An Exact Algorithm for the Zero Exemplar Breakpoint Distance Problem

Daming Zhu and Lusheng Wang

Abstract—The exemplar breakpoint distance problem is one of the most important problems in genome comparison and has been extensively studied in the literature. The exemplar breakpoint distance problem cannot be approximated within any factor even if each gene family occurs at most twice in a genome. This is due to the fact that its decision version, the zero exemplar breakpoint distance problem where each gene family occurs at most twice in a genome (ZEBD(2, 2) for short) is NP-hard. Thus, the basic version ZEBD(2, 2) has attracted the attention of many scientists. The best existing algorithm for ZEBD(2, 2) runs in $O(n^2 \log n)$ time. In this paper, we propose a new algorithm for ZEBD(2, 2) with running time $O(n^2 1.66121^n)$. We have implemented the algorithm in Java. The software package is available upon request.

Index Terms—Exemplar breakpoint distance, genome, gene family, algorithms, time complexity

1 INTRODUCTION

Finding conserved sets of genes between two genomes is one of the fundamental problems in comparative genomics. A set of genes that is conserved in the same order in genomes suggests that they participate in the same biological process. To find conserved sets of genes in genomes, many measures have been proposed, for example, the number of breakpoints, the number of adjacencies, the number of conserved intervals, and the number of common intervals [3], [11].

All these measures are well defined when genomes do not contain duplicates. Gene duplicate is one of the primary driving forces in the evolution of genomes and occurs frequently in genomes [10]. When duplications occur, a gene may appear in several different places. Following the notations of Blin et al. [4], let $\Sigma = \{1, 2, \ldots, n\}$ be a set of $n$ integers, each represents a gene family. A genome $G$ on $\Sigma$ is a sequence of signed integers, where each integer is in $\Sigma$ and the sign represents the direction of the DNA strand. Each signed integer in a genome $G$ is referred to as a gene, which is an occurrence of a gene family.

A gene family $g$ is trivial if $g$ occurs only once in a genome $G$. Otherwise, it is nontrivial. A genome is trivial if all its gene families are trivial, and otherwise, it is nontrivial.

Given two trivial genomes $G_1 = g[1, 1], g[1, 2], \ldots, g[1, n]$ and $G_2 = g[2, 1], g[2, 2], \ldots, g[2, n]$ on $\Sigma$, where $\Sigma = \{1, 2, \ldots, n\}$ and each gene family appears exactly once in $\Sigma$, the two adjacent genes $g[1, i]$ and $g[1, i + 1]$ in $G_1$ form an adjacency in $G_1$ with respect to $G_2$, if there are two genes $g[2, j]$ and $g[2, j + 1]$ in $G_2$ satisfying (1) $g[1, i], g[1, i + 1] = g[2, j], g[2, j + 1]$ or (2) $g[1, i], g[1, i + 1] = -g[2, j + 1], -g[2, j]$. Otherwise, $g[1, i]$ and $g[1, i + 1]$ form a breakpoint in $G_1$ with respect to $G_2$. The number of breakpoints in $G_1$ with respect to $G_2$ is referred to as the breakpoint distance between $G_1$ and $G_2$, and is denoted as $d(G_1, G_2)$. It is well known that $d(G_1, G_2) = d(G_2, G_1)$ [3], [4], [11].

The breakpoint distance is not well defined for nontrivial genomes. Sankoff [14] introduced the exemplar model, which keeps only one gene for each gene family and goes back to the breakpoint distance for two trivial genomes (permutations). Given a nontrivial genome $G$ on $\Sigma$, an exemplar $E(G)$ of $G$ is a subsequence of $G$ such that it contains exactly one gene for each gene family in $G$. Given two genomes $G_1$ and $G_2$ on $\Sigma$, for the exemplar breakpoint distance problem (EBD for short), we want to find the exemplars $E(G_i)$ of $G_i$, for $i = 1, 2$ such that $d(E(G_1), E(G_2))$ is minimized. In particular, if we want to find an exemplar $E(G_i)$ of $G_i$, for $i = 1, 2$ such that $d(E(G_1), E(G_2)) = 0$, the problem is referred to as the zero exemplar breakpoint distance problem (ZEBD for short). When each gene family occurs in $G_1$ and $G_2$ at most $p$ and $q$ times, the zero exemplar breakpoint distance problem and the exemplar breakpoint distance problem are referred to as ZEBD($p, q$) and EBD($p, q$), respectively.

Sankoff [14] gave a branch and bound algorithm to solve the exemplar breakpoint distance problem. Nguyen et al. [15] proposed a divide and conquer method, and developed a program combining both the divide and conquer and the branch and bound methods. The efficiency of this approach was verified by using bacteria genomes. Bryant [6] showed that EBD is NP-Hard. In terms of inapproximability, Angibaud et al. [2] showed that EBD is Max SNP-Hard. Chen et al. [8] showed that no approximation factor can be achieved for EBD, even if each gene family occurs at most three times in a genome. Blin et al. [4] further showed that no approximation factor can be achieved for EBD, even if each gene family occurs at most twice in a genome. The reason why EBD cannot be approximated within any factor.
is due to the NP-Hardness of ZEBD(2, 2). Several exact algorithms with exponential running time have been proposed. Blin et al. used the color-coding method [1] to design an algorithm for ZEBD with \( O(n2^n) \) running time, where \( n \) is the number of gene families in \( \Sigma \) [4]. They gave a parameterized algorithm for ZEBD with \( O(m^{2s^2}) \) running time, where \( m \) is the shortest input genome size and \( s \) is the span of the input genomes. Recently, Fu and Zhang [12] developed an \( O(2^n n(O(1))) \) time algorithm for EBD(1, q), where \( n \) is the number of the gene families and \( m \) is the number of genes in the genomes. Jiang [13] proposed a polynomial algorithm for ZEBD(1, q). Moreover, Zhu [16] proved the tractability for parameterized computation of universal EBD problems. Other approaches for the exemplar breakpoint distance problem for two nontrivial genomes can be found in [3], [5], [7], [8].

In this paper, we focus on the basic version ZEBD(2, 2) and design an \( O(n^1.86121^n) \) time algorithm for ZEBD(2, 2) which improves upon the best known \( O(n2^n) \) time algorithm. Here, we propose a new approach. The key idea is to introduce the exemplar graph and develop a technique to decompose an exemplar subgraph containing two gene families into at most three subgraphs. This technique reduces the number of cases for handling two gene families from 4 to 3 and leads to a faster algorithm.

## 2 The Exemplar Graph of Two Genomes

To solve the problem of ZEBD(2, 2) for the input instance \( G_1 \) and \( G_2 \), we need to find a pair of identical exemplars for \( G_1 \) and \( G_2 \).

**Example 1.** Let \( G_1 = \{+1, -1, +2, +3, -4, +3, -5, +5 \} \) and \( G_2 = \{-2, +1, +2, -5, -3, -4, +4, +5, +1 \} \). Then, \( G_1 \) and \( G_2 \) give an instance of ZEBD(2, 2). The signed permutation \( P = +1, +2, +3, -4, +5 \) is an exemplar of both \( G_1 \) and \( G_2 \).

To decide if a pair of identical exemplars exists for the two input nontrivial genomes \( G_1 \) and \( G_2 \), we construct a vertex-labeled bipartite graph \( B(G_1, G_2) \) from \( G_1 \) and \( G_2 \). The bipartite graph contains two ordered sets of vertices \( X \) and \( Y \) corresponding to \( G_1 \) and \( G_2 \), respectively. For each gene in \( G_1 \) (respectively, \( G_2 \)), there is a vertex labeled with the gene (signed integer) in \( X \) (respectively, \( Y \)). There is a linear order for the vertices in \( X \) (respectively, \( Y \)) according to the linear order of the genes in the genome. There is an edge between a vertex in \( X \) and a vertex in \( Y \), if the two vertices are labeled with the same signed integer. Such an edge is referred to as \( +i \)-labeled or \( -i \)-labeled. We also say that an edge is \( i \)-labeled when the sign is ignored. The obtained bipartite graph \( B(G_1, G_2) \) is called the exemplar graph of \( G_1 \) and \( G_2 \).

Formally, let \( G_1 = [g_1, 1], [g_1, 2], \ldots, [g_1, m_1] \) and \( G_2 = [g_2, 1], [g_2, 2], \ldots, [g_2, m_2] \) be two genomes on \( \Sigma = \{1, 2, \ldots, n\} \). Then, the bipartite graph is constructed as \( B(G_1, G_2) = (X, Y, E) \), where \( X = \{ [x[i], 1], [x[i], 2], \ldots, [x[i], m_1] \} \) and \( Y = \{ [y[j], 1], [y[j], 2], \ldots, [y[j], m_2] \} \) are the two ordered sets of vertices such that \( x[i] \) is labeled with \([g_1, i] \) and \( y[j] \) is labeled with \([g_2, j] \), and \( E = \{ [x[i], y[j]] | [g_1, i] = [g_2, j], 1 \leq i \leq m_1, 1 \leq j \leq m_2 \} \) is its set of edges.

**Example 2.** The exemplar graph of \( G_1 \) and \( G_2 \) in Example 1 is given in Fig. 1.

Let \( e_1 = (x[i_1], y[j_1]) \) and \( e_2 = (x[i_2], y[j_2]) \) be two edges in the exemplar graph. We say that \( e_1 \) and \( e_2 \) are compatible if \( i_1 < i_2 \) and \( j_1 < j_2 \), or \( i_1 > i_2 \) and \( j_1 > j_2 \). Otherwise, we say that \( e_1 \) and \( e_2 \) are in conflict.

A vertex with label \( +i \) (or \( -i \)) is referred to as \( +i \)-labeled (or \( -i \)-labeled). A vertex is also called \( i \)-labeled, if it the sign is ignored.

A set of \( n \) edges in \( B(G_1, G_2) \) is an exemplar set, if all the \( n \) edges are mutually compatible and each of the \( n \) edges is labeled with a distinct gene family in \( \Sigma = \{1, 2, \ldots, n\} \). Once the exemplar graph \( B(G_1, G_2) \) is constructed, the task of finding a pair of identical exemplars for the two input genomes becomes finding an exemplar set in \( B(G_1, G_2) \).

**Proposition 1.** Two genomes \( G_1 \) and \( G_2 \) have a pair of identical exemplars, if and only if \( B(G_1, G_2) \) has an exemplar set.

In the next section, we describe a special kind of exemplar graphs and show that for this kind of exemplar graphs there is a polynomial time algorithm to find an exemplar set.

## 3 Finding the Exemplar Set in Simple Exemplar Graphs

Let \( G_1 \) and \( G_2 \) be the two input genomes of ZEBD(2, 2), and \( B = B(G_1, G_2) = (X, Y, E) \) the exemplar graph of \( G_1 \) and \( G_2 \). An \( i \)-labeled component \( B[i] \) of \( B \) is a subgraph of \( B \) derived from all the \( i \)-labeled vertices. When the name of the gene family is not important, we simply use a component to represent a subgraph of \( B \) containing all the vertices of a gene family.

An exemplar graph is simple, if for each \( i \in \Sigma \) the \( i \)-labeled component in the graph has at most two edges. In this section, we will work on the simple exemplar graphs and will design a polynomial time algorithm to decide if a simple exemplar graph has an exemplar set. The algorithm starts with the following observations.

**Lemma 2.** If a \( j \)-labeled edge is in conflict with all the \( i \)-labeled edges for some \( i \neq j \), then this \( j \)-labeled edge cannot belong to any exemplar set.

**Proof.** If the exemplar set contains a \( j \)-labeled edge which is in conflict with all the \( i \)-labeled edges, then no \( i \)-labeled edge can be in the exemplar set. Since an exemplar set must contain \( n \) edges, one for each gene family in \( \Sigma = \{1, 2, \ldots, n\} \), this is a contradiction. \( \square \)
A j-labeled edge is useless in an exemplar graph, if it is in conflict with all the i-labeled edges in the exemplar graph for some i ≠ j. An i-labeled component is trivial if it contains exactly one edge. If an edge is in conflict with the unique edge of a trivial component, then such an edge is useless.

Our algorithm starts with deleting useless edges in the exemplar graphs. New useless edges may occur after some useless edges are deleted.

**Example 3.** Let us consider the exemplar graph shown in Fig. 2. Edge e is useless since it is in conflict with the unique edge for gene family 3 and e’ is currently not useless since it is in conflict with one of the two +2-labeled edges. After deleting the useless edge e, e’ becomes useless since it is in conflict with the unique +2-labeled edge.

The following lemma can be used to decide which edges will become useless after a useless edge is deleted.

**Lemma 3.** An i-labeled edge e₁ which is not useless in a simple exemplar graph becomes useless after deleting a j-labeled edge e for j ≠ i, if and only if there are two j-labeled edges e and e’, and e₁ is compatible with e and in conflict with e’ in the simple exemplar graph.

**Proof.** Let B be the simple exemplar graph.

(i) Let e and e’ be the two j-labeled edges in B. Then e₁ must be compatible with e and in conflict with e’. After e is deleted, e’ becomes the unique j-labeled edge. Thus, e₁ becomes useless when e is deleted.

(only if) If e is the unique j-labeled edge, and e₁ is not useless in B, then e₁ is also not useless when e is deleted, a contradiction. Thus, there are two j-labeled edges in B. Let e and e’ be the two j-labeled edges. The edge e₁ must be in conflict with e’, because it becomes useless when e is deleted. The edge e₁ is compatible with e, because it is not useless in B.

By Lemma 3, we can repeatedly remove the useless edges in an exemplar graph. The algorithm is given as RemoveUseless(B) in Fig. 3, where the set U is used to save the useless edges obtained by Lemmas 2 and 3. Moreover, the set E(B) and E[j] are used to save all the edges and all the j-labeled edges of B, respectively.

Let us look at the running time of RemoveUseless(•). It requires O(n²) time to find all pairs of conflict edges. It takes O(n) time to decide if an edge is useless. Thus, the initial set U of useless edges can be computed in O(n²) time. Each while loop takes O(n) time to delete one edge and add new useless edges to U. If the while loop is repeated k times totally, the running time to delete useless edges is O(kn).

1. Find all the pairs of edges that are in conflict;
2. $U \leftarrow \{e \mid e \text{ is i-labeled and in conflict with every edge in } E[j] \text{ for some } j \neq i\}$;
3. While($U \neq \emptyset$)
   4. Let $e \in U$ be j-labeled for some $j$; $U \leftarrow U - \{e\}$;
   5. $E(B) \leftarrow E(B) - \{e\}$; $E[j] \leftarrow E[j] - \{e\}$;
   6. For each edge $e_j \in E(B) - E[j]$
      7. If ($e_j$ is in conflict with every edge in $E[j]$)
         then $U \leftarrow U \cup \{e_j\}$;
9. End for
10. End while
11. Return $B$

The while loop can be repeated by at most $2n$ times, because at most $2n$ edges can be deleted in a simple exemplar graph. Thus, the time complexity of RemoveUseless(•) is $O(n^2)$.

A useless edge in B does not belong to any exemplar set of B.

A simple i-labeled component can have at most two j-labeled edges in B. Let $I[B,0], I[B,1]$, and $I[B,2]$ be the sets of components in B with 0, 1, and 2 edges, respectively.

**Lemma 4.** Let $B'$ be returned by RemoveUseless(B). If $I[B',0] \neq \emptyset$, then no exemplar set of B exists. If $I[B',1] \neq \emptyset$, any edge of the components in $I[B',1]$ is compatible with the rest of edges in $B'$.

**Proof.** If $I[B',0] \neq \emptyset$, then no edge of a component in $I[B',0]$ can belong to an exemplar set of $B'$. That is to say, $B'$ has no exemplar set. Since all the edges in B that are not in $B'$ are useless edges (which cannot be in an exemplar set), B has no exemplar set.

If $I[B',1] \neq \emptyset$, let $B[j] \in I[B',1]$ and e₁ be an j-labeled edge in $B[j]$. If e₁ is in conflict with another edge e₂ in $B'$, then e₂ must be useless in $B'$, which should have been removed from $B'$ by RemoveUseless(B). Thus, e₁ is compatible with all the edges in $B'$.

If RemoveUseless(B) returns a graph $B'$ with $I[B',0] = \emptyset$ and $I[B',2] = \emptyset$, then the edges in $B'$ form an exemplar set of B by Lemma 4. If $I[B',0] = \emptyset$ and $I[B',2] = \emptyset$, we need to further test if $B'$ has an exemplar set.

A simple exemplar graph B is hopeful if $I[B,0] = \emptyset$ and there is no useless edge in B. An edge of an i-labeled component in $I[B,1]$ must be compatible with all the other edges in a hopeful exemplar graph.

Let B be a hopeful exemplar graph, e₁ and e₂ be the two distinct i-labeled edges in B. To find an exemplar of B, we need to choose one of e₁ and e₂.

Let $B_1 = \text{RemoveUseless}(B \setminus e_2), B_2 = \text{RemoveUseless}(B \setminus e_1)$, where $B \setminus e_1$ and $B \setminus e_2$ are the subgraphs of B obtained by deleting e₁ and e₂ in B, respectively. If an exemplar set of B contains e₁ (e₂), it must be an exemplar set of $B_1$ ($B_2$). Thus, if $I[B,0] = \emptyset$, e₁ must be the unique i-labeled edge in $B_1$. On the other hand, if $I[B,2] = \emptyset$, e₂ must be the unique i-labeled edge in $B_2$. The key idea here is to test if $I[B,0] = \emptyset$ or $I[B,2] = \emptyset$ and choose an edge eᵢ for $i = 1, 2$ such that $I[B_i,0] = \emptyset$. Most importantly, we will show that if both $I[B,0] = \emptyset$ and $I[B,2] = \emptyset$, we can arbitrarily choose one of them, say, e₁.
Example 4. Let us consider a hopeful exemplar graph shown in Fig. 4a, where there are two +3-labeled edges e1 and e2 with I[B1, 0] = φ and I[B2, 0] = φ. There are two +1-labeled edge and two +2-labeled edge. The +1-labeled (respectively, +2-labeled) edge which is in conflict with e1 must be compatible with e2 (otherwise, such an edge is useless) and the other +1-labeled (respectively, +2-labeled) edge which is in conflict with e2 must be compatible with e1 (otherwise, such an edge is useless).

Figs. 4b and 4c show the two exemplar sets containing e1 and e2, respectively. The exemplar set shown in Fig. 4c can be obtained from the exemplar set shown in Fig. 4b by deleting edge e1 and all edges in conflict with e2 (the +1-labeled edge and the +2-labeled edge in Fig. 4b) and adding edge e2 and all the edges in conflict with e1 (the +1-labeled edge and the +2-labeled edge in Fig. 4c).

The reason that we can do this is due to the fact that in such a hopeful exemplar graph B if there is an i-labeled edge (i ≠ 3) which is in conflict with e1, then there must be the other i-labeled edge which is compatible with e1. (If there is no other i-labeled edge, I[B1, 0] ≠ φ. If the other i-labeled edge which is in conflict with e1, then e1 is useless.) Similarly, if there is an i-labeled edge which is in conflict with e2, then there must be the other i-labeled edge which is compatible with e2 since I[B1, 0] = φ. Therefore, I[B1, 0] = φ and I[B2, 0] = φ ensure that choosing either e1 or e2 can give an exemplar set if one exists.

Proposition 5. Let B be a hopeful exemplar graph with two i-labeled edges e1 and e2 such that I[B1, 0] = φ and I[B2, 0] = φ. If there is a j-labeled edge (j ≠ i) which is in conflict with e1, then there must be the other j-labeled edge which is compatible with e1. Similarly, if there is a j-labeled edge which is in conflict with e2, then there must be the other j-labeled edge which is compatible with e2.

Based on the discussion in Example 4, let B be a hopeful exemplar graph and e1 and e2 the two i-labeled edges in B with I[B1, 0] = φ and I[B2, 0] = φ. Given an exemplar set of B containing edge e1, one can obtain another exemplar set containing e2 by deleting edge e1 and all edges in conflict with e2 and adding edge e2 and all the edges in conflict with e1.

Lemma 6. Let B be a hopeful exemplar graph. Let e1 and e2 be the two i-labeled edges in B with I[B1, 0] = φ. B has an exemplar set if and only if B1 has an exemplar set.

Proof. (if)B1 is obtained from B by deleting some edges. Thus, any edge in B1 is also in B. We can immediately conclude that if B1 has an exemplar set, then B also has an exemplar set.

(only if) Let Ex(B) be an exemplar set of B containing e2. If I[B1, 1] ≠ φ, every edge in a trivial component of B1 is compatible with the rest of edges in B1 by Lemma 4. Moreover, since B1 is obtained from B by deleting some edges, then every simple component in B1 is also a simple component in B. Thus, we have I[B1, 2] ⊆ I[B, 2]. Therefore, we can construct an exemplar set Ex(B1) for B1 as follows: 1) for any trivial component in B1, Ex(B1) contains the unique edge of it, 2) for any simple component in B1, Ex(B1) contains the edge of the component which is also in Ex(B). Since I[B1, 0] = φ, Ex(B1) has n mutually compatible edges, each of which is labeled with a distinct gene family. Thus, Ex(B1) is an exemplar set of B1.

An alternative way to look at this is as follows: Let Ex(B) be an exemplar set of B containing e2. We can obtain a new exemplar set of B by deleting edge e2 and all edges in conflict with e1 and adding edge e1 and all the edges in conflict with e2 based on Proposition 5.

The algorithm for testing if an exemplar set exists for a simple exemplar graph is given as Simple-Solver(B) in Fig. 5. A candidate exemplar set C is maintained in the computation of the algorithm. All the edges in C come from the trivial components in I1. In Step 6, the subroutine RemoveUseless1(B \ e) is a slight modification of RemoveUseless(B \ e), where U contains the set of those edges which are in conflict with the i-labeled edge other than e (instead of the original Steps 1 and 2 in RemoveUseless(●)).

To compute I0 = I[B, 0], I1 = I[B, 1], and I2 = I[B, 2], we can look at each i-labeled component in B one by one. Since each component contains at most two edges in a simple exemplar graph B, it takes O(n) time to compute each I0 = I[B, 0], I1 = I[B, 1], and I2 = I[B, 2]. The running time of RemoveUseless1(●) is O(km) if k useless edges have been removed in the execution of it. There are at most 2n edges in a simple exemplar graph B. During the whole process, there are at most n out of 2n useless edges will be removed. Thus, the total running time for RemoveUseless1(●) is O(n^2).

For the same reason, the white loop in Steps 4~10 will be repeated by at most n times. Thus, the total time to compute I0 = I[B, 0], I1 = I[B, 1], and I2 = I[B, 2] is also O(n^2).
Therefore, the time complexity of Simple-Solver($B$) is $O(n^2)$. In summary, we have the following:

**Theorem 7.** The algorithm Simple-Solver($B$) returns an exemplar set of $B(G_1, G_2)$ if one exists. Otherwise, Simple-Solver($B$) returns the empty set. The running time of Simple-Solver($B$) is $O(n^2)$ time.

Using Simpler-Solver($\bullet$), an algorithm for the general case of ZEBD($2,2$) can be derived as follows: For ZEBD($2,2$), each $j$-labeled component has a set of edges $E[j]$ containing at most four edges. In the worst case, $E[j]$ can be arbitrarily partitioned into edge disjoint sets $E[j,1]$ and $E[j,2]$, each of which contains at most two edges. For every $j \in \Sigma$, either $E[j,1]$ or $E[j,2]$, we can obtain at most $2^n$ instances of simple exemplar graphs. Using Simple-Solver($\bullet$), we can solve all the $2^n$ instances of simple exemplar graphs in time $O(n^2 2^n)$.

In the next section, we will design a faster exact algorithm for ZEBD($2,2$) by looking at two components (for two different gene families) at a time.

### 4 The 1.86121n Time Algorithm for ZEBD($2,2$)

The basic idea of getting an exact algorithm with running time better than $O(n^2 2^n)$ is as follows: We look at two components at a time. We can decompose the exemplar subgraph induced by a pair of components into at most three simple subgraphs, each of which has no conflict edges. In this way, we can get about $2^{0.5n}$ instances of simple exemplar graphs and use the algorithm designed in Section 3 to find solutions of the simple exemplar graphs.

Note that a component cannot have three edges. A component is complex if it has four edges. Two components in $B$ as a whole are referred to as a component pair if they are both complex.

Let $B = (X,Y,E)$ be an exemplar graph for the two input genomes. Consider a component pair induced by the $i$-labeled component $B[i]$ and the $j$-labeled component $B[j]$. Each genome contains two genes from each of the two gene families $i$ and $j$. The order of the four genes can be one of the following six patterns: $Q_1 = [i,i,j,j], Q_2 = [i,j,j,i], Q_3 = [i,j,i,j], Q_4 = [j,i,j,i], Q_5 = [j,i,i,j], Q_6 = [j,j,i,j]$. Thus, there are totally 36 possible different subgraphs of $B$ induced by $B[i]$ and $B[j]$. These 36 possible subgraphs are shown in Fig. 6.

For convenience, we organize the 36 subgraphs in a $6 \times 6$ matrix $M$, where the six rows and the six columns correspond to $Q_1, Q_2, ..., Q_6$, respectively.

In particular, it is worth to note that an exemplar graph containing $M[1,6]$ or $M[6,1]$ does not have any exemplar set and, thus, will not be considered in the decomposition. Also for $M[1,1]$ or $M[6,6]$, any $i$-labeled edge is always compatible with all the $j$-labeled edges. Thus, we do not consider to decompose $M[1,1]$ and $M[6,6]$, either. A component pair is nontrivial, if it is one of the rest 32 combinations.

#### 4.1 Decomposition of a Nontrivial Component Pair

Let $B[i]$ and $B[j]$ be two complex components forming a nontrivial component pair in $B$. We use $B[i] \cup B[j]$ to represent the subgraph of $B$ containing all the vertices and edges of $B[i]$ and $B[j]$. We will decompose the subgraph $B[i] \cup B[j]$ into at most three simple subgraphs, such that (C1) each pair of $i$-labeled and $j$-labeled compatible edges appears at least once in the set of at most three simple subgraphs.

Note that the two edges in an exemplar set for gene families $i$ and $j$ are always compatible. Thus, condition (C1) will ensure that all possible solutions will be included in the decomposition. A set of subgraphs for $B[i] \cup B[j]$ is safe if condition (C1) holds.

**Decomposition of the 32 subgraphs.** We classify the 32 combinations of $B[i] \cup B[j]$ into eight types, so that each type of combinations can be decomposed into a set of at most three simple subgraphs such that condition (C1) holds.

Type 1. $M[1,2]$ shown in Fig. 7a can be decomposed into a set of three subgraphs shown in Figs. 7b, 7c, and 7d, respectively. Every pair of $i$-labeled and $j$-labeled compatible edges in $M[1,2]$ appears in the subgraphs in Figs. 7b,
Moreover, the structures of Figs. 7b and 7c, respectively. Moreover, the structures of Figs. 8b and 8c, respectively. Every pair of subgraphs shown in Figs. 9b and 9c, respectively. Every pair of subgraphs in Figs. 10b, 10c, and 10d, respectively. Every pair of subgraphs shown in Figs. 11b and 11c, respectively. M[2, 6] can be treated in the same way.

Type 5. M[2, 3] in Fig. 11a can be decomposed into a set of two subgraphs such that condition (C1) holds as shown in Figs. 12b and 12c, respectively. Moreover, M[4, 2] can be treated in the same way.

Type 6. M[2, 5] in Fig. 12a can be decomposed into a set of two subgraphs such that condition (C1) holds as shown in Figs. 12b and 12c, respectively. Moreover, M[5, 2] can be treated in the same way.

Type 7. M[3, 3] in Fig. 13a can be decomposed into a safe set of two subgraphs in Figs. 13b and 13c, respectively. Moreover, M[4, 4] can be treated in the same way.

Type 8. M[3, 4] in Fig. 14a can be decomposed into a set of two subgraphs such that (C1) holds as shown in Figs. 14b and 14c, respectively. Moreover, M[4, 3] can be treated in the same way.

Now, all the 32 cases have been covered.

4.2 Decomposition of an Exemplar Graph

Let B be the exemplar graph of the two input genomes. Now, we try to decompose B into a set of simple exemplar graphs by looking at nontrivial component pairs.

Finding a set of nontrivial component pairs. There are many ways to find nontrivial component pairs. The selection of one nontrivial component pair will affect the selection of the other nontrivial component pair. For example, it is possible to find three gene families i, j, and k such that both B[i]∪B[j] and B[i]∪B[k] can form two nontrivial component pairs. However, once the nontrivial component pair B[i]∪B[j] is used for decomposition, B[i]∪B[k] cannot be used anymore.

Our strategy is to look at all pairs of gene families in an arbitrary order and decompose the nontrivial component pairs once they are found. The details is as follows:

1. Look at a pair of labels i and j in Σ at a time.
2. Test if B[i]∪B[j] is M[1, 6] or M[6, 1]. If yes, there is no exemplar set for B and stop the whole process.
3. Test if B[i]∪B[j] is a nontrivial component pair.
4. If yes, we decompose B[i]∪B[j] into a safe set B[i, j] of at most three simple subgraphs, and save B[i, j] to S.
5. Repeat 1-4 until there is no nontrivial component pair.
The above procedure is referred to as $\text{Decomposition}(B)$. It takes $O(1)$ time to test if $B[i] \cup B[j]$ is nontrivial. Since there are $O(n^2)$ pairs of gene families, $\text{Decomposition}(B)$ requires $O(n^2)$ time.

In practice, we can first try to find nontrivial component pairs of type 3 with one simple subgraph in the safe set of subgraphs, then try to find nontrivial component pairs of types with two simple subgraphs in the safe set of subgraphs, and finally find nontrivial component pairs of types with three simple subgraphs in the safe set of subgraphs. In this way, the algorithm should be faster in practice.

$\text{Decomposition}$ of $B$. Let $S$ be the set of safe sets obtained in $\text{Decomposition}(B)$. Since each safe set $T[i,j]$ is in $S$ at most three simple subgraphs, we can decompose $B$ into a set of at most $3|S|$ so-called decomposed exemplar graphs. A decomposed exemplar graph from $S$ is obtained from $B$, by selecting a simple subgraph in each of the $|S|$ safe sets, and replacing the corresponding nontrivial component pair with the selected simple subgraph. Recall that our original target is to decompose $B$ into a set of simple exemplar graphs. Unfortunately, the decomposed exemplar graphs obtained from $S$ may not be simple since there may exist component pairs of type $M[1,1]$ and $M[6,6]$. (We only consider 32 out of 36 cases for decomposition.) It should be emphasized that each decomposed exemplar graph does not contain any nontrivial component pair.

Lemma 8. The exemplar graph $B$ has an exemplar set, if and only if a decomposed exemplar graph obtained from $S$ has an exemplar set.

Proof. (if) Let $B'$ be a decomposed exemplar graph obtained from $S$. Since any edge in $B'$ is also in $B$, if $B'$ has an exemplar set, then $B$ has the exemplar set, too.

(only if) Let $B$ have an exemplar set $E_x(B)$. In any safe set $T[i,j]$ in $S$, there must exist a simple subgraph which contains an $i$-labeled and a $j$-labeled edge in $E_x(B)$. Thus, such kind of subgraphs can be used to form a decomposed exemplar graph $B'$ containing all the edges in $E_x(B)$.

Since the decomposed exemplar graphs are not always simple, we cannot directly use the algorithm in Section 3 to solve the problem for the decomposed exemplar graphs. We need some other method to handle this.

4.3 The 1.86121$^n$ Time Algorithm

The decomposed exemplar graphs may contain component pairs of type $M[1,1]$ or $M[6,6]$. Two components are compatible, if all the edges of one component are compatible with all the edges of the other component.

Now, we consider another kind of exemplar graphs. An exemplar graph is (4,1), if every component in the graph is either complex or trivial, and all complex components are compatible with each other.

Lemma 9. There is an exact algorithm for (4,1) exemplar graphs with running time $O(n^2)$.

Proof. Let $B'$ be a (4,1) exemplar graph. We can use $\text{RemoveUseless}(B')$ to solve it. If $\text{RemoveUseless}(B')$ returns a graph $B''$ with $I[B'',0] \neq \emptyset$, then $B'$ has no exemplar set. Otherwise, we can obtain an exemplar set of $B'$ by arbitrarily selecting an $i$-labeled edge in $B''$ for each $i$. The reason is that for any pair of edges labeled with $i$ and $j$ for $i \neq j$, they are compatible in $B''$. The running time of $\text{RemoveUseless}(B')$ is $O(n^2)$.

Further decomposition of the decomposed exemplar graphs into (4,1) exemplar graphs. Let $B$ be the exemplar graph of the two input genomes. Let $r$ be the number of simple components in $B$. $S$ be the set of safe sets obtained by $\text{Decomposition}(B)$.

Lemma 10. Let $B'$ be a decomposed exemplar graph obtained from $S$. Then $B'$ can be further decomposed into a set of at most $2^{2|S|+r}(4,1)$ exemplar graphs such that $B'$ has an exemplar set if and only if one of the (4,1) exemplar graphs has an exemplar set.

Proof. There are $2|S|+r$ simple components in $B'$. Every pair of complex components in $B'$ must be compatible with each other by the decomposition process $\text{Decomposition}(B)$. We can delete one edge in every simple component of $B'$, and obtain a (4,1) exemplar graph of $B'$. This method can produce at most $2^{2|S|+r}(4,1)$ exemplar graphs. It is clear that $B'$ has an exemplar set, if and only if one of the (4,1) exemplar graphs has an exemplar set.

By Lemmas 8, 9, and 10, we can design an exact algorithm with running time $O(n^23|S|2^{2|S|+r})$ as shown in Fig. 15. The algorithm is referred to as Algorithm 1.

Further Decomposition of the decomposed exemplar graphs into simple exemplar graphs. Alternatively, a decomposed exemplar graph $B'$ from $S$ can be further decomposed into $2n-2|S|-r$ simple graphs.

Lemma 11. Let $r$ be the number of simple components in $B$. $S$ be the set of safe sets obtained by $\text{Decomposition}(B)$. Let $B'$ be a decomposed exemplar graph obtained from $S$. Then, a set of at most $2n-2|S|-r$ simple exemplar graphs exists such that $B'$ has an exemplar set if and only if one of the simple exemplar graphs has an exemplar set.

Proof. There are at most $n-2|S|-r$ complex components (with four edges) in $B'$. We split each complex component into two parts, each of which contains two edges. At most $n-2|S|-r$ simple components can be obtained by selecting one of the two parts. These simple components together with all the other components of $B'$ can be reassembled into a simple exemplar graph of $B'$. At most $2n-2|S|-r$ such simple exemplar graphs can be
1. Use Decomposition($B$) to get a set $S$ of safe sets.
2. For each decomposed exemplar graph $B'$ obtained from $S$, do Step 3.
3. For each simple graph obtained from $B'$ by the method in Lemma 11, use Simple-Solver($\bullet$) to solve it in $O(n^2)$ time. If the simple exemplar graph has an exemplar set, return the exemplar set and stop.
4. Otherwise, there is no exemplar set for $B$ and stop.

Fig. 16. Algorithm 2: The algorithm for ZEBD(2,2) by solving simple exemplar graphs.

By Lemmas 8 and 11 and Theorem 7, we can also design an $O(n^23^[2\Sigma-2]-[r]-r)$ time algorithm to solve ZEBD(2,2) for the input exemplar graph $B$ as in Fig. 16. The algorithm is referred to as Algorithm 2.

The whole algorithm. The whole algorithm is as follows: If $2|S|+r \geq n-2|S|-r$, we use Algorithm 2. Otherwise, we use Algorithm 1. The following theorem gives the running time.

Theorem 12. There is an $O(n^21.86121^n)$ time algorithm to solve ZEBD(2,2).

Proof. The exemplar graph $B$ for ZEBD(2,2) can be constructed in $O(n^2)$ time.

If $2|S|+r \geq n-2|S|-r$, then $2|S|+r \geq \frac{n}{2}$. We use Algorithm 2 to find the exemplar set of $B$. The running time is

$$O(n^23^[2|S|-2]-|S|-r) = O(n^23^[n/2]) = O(n^21.86121^n).$$

If $2|S|+r < n-2|S|-r$, then $2|S|+r < \frac{n}{2}$. We use Algorithm 1 to find the exemplar set of $B$. The running time is

$$O(n^23^[2|S|+r]) = O(n^23^[n/4]) = O(n^21.86121^n).$$

Once an exemplar set is obtained, the identical exemplars of $G_1$ and $G_2$ can be obtained accordingly. □

It takes $O(n)$ space to save the exemplar graph of two genomes. It takes $O(n^2)$ space to save all the pairs of conflict edges in the exemplar graph. The set $S$ of safe sets for nontrivial component pairs requires $O(n)$ space. Thus, the space complexity of our algorithm is $O(n^2)$.

5 Simulation Experiments

We have implemented our algorithm in Java. A gene is trivial (nontrivial) if it belongs to a trivial (nontrivial) gene family. A nontrivial gene family is pegged if there is at least one trivial gene between any two genes of the family. Otherwise, the nontrivial gene family is unpegged. A genome is pegged if all the nontrivial gene families are pegged. Given two genomes $G_1$ and $G_2$ on $\Sigma$, a gene family is referred to as a single-pair family if it is trivial in both $G_1$ and $G_2$. A gene family is a multipair family if it is not a single-pair family. In practice, lots of gene families in genomes are pegged [14]. Sankoff [14] gave a procedure to produce the pegged genomes based on a given configuration of multipair families, and tested the computational costs of his branch and bound approaches. We set the configuration as ZEBD(2,2) and use the same procedure to generate our data for experiments.

The real running time to compute the case where all nontrivial gene families are pegged is shown in Fig. 17. We can show that our program can output results within 0.25 seconds for genomes with 500 gene families, where at least $3/4$ of them are multipair families. This implies that the pegged genomes for ZEBD(2,2) case is very easy to solve.

Now, let us look at more difficult cases, where we allow the 80, 60, 40, 20, and 0 percent of gene families to be pegged.

Recall that the total running time of our algorithm is $O(n^21.86121^n)$, where $O(n^2)$ is the running time for solving a (4,1) exemplar graph (we denote such a subroutine as By(4,1)($\bullet$)) or the running time for solving a simple exemplar graph (Simple-Solver($\bullet$)). Similar to that in [14],
we use the number of times that the subroutine By(4, 1) (●) or Simple-solver (●) are called to represent the computational costs of the algorithm. Thus, the results will not be related to a specific computer. The results are shown in Fig. 18. Here, we set one million as the upper bound for the number of times that By(4, 1) (●) or Simple-Solver (●) are called. The vertical axis represents the logarithm of the number of calls for By(4, 1) (●) or Simple-Solver (●). From Fig. 18, we can see that within one million calls of By(4, 1) (●) or Simple-Solver (●), our algorithm can solve the instances with 40 unpegged multipair gene families. We also observed that the algorithm can solve the instances with 170 multipair gene families, while 80 percent of them are pegged.

6 Conclusion

In this paper, we presented an exact algorithms for ZEBD(2, 2) with running time $O(n^2 1.86121^n)$. We have implemented the algorithm in Java. Experiments show that the algorithm can handle pegged genomes with large number of gene families. When the number of nonpegged gene families increases, the running time of our algorithm increases dramatically. Our key idea for the $O(n^2 1.86121^n)$ exact algorithm is to look at a pair of complex gene families at a time. It is interesting to know if one can get faster algorithms by looking at more than two gene families in the exemplar graph.

Acknowledgments

The authors would like to thank reviewers for their helpful suggestions. In particular, reviewer 1 suggests that the algorithm will be faster in practice if priorities are given to different types of component pairs in Decomposition (●). They would also like to thank Renjie Li for helping them implement the algorithm. This work was supported by the Science Foundation of Shandong Province, [ZR2012Z002], National Science Foundation of China 61070019, and a grant from the Research Grants Council of the Hong Kong Special Administrative Region [Project No. CityU 123013].

References