langle [0, d_1], [-d_2, 0] \rangle, \langle [d_1, 0], [0, -d_2] \rangle\}$. There are two PSSPAs, $p_1 = \langle [a_1, c_1], [a_2, c_3] \rangle$ and $p_2 = \langle [d_1, 0], [0, -d_2] \rangle$, that survive in G'_1 and G'_2, and p_1 and p_2 are the indicators of q_1 and q_2, respectively.

family, resulting in a new genome satisfying $|F(G, f)| = 1$ for all $f \in \widetilde{\mathcal{A}}(G)$, called an *exemplar* of G.

Two consecutive genes g and h on the same chromosome form an *adjacency*, written as (g, h). Notice that (g, h) and $(-h, -g)$ are the same adjacency. We say an adjacency (g, h) is *simple* if g and h come from different gene families. Given two genomes G_1 and G_2, we say two simple adjacencies $(g_1, h_1) \in G_1$ and $(g_2, h_2) \in G_2$ form a *pair of shared simple adjacencies* (PSSA), written as $\langle (g_1, h_1), (g_2, h_2) \rangle$, if g_1 and g_2 (and also h_1 and h_2) have the same sign and come from the same gene family. We use $s(G_1, G_2)$ to denote the number of PSSAs between G_1 and G_2. Given two genomes G_1 and G_2 with $\widetilde{\mathcal{A}}(G_1) = \widetilde{\mathcal{A}}(G_2)$, the *exemplar breakpoint distance problem* is to compute a pair of exemplars, G'_1 and G'_2 for G_1 and G_2 respectively, such that $s(G'_1, G'_2)$ is maximized (see Figure 1 for an example).

3 Algorithm

We describe a fast and exact algorithm for the exemplar breakpoint distance problem. The algorithm consists of a preprocessing phase to reduce the complexity of the instance, followed by an integer linear program (ILP) to solve the reduced instance. In the preprocessing phase, the algorithm iteratively identifies optimal substructures, fixing in place those it finds, until no such substructure can be found. The reduced instance is then formulated as an ILP. By using the properties of the optimal solution, we can add a collection of new constraints to the ILP, thereby reducing the search space and making the ILP very efficient. For ease of description, we present first the ILP formulation, in Section 3.1, then the algorithm for adding the extra constraints, in Section 3.2, and finally the algorithm for identifying the optimal substructures, in Section 3.3.

3.1 ILP Formulation

We first generalize the definition of adjacency. We say two genes g and h on the same chromosome form a *potential adjacency*, written as $[g, h]$, if we can remove all genes between g and h and yet retain at least one gene in every gene family in every genome. (Adjacencies are just potential adjacencies where nothing need be removed.) We say a potential adjacency $[g, h]$ is *simple* if g and h come from different gene families. Given two genomes G_1 and G_2, we say two simple potential adjacencies $[g_1, h_1] \in G_1$ and $[g_2, h_2] \in G_2$ form a *pair of shared simple potential adjacencies* (PSSPA), written as $\langle [g_1, h_1], [g_2, h_2] \rangle$, if g_1 and g_2 (and also h_1 and h_2) have the same sign and come from the same gene family. Given a pair of exemplars G_1' and G_2' of G_1 and G_2 respectively, we say a PSSPA $p = \langle [g_1, h_1], [g_2, h_2] \rangle$ between G_1 and G_2 *survives* in G_1' and G_2' if $q = \langle (g_1, h_1), (g_2, h_2) \rangle$ is a PSSA between G_1' and G_2'; we then say that the PSSPA p is the *indicator* of the PSSA q. We denote by $\mathcal{P}(G_1, G_2)$ the set of all the PSSPAs between G_1 and G_2 (see Figure 1). Clearly, all PSSAs in any pair of exemplars of G_1 and G_2 can find their indicators in $\mathcal{P}(G_1, G_2)$. We say a set of PSSPAs P can *coexist* wrt G_1 and G_2 if there exists a pair of exemplars G_1' and G_2' of G_1 and G_2 such that all PSSPAs in P survive in G_1' and G_2' simultaneously. Thus, the exemplar breakpoint distance problem can be restated as computing a subset of $\mathcal{P}(G_1, G_2)$ with maximum cardinality that can coexist—and this new perspective leads to our ILP formulation.

We use two types of variables. First, for every gene g in the two given genomes, we use one binary variable x_g to indicate whether this gene appears in the final pair of exemplars. Second, for every PSSPA $p \in \mathcal{P}(G_1, G_2)$ we use one binary variable y_p to indicate whether p survives.

Our ILP has two types of general constraints. First, we require that for each non-capping gene family in each genome, exactly one gene appears in the final exemplar:

$$\sum_{g \in F(G_1, f)} x_g = 1, \quad \forall f \in \widetilde{\mathcal{A}}(G_1),$$
$$\sum_{g \in F(G_2, f)} x_g = 1, \quad \forall f \in \widetilde{\mathcal{A}}(G_2).$$

Second, for each PSSPA $p = \langle [g_1, h_1], [g_2, h_2] \rangle$, we require that, if p survives, then the two pairs, $\langle g_1, g_2 \rangle$ and $\langle h_1, h_2 \rangle$, must appear in the final exemplars, while those genes between g_1 and h_1 (and also between g_2 and h_2) cannot appear in the final exemplars:

$$y_p \leq x_{g_1}, x_{g_2}, x_{h_1}, x_{h_2}, \forall p = \langle [g_1, h_1], [g_2, h_2] \rangle \in \mathcal{P}(G_1, G_2),$$
$$x_g \leq 1 - y_p, \qquad \forall g \text{ between } g_1 \text{ and } h_1 \text{ or between } g_2 \text{ and } h_2.$$

The objective of the ILP is to maximize the sum of the variables for PSSPAs:

$$\max \sum_{p \in \mathcal{P}(G_1, G_2)} y_p.$$

Our ILP formulation is similar to the one proposed in [13], but has fewer variables; it uses some of the same ideas we used for the DCJ distance problem [15]. In our testing, our ILP instances are solved with GUROBI.

3.2 Adding Inference Constraints

The constraints we add are based on two new properties of optimal exemplars.

Lemma 1. *Let G_1 and G_2 be two exemplars. If we remove one non-capping gene from the same gene family for each genome, resulting in two new genomes $\overline{G_1}$ and $\overline{G_2}$ respectively, then we have $s(\overline{G_1}, \overline{G_2}) \geq s(G_1, G_2) - 1$.*

Proof. Let g_1 and g_2 be the two genes that are removed from G_1, resp. G_2, Assume that g_1 and g_2 have the same sign; otherwise we can reverse the chromosome on g_2 is found. Let $\overleftarrow{g_1}$ and $\overrightarrow{g_1}$ (resp. $\overleftarrow{g_2}$ and $\overrightarrow{g_2}$) be the predecessor and successor of g_1 (resp. g_2). In terms of adjacencies, we have that $(\overleftarrow{g_1}, g_1)$ and $(g_1, \overrightarrow{g_1})$ (resp. $(\overleftarrow{g_2}, g_2)$ and $(g_2, \overrightarrow{g_2})$) are removed from G_1 (resp. G_2), and $(\overleftarrow{g_1}, \overrightarrow{g_1})$ (resp. $(\overleftarrow{g_2}, \overrightarrow{g_2})$) is added to G_1 (resp. G_2). If $(\overleftarrow{g_1}, g_1)$ and $(\overleftarrow{g_2}, g_2)$ do not form a PSSA, or $(g_1, \overrightarrow{g_1})$ and $(g_2, \overrightarrow{g_2})$ do not form a PSSA, then we lose at most one PSSA and thus the conclusion holds. Otherwise, $(\overleftarrow{g_1}, \overrightarrow{g_1})$ and $(\overleftarrow{g_2}, \overrightarrow{g_2})$ must form a new PSSA between $\overline{G_1}$ and $\overline{G_2}$, so the conclusion also holds. □

Let G_1' and G_2' be a pair of exemplars of G_1 and G_2, respectively. We say a PSSA $p = \langle (g_1, h_1), (g_2, h_2) \rangle$ between G_1 and G_2 *agrees with* G_1' and G_2' if either both $\langle g_1, g_2 \rangle$ and $\langle h_1, h_2 \rangle$ appear in G_1' and G_2', or neither $\langle g_1, g_2 \rangle$ nor $\langle h_1, h_2 \rangle$ appears in G_1' and G_2'.

Lemma 2. *For any PSSA p between G_1 and G_2, there always exists an optimal pair of exemplars of G_1 and G_2 with which p agrees.*

Proof. Let $p = \langle (g_1, h_1), (g_2, h_2) \rangle$ be a PSSA. Suppose that there exists one optimal pair of exemplars G_1^* and G_2^* with which p does not agree; otherwise the lemma is proved. Without loss of generality, we assume that $\langle g_1, g_2 \rangle$ appears in G_1^* and G_2^*, but $\langle h_1, h_2 \rangle$ does not (thus h_1 and h_2 are non-capping genes). Let h_1' and h_2' be the two genes in G_1^* and G_2^* that come from the same gene family as h_1 and h_2 (we may have $h_1' = h_1$ or $h_2' = h_2$, but not both). Let $\overline{G_1^*}$ and $\overline{G_2^*}$ be the two genomes after removing $\langle h_1', h_2' \rangle$ from G_1^* and G_2^*. According to Lemma 1, we have $s(\overline{G_1^*}, \overline{G_2^*}) \geq s(G_1^*, G_2^*) - 1$. We then insert $\langle h_1, h_2 \rangle$ into $\overline{G_1^*}$ and $\overline{G_2^*}$ to create the PSSA $\langle (g_1, h_1), (g_2, h_2) \rangle$, resulting in two new genomes G_1' and G_2'. Clearly, we have $s(G_1', G_2') = s(\overline{G_1^*}, \overline{G_2^*}) + 1$. Combining these two formulas yields $s(G_1', G_2') \geq s(G_1^*, G_2^*)$, which implies that G_1' and G_2' are also an optimal pair of exemplars of G_1 and G_2. □

Given a PSSA $\langle (g_1, h_1), (g_2, h_2) \rangle$, Lemma 2 allows us to add the following *inference constraints* to the ILP, which guarantee that, for the two pairs of genes $\langle g_1, g_2 \rangle$ and $\langle h_1, h_2 \rangle$, the appearance of one pair implies the appearance of the other pair:

$$x_{h_1}, x_{h_2} \geq x_{g_1} + x_{g_2} - 1,$$
$$x_{g_1}, x_{g_2} \geq x_{h_1} + x_{h_2} - 1.$$

However, we cannot add such constraints simultaneously for two or more PSSAs, as the following example demonstrates.

Example 1. Let $G_1 = (0, a_1, b_1, c_1, d_1, a_2, e_1, 0)$ and $G_2 = (0, a_3, b_2, e_2, d_2, a_4, c_2, 0)$. We have $|F(G_1, f)| = |F(G_2, f)| = 1$ for each $f \in \{b, d, e\}$. If the appearance of $\langle b_1, b_2 \rangle$ implies that of $\langle a_1, a_3 \rangle$ *and* the appearance of $\langle d_1, d_2 \rangle$ implies that of $\langle a_2, a_4 \rangle$, we get two genes from gene family a in each exemplar.

For a PSSA p, we denote by $\mathcal{A}(p)$ the two gene families in p. We say the pair of PSSAs p and q are *independent*, if we have $\mathcal{A}(p) = \mathcal{A}(q)$, or $\mathcal{A}(p) \cap \mathcal{A}(q) = \emptyset$. Given two genomes G_1 and G_2, we denote by $\mathcal{Q}(G_1, G_2)$ the set of all PSSAs between G_1 and G_2 (see Figure 2 for an example). We say a subset $Q \subset \mathcal{Q}(G_1, G_2)$ is *mutually* independent, if every pair of PSSAs in Q is independent.

Lemma 3. *Let $Q \subset \mathcal{Q}(G_1, G_2)$ be a set of PSSAs. If Q is mutually independent, then there exists an optimal pair of exemplars with which every PSSA in Q agrees.*

Proof. Let G_1^* and G_2^* be any optimal pair of exemplars. Suppose that there exists one PSSA $p = \langle (g_1, h_1), (g_2, h_2) \rangle \in Q$ that does not agree with G_1^* and G_2^*; otherwise the lemma is proved. Without loss of generality, we assume that $\langle g_1, g_2 \rangle$ appears in G_1^* and G_2^*, but $\langle h_1, h_2 \rangle$ does not. As we did in the proof of Lemma 2, now we remove the pair of genes $\langle h_1', h_2' \rangle$ that come from the same gene family as $\langle h_1, h_2 \rangle$ in G_1^* and G_2^*, and then insert $\langle h_1, h_2 \rangle$ into G_1^* and G_2^*, thereby producing a new optimal pair of exemplars G_1' and G_2'. Clearly, p agrees with G_1' and G_2'.

We now show that those PSSAs that agree with G_1^* and G_2^* will also agree with G_1' and G_2'. To prove that, we need only consider those PSSAs affected by the removal of $\langle h_1', h_2' \rangle$. If we have another PSSA $q \in Q$ that contains $\langle h_1', h_2' \rangle$, say $q = \langle (h_1', x_1), (h_2', x_2) \rangle$, then $\langle x_1, x_2 \rangle$ must come from the same gene family as $\langle g_1, g_2 \rangle$, because p and q are independent. Since $\langle g_1, g_2 \rangle$ appears in G_1^* and G_2^*, we know that $\langle x_1, x_2 \rangle$ does not appear, which implies that q does not agree with G_1^* and G_2^*, but does agree with G_1' and G_2'. If we have another PSSA $q \in Q$ that contains h_1' or h_2' but not $\langle h_1', h_2' \rangle$, e.g., $q = \langle (h_1', x_1), (h_3, x_2) \rangle$, then for the same reason we know that $\langle x_1, x_2 \rangle$ comes from the same gene family as $\langle g_1, g_2 \rangle$, which implies that q agrees with G_1^* and G_2^* and also agrees with G_1' and G_2'. Thus, comparing with G_1^* and G_2^*, we have more PSSAs agreeing with G_1' and G_2'. We can repeat this procedure and end up with an optimal pair of exemplars with which all PSSAs in Q agree. □

Hence, in order to add simultaneous inference constraints to the ILP, we need to find mutually independent PSSAs. We thus focus on the following optimization problem: given $\mathcal{Q}(G_1, G_2)$, compute a mutually independent subset $Q \subset \mathcal{Q}(G_1, G_2)$ of maximum cardinality. We now give an efficient algorithm for this problem by reducing it to a maximum-weight matching problem.

We build the graph $G = (V, E)$ as follows. For each gene family in $\mathcal{A}(G_1) \cup \mathcal{A}(G_2)$, we add one vertex to V. For each PSSA $p \in \mathcal{Q}(G_1, G_2)$, we check whether

there is already an edge between the two vertices corresponding to the two gene families in $\mathcal{A}(p)$. If such an edge already exists, we increase its weight by 1; otherwise, we create it and set its weight to 1. Clearly, two PSSAs in $\mathcal{Q}(G_1, G_2)$ are not independent if and only if their corresponding edges share exactly one vertex in G. Thus, the mutually independent subset $Q \subset \mathcal{Q}(G_1, G_2)$ with maximum cardinality corresponds to the matching in G with maximum total weight—see Figure 2 for an example. We can use Edmonds's algorithm to compute the maximum-weight matching in G, which gives us the mutually independent subset $Q \subset \mathcal{Q}(G_1, G_2)$ of maximum cardinality. We can then safely add these inference constraints to the ILP for all PSSAs in Q simultaneously. The efficiency of these constraints is studied in Section 4.

3.3 Identifying Optimal Substructures

We say n consecutive genes $(g_1, g_2, \cdots, g_n)$ on one chromosome form a *segment* of length n. Notice that $(g_1, g_2, \cdots, g_n)$ and $(-g_n, \cdots, -g_2, -g_1)$ are the same segment. For a segment t of length n, we use $\mathcal{A}(t)$ to denote the set of gene families among these genes in t, and we say t is *simple* if we have $\mathcal{A}(t) = n$. Given two genomes G_1 and G_2, we say two simple segments $(g_1, g_2, \cdots, g_n) \in G_1$ and $(h_1, h_2, \cdots, h_n) \in G_2$ form a *pair of shared simple segments* (PSSS for short), written as $\langle (g_1, g_2, \cdots, g_n), (h_1, h_2, \cdots, h_n) \rangle$, if g_i and h_i have the same sign and come from the same gene family for all $1 \leq i \leq n$ (see Example 2). Intuitively, those PSSSs between two genomes are more likely to stay in the optimal exemplars, since each PSSS of length n can contribute $(n-1)$ PSSAs to the exemplars. In this section we give a sufficient condition for a given PSSS to be contained in some optimal exemplars, and devise an algorithm to test the condition.

Let $p = \langle (g_1, g_2, \cdots, g_n), (h_1, h_2, \cdots, h_n) \rangle$ be a PSSS between G_1 and G_2. We say a PSSPA $\langle [x_1, y_1], [x_2, y_2] \rangle$ *spans* p if $(g_1, \cdots, g_n)$ is between x_1 and y_1, or $(h_1, \cdots, h_n)$ is between x_2 and y_2, or both. We denote by $S_1(p) \subset \mathcal{P}(G_1, G_2)$ the set of all PSSPAs that span p. Let $S_2(p)$ be the set of PSSPAs that consist of at least one gene pair in p, i.e., those PSSPAs $\langle [x_1, y_1], [x_2, y_2] \rangle$ satisfying $\langle x_1, x_2 \rangle = \langle g_i, h_i \rangle$, or $\langle y_1, y_2 \rangle = \langle g_j, h_j \rangle$, or both, for some $1 \leq i \neq j \leq n$. Let $S_3[p]$ be the set of PSSPAs that consist of at least one gene pair from gene family

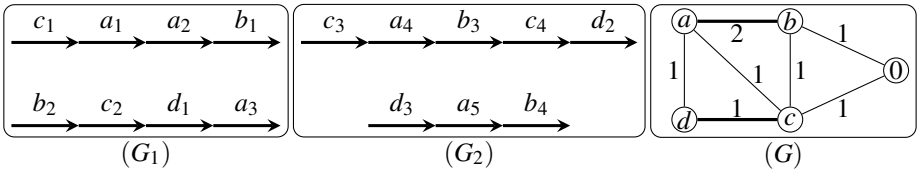

Fig. 2. $\mathcal{Q}(G_1, G_2) = \{\langle (0, c_1), (0, c_3) \rangle, \langle (c_1, a_1), (c_3, a_4) \rangle, \langle (a_2, b_1), (a_4, b_3) \rangle, \langle (a_2, b_1), (a_5, b_4) \rangle, \langle (b_1, 0), (b_4, 0) \rangle, \langle (b_2, c_2), (b_3, c_4) \rangle, \langle (c_2, d_1), (c_4, d_2) \rangle, \langle (d_1, a_3), (d_3, a_5) \rangle\}$. G is the corresponding graph, in which one maximum weight matching is shown as bold edges.

in $\mathcal{A}(p)$, i.e., those PSSPAs $\langle [x_1, y_1], [x_2, y_2] \rangle$ such that the gene family of $\langle x_1, x_2 \rangle$ or of $\langle y_1, y_2 \rangle$, or of both, comes from $\mathcal{A}(p)$. Clearly, we have $S_2(p) \subset S_3(p)$. Set $S(p) = S_1(p) \cup S_3(p) \setminus S_2(p)$ and let $m(p)$ be the maximum number of PSSPAs in $S(p)$ that can coexist wrt G_1 and G_2 (see Example 2). Intuitively, $m(p)$ is the maximum number of PSSAs that can be destroyed by the appearance of p. Thus, if we have $m(p) \leq n-1$, where $n-1$ is the number of PSSAs inside p, then some optimal solution must include p. Formally, we have the following lemma.

Lemma 4. *Let p be a PSSS with n gene pairs between G_1 and G_2 and assume $m(p) \leq n-1$; then there exists an optimal pair of exemplars of G_1 and G_2 that contains p.*

Proof. We prove the lemma by contradiction. Write $p = \langle (g_1, g_2, \cdots, g_n), (h_1, h_2, \cdots, h_n) \rangle$ and let G_1^* and G_2^* be an optimal pair of exemplars of G_1 and G_2 that does not contain p. We can assume that G_1^* and G_2^* do not contain any pair of genes in p, i.e., do not contain $\langle g_i, h_i \rangle$ for any $1 \leq i \leq n$, since otherwise, we can iteratively apply Lemma 2 to build another optimal pair of exemplars that include all pairs of genes in p, which contradicts the assumption.

Now we modify G_1^* and G_2^* to create a new pair of exemplars G_1' and G_2'. We first remove those n gene pairs coming from the gene families in $\mathcal{A}(p)$. Exactly those PSSAs between G_1^* and G_2^* that contain at least one gene pair with gene family in $\mathcal{A}(p)$ are destroyed. We denote by P_3 those PSSAs that are destroyed in this step. Second, we insert p back to G_1^* and G_2^*. This step can destroy at most two PSSAs between G_1^* and G_2^*, since the insertion of two segments (one in each genome) will break only two adjacencies. We denote by P_1 the set of PSSAs that are destroyed in this step.

Let I_3 and I_1 be the sets of the indicators of P_3 and P_1, respectively. According to the construction, we have $I_3 \subset S_3(p) \setminus S_2(p)$ and $I_1 \subset S_1(p)$. Moreover, these PSSPAs in $I_3 \cup I_1$ can coexist wrt G_1 and G_2, since P_3 and P_1 can appear in a pair of exemplars, G_1^* and G_2^*, simultaneously. Using the condition, we get $|P_3 \cup P_1| = |I_3 \cup I_1| \leq n-1$, which means that, during the modification, at most $(n-1)$ PSSAs are destroyed. On the other hand, the insertion of p introduces $(n-1)$ new PSSAs to G_1' and G_2'. Thus, we can write $s(G_1', G_2') \geq s(G_1^*, G_2^*)$, which contradicts the assumption. □

We now give an algorithm to compute an upper bound for $m(p)$. We create a new pair of genomes $G_1(p)$ and $G_2(p)$ by keeping only the genes appearing in some PSSPA in $S(p)$ and removing rest of the genes in G_1 and G_2. Those PSSPAs in $S(p)$ that can coexist wrt G_1 and G_2 can also coexist wrt $G_1(p)$ and $G_2(p)$, so that the maximum number of PSSPAs in $S(p)$ that can coexist wrt $G_1(p)$ and $G_2(p)$ is an upper bound for $m(p)$. Thus, we can apply the algorithm described in Section 3.1 to compute the upper bound: in the ILP formulation, we just need to replace G_1, G_2, and $\mathcal{P}(G_1, G_2)$ by $G_1(p)$, $G_2(p)$, and $S(p)$, respectively, and the optimal objective value of this ILP will give us an upper bound for $m(p)$. If the upper bound is no larger than $(n-1)$, then according to Lemma 4, we know that p is contained in some optimal solution. We can then safely fix p by removing other genes from gene families in $\mathcal{A}(p)$ except those in p.

Example 2. Let $G_1 = (0, -d_1, a_1, b_1, c_1, e_1, c_2, -f_1, b_2, a_2, 0)$ and $G_2 = (0, -b_3, a_3, b_4, c_2, -e_2, -b_5, c_3, d_2, f_2, 0)$. Consider the PSSS $p = \langle (a_1, b_1, c_1), (a_3, b_4, c_2) \rangle$. We have $S_1(p) = \{\langle (-d_1, e_1), (-e_2, d_2) \rangle\}$ and $S_3(p) \setminus S_2(p) = \{\langle (-d_1, b_1), (-b_5, d_2) \rangle, \langle (b_1, e_1), (-e_2, -b_5) \rangle, \langle (b_2, 0), (0, -b_3) \rangle\}$. We then create the $G_1(p)$ and $G_2(p)$ based on $S(p)$, obtaining $G_1(p) = (-d_1, b_1, e_1, b_2, 0)$ and $G_2(p) = (0, -b_3, -e_2, -b_5, d_2)$. There are at most two PSSPAs in $S(p)$ that can coexist wrt $G_1(p)$ and $G_2(p)$, i.e., $m(p) \leq 2$. Thus, p is contained in some optimal exemplars of G_1 and G_2. We can then simplify G_1 and G_2 to $G_1' = (0, -d_1, a_1, b_1, c_1, e_1, -f_1, 0)$ and $G_2' = (0, a_3, b_4, c_2, -e_2, d_2, f_2, 0)$.

For an arbitrary PSSS p, the cardinality of $S(p)$ is usually small, since it consists of only the PSSPAs related with the gene families in $\mathcal{A}(p)$ and those spanning p. Thus, we expect the corresponding ILP instance to be small and efficiently solvable. The preprocessing phase applies this algorithm iteratively. At each iteration, we list all possible *maximal* PSSSs, i.e., those PSSSs that cannot be extended into longer ones. For each of these maximal PSSSs, we use the above algorithm to test whether it is contained in some optimal solution. If it is, then we fix it and start a new iteration. If the test fails for all PSSSs, then the preprocessing phase terminates. The efficacy of this procedure is studied in Section 4.

4 Simulation Results

We simulate a pair of genomes as follows. We start from an ancestor genome with only one linear chromosome consisting of N gene families, each of which contains exactly one gene. The two extant genomes then evolve independently from this ancestor genome. The evolution process on each branch includes a series of genome-scale events: inversions, which occur with probability p, and segmental duplications, which occur with probability $(1-p)$. An inversion randomly chooses two positions on the same chromosome and then reverses the segment in-between these positions. A segmental duplication randomly chooses a segment of length L and insert its copy to another random position. We make sure that the expected number of genes per gene family is two in each genome—so that the number of events on each branch is $N/((1-p) \cdot L)$. Thus, a simulation configuration is determined by the triple (N, p, L).

First, we evaluate the efficacy of the two features (preprocessing and additional constraints) in our algorithm. The preprocessing procedure to identify and fix the optimal substructures (Section 3.3) is referred to as feature 1, while the procedure to add the inference constraints (Section 3.2) is referred to as feature 2. We denote by ALGO0 the basic ILP formulation, denote by ALGO1 (resp. ALGO2) the basic ALGO0 with feature 1 (resp. feature 2), and denote by ALGO3 the basic ALGO0 with both features. We use the configuration $(1000, p, L)$ to evaluate the 4 algorithms, where p ranges from 0 to 0.9 and $L \in \{1, 5, 10\}$. The results are shown in Table 1. Observe that, as L increases, all versions of the program take longer, because the difficulty of the problem is closely related to the density of the shared adjacencies in each genome. With $L = 1$, each duplication copies a single

gene, thus adjacencies are rarely copied; the larger L is, the more shared adjacencies appear in each genome. Similarly, observe that, as p increases, all versions of the program use less time, because the increased number of inversions destroy many shared adjacencies. While each of the features improves the performance (with feature 2 bringing more significant improvements than feature 1, especially for small p), the two features are synergistic, as their combination gains more than the sum of the individual gains. Indeed, when both features are used, the resulting algorithm is very fast, finishing all instances in a very short time.

Next, we compare our algorithm ALGO3 with the divide-and-conquer algorithm proposed by Nguyen *et al.* [12], referred to here as ALGOX. The results are shown in Table 2. ALGO3 runs very fast for small L and large p. Moreover, for those instances with larger L ($L = 5$) and smaller p ($p \leq 0.3$), ALGO3 still finishes in a very short time (roughly one minute), even for genomes with 10000 genes. ALGOX, however, can finish in 20 minutes only when both N and p are small—in this case the number of shared adjacencies in the optimal exemplars is close to N, so that ALGOX can cut off most of the branches in the search process.

5 Application to Orthology Assignment

We consider the following *generalized breakpoint distance* problem: given two genomes, select the same number of genes in each gene family in each genome and compute a bijection between the genes in the same gene family, such that the number of shared adjacencies between the resulting two genomes (without duplicate genes, since we can think of each assigned pair as belonging to its own family) is

Table 1. Comparison of the four algorithms. For each value of L, the four columns correspond to ALGO0, ALGO1, ALGO2 and ALGO3, respectively. For each configuration, we generate and solve 10 independent instances, and compute the average running time (in seconds) for the instances that are finished in 20 minutes. If the number of instances that are finished in 20 minutes is less that 10, we put this number in a parentheses. Programs are run on a 64-core (2.3GHz) machine with 512GB memory.

p	$L = 1$				$L = 5$				$L = 10$			
0.0	0	0	0	0	(0)	22(6)	169	8	(0)	145(4)	181(9)	107
0.1	0	0	0	0	(0)	39(6)	60	12	(0)	87(2)	233(8)	237
0.2	0	0	0	0	(0)	68	44	6	(0)	489(2)	184(8)	201
0.3	0	0	0	0	688(4)	62	45	4	(0)	268(2)	146(8)	189
0.4	0	0	0	0	271(8)	8	40	1	(0)	622(6)	130(8)	368
0.5	0	0	0	0	34	1	11	1	(0)	156(9)	88(9)	110
0.6	0	0	0	0	9	0	6	0	(0)	134	89(9)	36
0.7	0	0	0	0	0	0	0	0	7	8	102	6
0.8	0	0	0	0	0	0	0	0	20	1	9	1
0.9	0	0	0	0	0	0	0	0	0	0	0	0

maximized. The exemplar breakpoint distance problem is a special case of this problem, in which one selects only one gene in each gene family.

We propose a new algorithm, which we denote by IALGO3, to solve this generalized breakpoint distance problem. The algorithm works in two steps. In the first step, the algorithm starts with the two given genomes G_1 and G_2 and uses ALGO3 iteratively to compute an optimal pair of exemplars between the two current genomes; each pair of genes in the exemplars is assigned to a distinct new gene family of its own, and the updated genomes are returned. Successive iterations will decrease the size of gene families and produce new exemplar pairs. This step ends when two consecutive iterations give the same exemplars. In the final exemplars, there are $\min\{F(G_1, f), F(G_2, f)\}$ genes in each genome for each gene family $f \in \widetilde{\mathcal{A}}(G_1) \cap \widetilde{\mathcal{A}}(G_2)$. The second step then uses a local improvement strategy to optimize the objective, by iteratively examining all the gene pairs in the bijection specified by the exemplars and removing those pairs whose removal can increase the number of shared adjacencies.

The bijection infers a subset of the orthologs between the two given genomes under a parsimonious evolutionary model. We compare the results of IALGO3 with another method for orthology assignment based (at least in part) on position, MSOAR [14]. MSOAR uses several heuristics to build a bijection between genes in the same gene family such that the inversion distance induced by the bijection is minimized. We apply both methods to five well annotated species, human, gorilla, pongo, mouse, and rat. The datasets were downloaded from Ensembl (`http://www.ensembl.org`). For each species, we collected the sequences and positions on the chromosomes for all the protein-coding genes; in case a gene had multiple alternative products, we kept its longest isoform. We follow the pipeline of MSOAR to generate gene families: we first run the all-versus-all gene sequence BLASTp comparison, and then build a graph with all genes as vertices and link two genes if they are among each other's top five

Table 2. Comparison of ALGO3 and ALGOX. For each configuration, the two columns give the running time of ALGO3 and ALGOX, respectively. The experimental setup is the same as that in Table 1. The command line for the ALGOX program uses "`-ebp -fastED -delete`".

N	$L = 1$						$L = 5$					
	$p = 0.1$		$p = 0.3$		$p = 0.5$		$p = 0.1$		$p = 0.3$		$p = 0.5$	
100	0	28(3)	0	(0)	0	(0)	0	0	0	278(6)	0	(0)
200	0	(0)	0	(0)	0	(0)	2	2	0	(0)	0	(0)
500	0	(0)	0	(0)	0	(0)	4	68(2)	5	(0)	0	(0)
1000	0	(0)	0	(0)	0	(0)	7	(0)	11	(0)	1	(0)
2000	0	(0)	0	(0)	0	(0)	9	(0)	14	(0)	2	(0)
5000	0	(0)	0	(0)	0	(0)	24	(0)	30	(0)	12	(0)
10000	1	(0)	0	(0)	0	(0)	56	(0)	75	(0)	18	(0)

bidirectional best hits; we finally take all the genes in a connected component of the graph as a gene family.

We do the pairwise comparison for all five species. For each pair of species, we run both methods to obtain two bijections. We then evaluate each bijection as follows. We call a pair in a bijection *trivial* if each of the two genes in this pair forms a singleton gene family. We focus on nontrivial gene pairs in each bijection. To assess them, we downloaded the gene symbols (HGNC symbols for the primate genes, MGI symbols for mouse genes, and RGD symbols for rat genes) from Ensembl as the reference criterion. For a nontrivial gene pair g_1 and g_2 with symbols s_1 and s_2 respectively, we say it is *assessable* if there exists one gene (which could be g_2) in G_2 that has symbol s_1, or there exists one gene (which could be g_1) in G_1 that has symbol s_2, or both. We say this pair is assigned correctly, if there exists one gene (which could be g_2) with gene symbol s_1 in G_2 that is in the same tandemly arrayed genes with g_2, or there exists one gene (which could be g_1) with gene symbol s_2 in G_1 that is in the same tandemly arrayed genes with g_1, or both. The *accuracy* of a bijection is then defined as the ratio between the number of correctly assigned pairs and the number of assessable pairs.

The results are shown in Table 3. More than half of the gene pairs in the two bijections are nontrivial—duplicate genes are very common in these five species. Both methods reach very high accuracy, with IALGO3 doing slightly better than MSOAR—but IALGO3 ran very much faster (by at least two orders of magnitude) than MSOAR. We remark that the first step of the iteration in IALGO3 is to run ALGO3 on the two given genomes. Therefore, the running time of IALGO3 showed in Table 3 is an upper bound on the time to compute the optimal exemplars using ALGO3.

Table 3. Pairwise comparison of the five species. For each category, the first column shows the number for IALGO3 and the second column shows the number for MSOAR. The running time (in seconds) is measured on a 6-core (3.5GHz) machine with 32GB memory.

species pairs	gene pairs		non-trivial		assessable		accuracy		time	
gorilla & human	16457	16525	8689	8599	7812	7714	**98.2**%	98.1%	12	1472
gorilla & mouse	15381	15535	8266	8278	7030	7019	**98.1**%	97.9%	9	3509
gorilla & pongo	15211	15368	7803	7804	5733	5698	**98.1**%	**98.1**%	9	1437
gorilla & rat	15468	15611	8663	8663	6098	6078	**96.3**%	96.0%	14	6964
human & mouse	15814	15868	8266	8185	7325	7267	**98.6**%	98.3%	10	2912
human & pongo	15157	15245	7651	7609	5930	5857	**98.8**%	98.6%	6	875
human & rat	15749	15802	8613	8531	6433	6404	**96.4**%	96.3%	15	4846
mouse & pongo	14437	14509	7429	7397	5390	5374	**98.3**%	**98.3**%	8	1921
mouse & rat	17722	17862	9772	9793	6759	6768	96.7%	**96.8**%	23	5360
pongo & rat	14465	14559	7754	7740	4616	4617	**96.7**%	96.5%	9	4261

6 Conclusion

By characterizing the finer structure of the exemplar breakpoint distance problem, we were able to devise several new techniques to simplify and solve that problem, yielding an exact algorithm that runs very fast—scaling up to full mammalian genomes. We ran our algorithms both on simulations and biological datasets to verify running time, robustness, and accuracy (when used for assigning pairwise orthologies). The high speed and good scaling properties of our algorithms make them practical for research on large-scale genomic evolution, but also for improved orthology assignment, as the exemplar concept finds broad applicability in comparative genomics.

The performance of our algorithms can be further improved. We expect that additional structure can be discovered and turned into constraints for the ILP formulation, thereby reducing the search space for the ILP solver. We are also studying the use of a set of PSSSs (rather than a single PSSS) to define candidates for fixing in the optimal substructure, because it is possible that several PSSSs as a group pass the test, while any single one of these PSSSs fails.

References

1. Fertin, G., Labarre, A., Rusu, I., Tannier, E., Vialette, S.: Combinatorics of Genome Rearrangements. MIT Press (2009)
2. Hannenhalli, S., Pevzner, P.A.: Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). In: Proc. 27th Ann. ACM Symp. Theory of Comput. (STOC 1995), pp. 178–189. ACM Press, New York (1995)
3. Bader, D.A., Moret, B.M.E., Yan, M.: A fast linear-time algorithm for inversion distance with an experimental comparison. J. Comput. Biol. **8**(5), 483–491 (2001)
4. Yancopoulos, S., Attie, O., Friedberg, R.: Efficient sorting of genomic permutations by translocation, inversion and block interchange. Bioinformatics **21**(16), 3340–3346 (2005)
5. Bergeron, A., Mixtacki, J., Stoye, J.: A unifying view of genome rearrangements. In: Bücher, P., Moret, B.M.E. (eds.) WABI 2006. LNCS (LNBI), vol. 4175, pp. 163–173. Springer, Heidelberg (2006)
6. Bailey, J.A., Eichler, E.E.: Primate segmental duplications: crucibles of evolution, diversity and disease. Nature Reviews Genetics **7**(7), 552–564 (2006)
7. Lynch, M.: The Origins of Genome Architecture. Sinauer (2007)
8. Sankoff, D.: Genome rearrangement with gene families. Bioinformatics **15**(11), 909–917 (1999)
9. Bryant, D.: The complexity of calculating exemplar distances. In: Sankoff, D., Nadeau, J.H. (eds.) Comparative Genomics. Computational Biology, vol. 1 (2000)
10. Chen, Z., Fu, B., Zhu, B.: The approximability of the exemplar breakpoint distance problem. In: Cheng, S.-W., Poon, C.K. (eds.) AAIM 2006. LNCS, vol. 4041, pp. 291–302. Springer, Heidelberg (2006)
11. Blin, G., Fertin, G., Sikora, F., Vialette, S.: The Exemplar Breakpoint Distance for non-trivial genomes cannot be approximated. In: Das, S., Uehara, R. (eds.) WALCOM 2009. LNCS, vol. 5431, pp. 357–368. Springer, Heidelberg (2009)

12. Nguyen, C.T., Tay, Y.C., Zhang, L.: Divide-and-conquer approach for the exemplar breakpoint distance. Bioinformatics **21**(10), 2171–2176 (2005)
13. Angibaud, S., Fertin, G., Rusu, I., Thévenin, A., Vialette, S.: A pseudo-boolean programming approach for computing the breakpoint distance between two genomes with duplicate genes. In: Tesler, G., Durand, D. (eds.) RECMOB-CG 2007. LNCS (LNBI), vol. 4751, pp. 16–29. Springer, Heidelberg (2007)
14. Fu, Z., Chen, X., Vacic, V., Nan, P., Zhong, Y., Jiang, T.: MSOAR: A high-throughput ortholog assignment system based on genome rearrangement. Journal of Computational Biology **14**(9), 1160–1175 (2007)
15. Shao, M., Lin, Y., Moret, B.: An exact algorithm to compute the DCJ distance for genomes with duplicate genes. In: Sharan, R. (ed.) RECOMB 2014. LNCS, vol. 8394, pp. 280–292. Springer, Heidelberg (2014)