Approximating the edit distance for genomes with duplicate genes under DCJ, insertion and deletion

Mingfu Shao*, Yu Lin*

From Tenth Annual Research in Computational Molecular Biology (RECOMB) Satellite Workshop on Comparative Genomics
Niterói, Brazil. 17-19 October 2012

Abstract
Computing the edit distance between two genomes under certain operations is a basic problem in the study of genome evolution. The double-cut-and-join (DCJ) model has formed the basis for most algorithmic research on rearrangements over the last few years. The edit distance under the DCJ model can be easily computed for genomes without duplicate genes. In this paper, we study the edit distance for genomes with duplicate genes under a model that includes DCJ operations, insertions and deletions. We prove that computing the edit distance is equivalent to finding the optimal cycle decomposition of the corresponding adjacency graph, and give an approximation algorithm with an approximation ratio of $1.5 + \frac{1}{L}$.

Introduction
The combinatorics and algorithmics of genomic rearrangements have been the subject of much research since the problem was formulated in the 1990s [1]. The advent of whole-genome sequencing has provided us with masses of data on which to study genomic rearrangements and has motivated further work. Genomic rearrangements include inversions, transpositions, block exchanges, circularizations, and linearizations, all of which act on a single chromosome, and translocations, fusions, and fissions, which act on two chromosomes. These operations are all implemented in terms of the single double-cut-and-join (DCJ) operation [2,3], which has formed the basis for much algorithmic research on rearrangements over the last few years [4-7]. A DCJ operation makes two cuts in the genome, either in the same chromosome or in two different chromosomes, producing four cut ends, then rejoins the four cut ends.

A basic problem in genome rearrangements is to compute the edit distance, i.e., the minimum number of operations needed to transform one genome into another. For unichromosomal genomes, Hannenhalli and Pevzner gave the first polynomial-time algorithm to compute the edit distance under signed inversions [8], which was later improved to linear time [9]. For multichromosomal genomes, the edit distance under the Hannenhalli-Pevzner model (signed inversions and translocations) has been studied through a series of papers [8,10-12], culminating in a fairly complex linear-time algorithm [4]; under DCJ operations, the edit distance can be computed in linear time in a simple and elegant way [2].

All of the above algorithms for computing edit distances assume equal gene content and no duplicate genes. El-Mabrouk [13] first extended the results of Hannenhalli and Pevzner to compute the edit distance for inversions and deletions. Chen et al. [14] studied the problem of computing the inversion distance for genomes with equal gene content in the presence of duplicate genes—a problem that comes up in determining orthologies, where greedy heuristics were used. Yancopoulous et al. [7] proposed some rules on how to incorporate insertions and deletions into the DCJ model, but no specific algorithms are given. Braga et al. [15] presented a linear-time algorithm to compute the edit distance for DCJ operations,
insertions and deletions, but still without duplications. Sébastien Angibaud et al. [16,17] studied several model-free measures between genomes with duplicate genes; they first established a one-to-one correspondence between genes of both genomes, and then computed the measure between two genomes without duplicate genes.

In this paper, we focus on the problem of computing the edit distance between two genomes in the presence of duplications. We define the edit distance at the adjacency set level on a unit-cost model including DCJ operations, insertions and deletions (duplications are a special case of insertions). We reduce the problem of computing such an edit distance to finding the maximum number of certain cycles in the adjacency graph. Finally we give a (1.5 + 1)-approximation algorithm.

**Edit distance**

We represent the genomes using the notations introduced by Bergeron et al. [2]. Denote each gene \(g\) with its two extremities, the head as \(g_h\) and the tail as \(g_t\). Two consecutive genes \(a\) and \(b\) can be connected by one adjacency, which is represented by a pair of extremities; thus adjacencies come in four types: \(ab_h\), \(ba_h\), \(ab_t\), and \(ba_t\) (there is no order for these two extremities, i.e., \(a_h b_t = b_t a_h\)). If gene \(g\) lies at one end of a linear chromosome, then this end can be represented by a single extremity, \(g_h\) or \(g_t\), called a telomere. The adjacencies and telomeres of a genome form a multiset, called the adjacency set.

We define three operations on an adjacency set. The corresponding operations on the structure of the genome (relative positions and orientations of genes on chromosomes) are illustrated on Figure 1.

1. **DCJ (double-cut-and-join)** [2], which acts on one or two elements (adjacencies or telomeres) in one of the following ways: \(\{pq, rs\} \rightarrow \{pr, qs\}\) or \(\{ps, qr\}\) (see Figure 1(a)); \(\{pq, r\} \rightarrow \{pr, q\}\) or \(\{p, qr\}\) (see Figure 1(b)); \(\{p, q\} \rightarrow \{pq\}\) or \(\{pq, p, q\}\) (see Figure 1(c)).

2. **Insertion**, which inserts a single gene (a pair of extremities) \(g dg t\), in one of the following ways: \(\{pq\} \rightarrow \{pg do\}\) or \(\{ps, qr\}\) (see the upper arrow in Figure 1(d)); \(\{p\} \rightarrow \{pg do\}\) or \(\{pq, gd\}\) (see the upper arrow in Figure 1(e)); \(\emptyset \rightarrow \{gd, gdo\}\) (see the upper arrow in Figure 1(f)); \(\emptyset \rightarrow \{gd, gdo\}\) (see the upper arrow in Figure 1(g)).

3. **Deletion**, which deletes a single gene \(g dg t\), in one of the following ways: \(\{gd, gdo\} \rightarrow \{pq\}\) (see the lower arrow in Figure 1(d)); \(\{gd, gdo\} \rightarrow \{p\}\) (see the lower arrow in Figure 1(e)); \(\{gd, gdo\} \rightarrow \emptyset\) (see the lower arrow in Figure 1(f)); \(\emptyset \rightarrow \{gd, gdo\}\) (see the lower arrow in Figure 1(g)).

The *edit distance* between two adjacency sets \(S_1\) and \(S_2\), denoted as \(d(S_1, S_2)\), is the minimum number of operations (including DCJ operations, insertions and deletions) that transform \(S_1\) into \(S_2\). Here we use a unit-cost model, in which all operations have the same cost.

Note that the edit distance is defined at the adjacency set level. For genomes without duplicate genes, an adjacency set denotes a unique genomic structure. However, for genomes with duplicate genes, two genomes with different structures may share the same adjacency set as illustrated in Figure 2. Thus, \(d(S_1, S_2)\) defined above is a lower bound for the edit distance between the two genomic structures. Given two adjacency sets \(S_1\) and \(S_2\) from two genomes, let \(E_i\) be the multiset of extremities collected from all elements in \(S_i\), \(i = 1, 2\). We pair extremities in \(E_1\) and \(E_2\) into ghost adjacencies (named for the similar ghost genes of [7]) to yield the adjacency set \(T_1\); similarly, we produce \(T_2\) from \(E_2\) and \(E_1\). Clearly, to transform \(S_1\) into \(S_2\), at least \(|T_1|\) deletions and \(|T_2|\) insertions are needed. The following theorem shows that these insertions and deletions are both necessary and sufficient.

**Theorem 1.** Given two adjacency sets \(S_1\) and \(S_2\), there exists an optimal series of operations with exactly \(|T_1|\) deletions, exactly \(|T_2|\) insertions and some DCJ operations that transforms \(S_1\) into \(S_2\).

**Proof.** We prove this theorem by contradiction. Suppose that every optimal series of operations contains more than \(|T_1|\) deletions and more than \(|T_2|\) insertions. Assume that \(O_i O_j \ldots O_m\) is an optimal series of operations that contains a minimum number of insertions and deletions. Let \(S^0 S^1 S^2 \ldots S^m\) be the trace of \(S_1\) in the process of transformation, where \(S^0 = S_1\) and \(S^m = S_2\). Note that for any insertion (or deletion) beyond the \(|T_1|\) deletions and \(|T_2|\) insertions, there must be a matching deletion (or insertion) to preserve gene content. Thus every optimal series of operations has at least a pair of insertion and deletion on the same gene. Without loss of generality, assume \(O_i\) inserts a pair of extremities \(g dg t\) and \(O_j\) deletes \(g dg t\) (\(i < j\)), and operations between \(O_i\) and \(O_j\) do not contain insertion or deletion on \(g dg t\). Now we will build a new series of operations \(O_i O_{i+1} \ldots O_j\) without the pair of insertion and deletion on \(g dg t\), to replace \(O_i \ldots O_j\), which produce the trace \(S^i S^{i+1} \ldots S^j\) and satisfy \(S^j = S^i\). This process is shown in Figure 3. Denote the two extremities inserted in \(O_k\) as \(g^*_k\) and \(g^*_j\) to distinguish them from other \(g dg t\) and \(g dg t\). For \(k = i, \ldots, j - 1\), we will keep the invariant \(S^{k-1} = (S^k \setminus \{p^k g^*_k g^*_j\}) \cup \{p^k q^k\}\), where \(p^k\) (\(q^k\)) is the extremity that shares an adjacency with \(g^*_k\) (\(g^*_j\)) in \(S^k\). Note that \(p^k\) or \(q^k\) might be empty if \(g^*_i\) or \(g^*_i\) forms a telomere, or \(g^*_k\) forms an adjacency in \(S^k\).

Clearly this holds for \(k = i\), since we have both \(S^{i-1} = S^i\) and \(S^i = (S^{i-1} \setminus \{p^i q^i\}) \cup \{p^i g^*_i, q^*_i\}\). To make this invariant hold for \(k = i + 1, \ldots, j - 1\), our new operation \(O_{k-1}\) will mimic operation \(O_k\) as follows: if \(O_k\) does not affect the adjacencies or telomeres containing \(g^*_k\) or \(g^*_j\), then set \(O_{k-1} = O_k\), and the invariant holds; if \(O_k\)
acts on at least one of \( g \) or \( h \), we will build \( O'_{k-1} \) from \( O_k \) by replacing \( g \star h \) with \( p^k (q^k) \) in \( O_k \). For example, if \( O_k \) is the DCJ operation given by \( \{ p^k \star q^k \}, \{ gh, gt \} \rightarrow \{ pq, pr \} \), then \( O'_{k-1} \) would be \( \{ p^k \star q^k \}, \{ gh, gt \} \rightarrow \{ pq, pr \} \).

Since \( O_k \) does not affect \( g \) and \( h \), we have \( q^k = q^{k-1} \). Besides, we have \( p^k = d \). Thus we have \( S^k \setminus \{ p^k \star q^k \}, \{ gh, gt \} \cup \{ p^k \star q^k \} = S^{k-1} \). Other types of operations can be expressed similarly.

Recall that \( O_i \) is a deletion, i.e., \( \{ ag, bg \} \rightarrow \{ ab \} \). If \( g \) and \( t \) are the same as \( g^* \) and \( g^* \), then we have \( S^{j-2} = S_j \), and we can skip \( O'_{j-1} \) and \( O_j \) in our constructed series. If \( g_h \) and \( g_t \) are different from \( g^* \), then we have \( \{ ag, bg, p^{j-1} g_h, q^{j-1} g_t \} \subset S^{j-1} \). We can set \( O'_{j-2} \) to be \( \{ ag, bg, g_h, g_t \} \rightarrow \{ ab, g_h \} \), and set \( O_j \) to be \( \{ p^{j-1} q^{j-1}, g_h, g_t \} \rightarrow \{ p^{j-1} q^{j-1}, g_h, g_t \} \). We can verify \( S^i = S_i \), and our constructed series contradicts the optimality of \( O_1 O_2 \cdots O_m \).

**Figure 1** The effect of DCJ operations, insertions and deletions on the genomic structure. (a) and (c) represent DCJ operations, (d) and (e) (f) and (g) represent insertion and deletion. In each subfigure, the central part represents operations, and the left part and right part represent the genomic structures.

**Figure 2** Two genomes with different structures share the same adjacency set. Each edge in this figure represents a gene, each node represents an adjacency.
Adjacency graph decomposition

Given two adjacency sets $S_1$ and $S_2$ from two genomes, their corresponding adjacency graph is defined as a bipartite multigraph, $A = [S_1 \cup T_2, S_2 \cup T_1, E]$, in which $u \in S_1 \cup T_2$ and $v \in S_2 \cup T_1$ are linked by one edge if $u$ and $v$ share one extremity, by two edges if they share two extremities. Note that $S_1 \cup T_2$ and $S_2 \cup T_1$ have the same set of extremities; we use $n$ to denote half of the number of extremities. In the case of genomes with the same gene content and without duplicate genes, $T_1 = T_2 = \emptyset$, and each vertex in the adjacency graph has degree 2, which means that the adjacency graph consists of vertex-disjoint cycles and paths. We define the length of a cycle or a path to be the number of edges it contains. Based on Theorem 1, $T_1 = T_2 = \emptyset$ implies there exists an optimal solution without insertion and deletion, thus $d(S_1, S_2)$ is just the minimum number of DCJ operations needed to transform $S_1$ into $S_2$. When $S_1$ has been transformed into $S_2$, the corresponding adjacency graph only consists of cycles of length 2 and paths of length 1. Since each DCJ operation can increase the number of cycles at most by 1, or increase the number of odd-length paths at most by 2, and we can always find out this kind of operation when $S_1$ and $S_2$ are different, we have $d(S_1, S_2) = n - c \cdot o/2$, where $c$ is the number of cycles and $o$ is the number of odd-length paths in the adjacency graph [2].

In the presence of duplicate genes, the adjacency graph may contain vertices with degree larger than 2, so that there may be multiple ways of choosing vertex-disjoint cycles and paths that cover all vertices as illustrated in Figure 4. We say that a cycle (or path) in the adjacency graph is alternating if no two adjacent edges in this cycle (or path) share the same extremity. A valid decomposition of the adjacency graph is a set of vertex-disjoint alternating cycles and paths that cover all vertices. We say that a cycle of length $\ell$ is helpful if at most $\ell/2 - 1$ vertices are ghost adjacencies, unhelpful if at least $\ell/2$ vertices are ghost adjacencies. In fact, an unhelpful cycle has exactly $\ell/2$ ghost adjacencies (all in $T_1$ or all in $T_2$), since adjacencies in $T_1$ and adjacencies $T_2$ do not have common extremities and thus cannot be linked in the adjacency graph. Now we show how to perform DCJ operations, insertions and deletions to transform $S_1$ into $S_2$ based on a decomposition of the corresponding adjacency graph.

**Lemma 1.** Given two adjacency sets $S_1$ and $S_2$, and a decomposition $D$ of the adjacency graph $A = [S_1 \cup T_2, S_2 \cup T_1, E]$ with $c$ helpful cycles and $o$ odd-length paths, we can perform $n - c \cdot o/2$ operations to transform $S_1$ into $S_2$, among which there are $|T_1|$ deletions, $|T_2|$ insertions and $n - c \cdot o/2 - |T_1| - |T_2|$ DCJ operations.

**Proof.** We prove this lemma in a constructive way. We will perform operations under the guidance of the graph decomposition. The goal is to transform the adjacency graph into a collection of cycles of length 2 and paths of length 1 without ghost adjacencies, indicating that $S_1$ has been transformed into $S_2$. In the following, we will prove that an unhelpful cycle of length $\ell$ costs $\ell/2$ operations, a path of even length $\ell$ costs $\ell/2$ operations, a helpful cycle of length $\ell$ costs $\ell/2 - 1$ operations, and a path of odd length $\ell$ costs $(\ell - 1)/2$ operations. In other words, a helpful cycle requires one less operation than an unhelpful cycle or an even-length path of the same length.

For a helpful cycle of length $\ell$ with $d$ adjacencies in $T_1$ and $i$ adjacencies in $T_2$, we first perform $d$ deletions guided by this cycle to reduce the size of the cycle to $\ell - 2d$. Then for each adjacency in $T_2$, we choose one of its non-ghost neighbors in $S_1$ and perform an insertion to create one more helpful cycle of length $2$. After all adjacencies in $T_2$ are handled, we transform the cycle of length $\ell$ into one of length $\ell - 2d - 2i$ without ghost adjacencies, on which finally we can perform $\ell/2 - d - i - 1$ DCJ operations to create $\ell/2 - d - i$ cycles of length 2. An example is shown in Figure 5(a).
For a \textit{unhelpful} cycle of length \(\ell\) with \(\ell/2\) adjacencies in \(T_1\), we can perform \(\ell/2\) deletions to remove the adjacencies in \(S_1\). For a \textit{unhelpful} cycle of length \(\ell\) with \(\ell/2\) adjacencies in \(T_2\), we can first insert a gene as initial operand, then perform \(\ell/2 - 1\) insertions to create \(\ell/2\) cycles of length \(2\)—see Figure 5(b)(d).

For a path with odd length \(\ell\), we need \((\ell - 1)/2\) operations, and for a path with even length \(\ell\), we need \(\ell/2\) operations—see Figure 5(c)(e).

In sum, there are \(|T_1|\) deletions, \(|T_2|\) insertions and \(n - c - o/2 - |T_1| - |T_2|\) DCJ operations.

Lemma 1 states that any decomposition of the adjacency graph gives an upper bound on the edit distance. The following lemma shows that an optimal decomposition also provides a lower bound.

**Lemma 2.** \(d(S_1, S_2) \geq n - \max_{D \in \mathcal{D}} (c_D + o_D)/2\), where \(\mathcal{D}\) is the space of all decompositions of \(A = S_1 \cup T_2, S_2 \cup T_1, E\), \(c_D\) and \(o_D\) is the number of helpful cycles and odd-length paths in \(D\), respectively.

**Proof.** Let \(\Delta_{D'} = \max_{D' \in \mathcal{D}'} (c_{D'} + o_{D'}/2) - \max_{D'' \in \mathcal{D}''} (c_{D''} + o_{D''}/2)\), where \(\mathcal{D}'\) and \(\mathcal{D}''\) are the space of the decomposition before and after performing operation \(P\), and \(P \in \{\text{DCJ}, \text{INS}, \text{DEL}\}\). By Theorem 1, there exists an optimal series of operations with exactly \(|T_1|\) deletions and \(|T_2|\) insertions. Summing over all \(\Delta_{D'}\) for these operations in this optimal solution yields \(\sum_{i=1}^{d(S_1, S_2)} \Delta_{T_i} = (n - |T_1|) - \max_{D \in \mathcal{D}} (c_D + o_D/2)\), where \((n - |T_1|)\) is the sum of the number of helpful cycles and half of the number of odd-length paths in the optimal decomposition of the adjacency graph when \(S_1\) has been transformed into \(S_2\). Define \(\delta_{\text{DCJ}} = 1\), \(\delta_{\text{INS}} = 1\) and \(\delta_{\text{DEL}} = 0\). In the following, we will prove \(\Delta_p \leq \delta_p\) \(P \in \{\text{DCJ}, \text{INS}, \text{DEL}\}\), which implies that \(\sum_{i=1}^{d(S_1, S_2)} \Delta_{T_i} \leq d(S_1, S_2) - |T_1|\). The combination of these two formulas proves this lemma.

We prove \(\Delta_p \leq \delta_p\) by contradiction. Let \(A'\) and \(A''\) be the adjacency graphs before and after performing the operation \(P\). Let \(\sigma(A')\) and \(\sigma(A'')\) be the optimal decomposition of \(A'\) and \(A''\), respectively. Suppose \(\Delta_p > \delta_p\), namely, \((c_{\sigma(A')} + o_{\sigma(A'')}/2) - (c_{\sigma(A'')} + o_{\sigma(A')}/2) > \delta_p\). Note that \(P\) is reversible; we denote the reversed operation as \(\hat{p}\). Let \(\hat{p}\) simultaneously transforms \(\sigma(A'')\) into a decomposition of \(A'\), denoted \(\gamma(A')\). Since \(\sigma(A')\) is optimal, we have \(c_{\sigma(A')} + o_{\sigma(A'')} + \delta_p/2 \leq \gamma(A')\). Thus, to get the contradiction, we only need to prove \((c_{\sigma(A')} + o_{\sigma(A'')}/2) - (c_{\sigma(A')} + o_{\sigma(A')}/2) \leq \delta_p\). Recall that \(\gamma(A')\) is obtained from \(\sigma(A')\) by performing operation \(\hat{p}\), and both \(\gamma(A')\) and \(\sigma(A')\) are decompositions, which includes only vertex-disjoint cycles and paths.

If \(P\) is a DCJ operation, then \(\hat{p}\) is still a DCJ operation. A DCJ operation may merge two cycles into one DCJ operation, split one cycle into two cycles, merge two paths into one path, split one path into two paths, merge one path and one cycle into one path, split one path into one cycle and one path, rearrange two odd(even)-length paths into two even(odd) paths or make no change in the number of cycles and odd-length paths. Among those possible operations, the following four cases can reduce the number of helpful cycles or odd-length paths: (i) merge two helpful cycles into one helpful cycle; (ii) merge two odd-length paths into one even-length path; (iii) rearrange two odd-length paths into two even-length paths; (iv) merge one helpful cycle and one odd-length path into one odd-length path. For any of these four cases, we have \((c_{\sigma(A')} + o_{\sigma(A'')}/2) - (c_{\sigma(A')} + o_{\sigma(A')}/2) = 1\). For other possible DCJ operations, we have \((c_{\sigma(A')} + o_{\sigma(A'')}/2) - (c_{\sigma(A')} + o_{\sigma(A')}/2) \leq 0\).

If \(P\) is an insertion, then \(\hat{p}\) is a deletion. Similarly, among all possible deletions, the following five cases can reduce the number of helpful cycles or odd-length paths: (i) merge two helpful cycles into one helpful cycle; (ii) merge two odd-length paths into one even-length path; (iii) rearrange two odd-length paths into two even-length paths; (iv) merge one helpful cycle and one odd-length path into one odd-length path; (v) change a helpful cycle into an unhelpful one. For any of these five cases, we have \((c_{\sigma(A')} + o_{\sigma(A'')}/2) - (c_{\sigma(A')} + o_{\sigma(A')}/2) = 1\). For other
possible deletions, we have 
\[
(c_s(A) + o_s(A))/2 - (c_g(A) + o_g(A))/2 \leq 0.
\]

If \( P \) is a deletion, then \( \hat{P} \) is an insertion. A insertion may split one cycle into two cycles, split one path into two paths, or split one path into one cycle and one path. All these possible insertions will not reduce the number of helpful cycles or odd-length paths. Thus, any deletion will not increase the number of helpful cycles or the number of odd-length paths, and we have 
\[
(c_s(A) + o_s(A))/2 \leq (c_g(A) + o_g(A))/2. \quad \Box
\]

Combining Lemma 1 and Lemma 2, we have the following theorem.

**Theorem 2.** 
\[
d(S_1, S_2) = n - \max_{D \in \mathcal{D}} (c_D + o_D)/2,
\]
where \( \mathcal{D} \) is the space of all decompositions of \( A = \{S_1 \cup T_2, S_2 \cup T_1\} \).
∪ T₁, E}, cD and oD are the numbers of helpful cycles and odd-length paths in D, respectively.

**Approximation algorithm**

We design an approximation algorithm by using techniques employed on the problem of BREAKPOINT GRAPH DECOMPOSITION[5,6,18-20]. The basic idea is to find the maximum number of vertex-disjoint helpful cycles of length 4 in the adjacency graph. This problem can be reduced to the problem of K-SET PACKING problem with k = 4, for which the best-to-date algorithm has an approximation ratio of $2 + \varepsilon [21,22]$.

To make use of such algorithm, we must remove telomeres and keep only cycles in the adjacency graph. This can be done by introducing null extremities τ and null adjacencies ττ, which are different from other extremities and adjacencies (the same definition is introduced in [7]). Given two adjacency sets S₁ and S₂ with 2k₁ and 2k₂ telomeres respectively, we replace each telomere x by the adjacency xt. If we additionally have $k_1 < k_2$, we must add $(k_2 - k_1)$ null adjacencies ττ to S₁ in order to balance the degrees. The corresponding adjacency graph is constructed in the same way as the case without null extremities: two adjacencies are linked by one edge if they share one extremity, by two edges if they share two extremities. Now we prove that this “telomere-removal” operation does not change $d(S_1, S_2)$.

**Theorem 3.** Let $S_1$ and $S_2$ be two adjacency sets and denote by $S_1'$ and $S_2'$ the adjacency sets obtained from $S_1$ and $S_2$ by removing telomeres. Then we can write $d(S_1, S_2) = d(S_1', S_2')$.

**Proof.** We first prove $d(S_1, S_2) \geq d(S_1', S_2')$. Let $A = \{S_1 \cup T_2, S_2 \cup T_1, E\}$ be the adjacency graph with respect to $S_1$ and $S_2$ and $\sigma(A)$ be the optimal decomposition of $A$. Let $A' = \{S_1' \cup T_2, S_2' \cup T_1, E\}$ be the adjacency graph with respect to $S_1'$ and $S_2'$ and $\sigma(A')$ be the optimal decomposition of $A'$. Suppose $\sigma(A)$ contains $c$ helpful cycles, $o$ odd-length paths, and $e$ even-length paths, and among these $e$ even-length paths, $e_1$ of them contain two telomeres in $S_1$ and $e_2$ of them contain two telomeres in $S_2$. Suppose $S_1$ and $S_2$ contains $2k_1$ and $2k_2$ telomeres respectively (w.l.o.g., assume $k_1 \leq k_2$). Since an odd-length path contains one telomere in each adjacency set while an even-length path contains two telomeres in one adjacency set, we have $o + 2e_1 = 2k_1$ and $o + 2e_2 = 2k_2$. We can perform the following modifications on $\sigma(A)$ to transform it into a decomposition of $A'$ (see Figure 6). Nothing needs to be done for cycles. For odd-length paths, link their two telomeres to form a helpful cycle; for each even-length path with both telomeres in $S_1$, arbitrarily choose one even-length path with both telomeres in $S_2$ and link these two paths to form a helpful cycle; for the remaining $e_2 - e_1$ even-length paths, use $e_2 - e_1 = k_2 - k_1$ null adjacencies ττ to transform each such path into a helpful cycle. Thus, there are $c + e_2$ helpful

---

**Figure 6** One example of the “telomere-removal” and “telomere-recovery” process. Thick circles represent adjacencies containing null extremities, and thick lines represent edges connecting null extremities.
cycles in this decomposition of $A'$, so that the upper bound on $d(S'_1, S'_2)$ is $(n + k_2) - c - e_2 = n - c - o/2 = d(S_1, S_2)$. Now we prove $d(S_1, S_2) \leq d(S'_1, S'_2)$. Note that $\sigma(A')$ only consists of vertex-disjoint cycles, and unhelpful cycles cannot contain any null extremity. We claim that, for each helpful cycle in $\sigma(A')$, there must be no more than two null extremities $r$ on each side. Otherwise, we can always choose two nonadjacent edges that are linked through $r$, exchange four ends of them, and divide this cycle into two cycles (see Figure 7), contradicting the optimality of $\sigma(A')$. Now we transform $\sigma(A')$ into a decomposition of $A$ by recovering all removed telomeres (see Figure 6). Each cycle falls into one of three cases: (a) it contains one $x$ adjacency on each side, then the recovery will yield one odd-length path; (b) it contains one $r$ adjacency on one side, then the recovery will yield one even-length path; (c) it contains two $x$-like adjacencies on each side, then the recovery will yield two even-length paths. In all three cases the value $n - c - o/2$ remains unchanged, and after the recovery we obtain a decomposition of $A$. Thus we have $d(S_1, S_2) \leq d(S'_1, S'_2)$. □

In summary, based on Theorems 2 and 3, we have stated the equivalence of the problem of computing the edit distance and that of finding a valid decomposition with a maximum number of helpful cycles in an adjacency graph without telomeres. The latter problem is NP-hard by a reduction from the NP-hard problem–BREAKPOINT GRAPH DECOMPOSITION[23], since any instance of the BREAKPOINT GRAPH DECOMPOSITION is indeed an adjacency graph without ghost adjacencies. Thus, the problem of computing the edit distance is also NP-hard.

Now we give the approximation algorithm and prove that its approximation ratio is $1.5 + \epsilon$.

**Approximation Algorithm**

**Input:** Two adjacency sets $S_1$ and $S_2$ from two genomes

**Output:** A series of operations to transform $S_1$ into $S_2$.

**Step 1** Add null adjacencies to $S_1$ and $S_2$ to obtain $S'_1$ and $S'_2$ without telomeres. Build the adjacency graph $A' = \{S'_1 \cup T_2, S'_2 \cup T_1, E\}$. 

**Step 2** Collect all helpful cycles of length 4 in $A'$ as $\mathcal{C}$. Find a subset $S$ of $\mathcal{C}$ in which no two cycles share one adjacency using the $(2 + \epsilon)$-approximation algorithm for the K-SET PACKING problem with $k = 4$.

**Step 3** Remove the adjacencies covered by cycles in $S$. Arbitrarily decompose the remaining part of $A'$ into cycles, denoting this set as $S'$.

**Step 4** Remove the null adjacencies of cycles in $S \cup S'$ to obtain a decomposition of $A$. Transform $S_1$ into $S_2$ according to Lemma 1 guided by these cycles and paths.

The running time of the above algorithm is dominated by the time complexity of the $(2 + \epsilon)$-approximation algorithm for the K-SET PACKING problem with $k = 4$, which is $O(|\mathcal{C}| \log n)$ and $|\mathcal{C}| = O(n^4)$[21,22].

**Theorem 4.** The approximation ratio of the above algorithm is $1.5 + \epsilon$.

**Proof**. Suppose the optimal decomposition of $A'$ contains $p$ helpful cycles of length 4 and $q$ longer helpful cycles. Clearly, we have $n \geq 2p + 3q$. Based on Theorem 2 and Theorem 3, we know that $d(S_1, S_2) = n - p - q$. In the algorithm, we find at least $|S|$ helpful cycles, which implies that the number of operations that our algorithm outputs is at most $n - |S|$. Since $S$ is a $(2 + \epsilon)$-approximation solution, we have $(2 + \epsilon)|S| \geq OPT \geq p$, where $OPT$ is the maximum number of independent helpful cycles of length 4 in $\mathcal{C}$. The approximation ratio is thus

$$r \leq \frac{n - |S|}{n - p - q} \leq \frac{n - \frac{p + q}{2}}{n - p - q} \leq 1 + \frac{p + q - \frac{p + q}{2}}{2p - 3q - p - q} = 1.5 + \epsilon.$$ 

**Conclusion**

We studied the edit distance problem for two genomes under a unit-cost model including DCJ operations, insertions (including duplications) and deletions. We proved that this problem is equivalent to finding maximum number of helpful cycles in the adjacency graph and gave a $(1.5 + \epsilon)$-approximation algorithm. We made two main assumptions in this work: single-gene insertions and deletions; and unit cost for DCJ operations.

---

**Figure 7** Two cases of the adjacency graph with more than 2 edges that are linked through $r$. Dashed lines might represent more than one edge.
insertions and deletions. Both are clearly unrealistic. For example, large segmental duplications are common in many mammalian genomes [24], paracentric rearrangements are more common than pericentric ones, at least in two Drosophila species [25], and short inversions are more common than long ones, in some prokaryotes and in the aforementioned Drosophila [26]. These constraints should be incorporated into our distance computation. Any additional constraint naturally creates complications, but we expect that at least a few natural constraints can be handled within the framework described here.

Acknowledgements
We thank Bernard Moret for helpful discussions.

This article has been published as part of BMC Bioinformatics Volume 13 Supplement 19, 2012. Proceedings of the Tenth Annual Research in Computational Molecular Biology (RECOMB) Satellite Workshop on Computational Molecular Biology (RECOMB) Satellite Workshop on

Supplement 19, 2012: Proceedings of the Tenth Annual Research in

Supplement 19, 2012: Proceedings of the Tenth Annual Research in

Genome

Rearrangements MIT Press, 2009.

2. Bergeron A, Mixtacki J, Stoye J: A unifying view of genome rearrangements. Proc 6th Workshop Algs in Bioinf (WABI’06), Volume 4175 of Lecture Notes in Comp Sci Springer Verlag, Berlin, 2006, 163-173.

3. Yancopoulos S, Atiu O, Friedberg R: Efficient sorting of genomic permutations by translocation, inversion and block interchange. Bioinformatics 2005, 21(16):3340-3346.

4. Bergeron A, Mixtacki J, Stoye J: A new linear-time algorithm to compute the genomic distance via the double cut and join distance. Theor Comput Sci 2009, 410(51):5300-5316.

5. Chen X, Sun R, Yu J: Approximating the double-cut-and-join distance between unsigned genomes. BMC Bioinformatics 2011, 12(Suppl 9):S17.

6. Yancopoulos S, Friedberg R: Sorting genomes with insertions, deletions and duplications by DCJ. recombinat 2008, 170-182.

7. Hannenhalli S, Pevzner P: Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). Proc 27th Ann ACM Symp Theory of Comput (STOC’95) ACM Press, New York, 1995, 178-189.

8. Bader D, Moret B, Yan M: A fast linear-time algorithm for inversion distance with an experimental comparison. J Comput Biol 2001, 8(5):462-491.

9. Jean G, Nikolikoi M: Genome rearrangements: a correct algorithm for optimal capping. Inf Proc Letters 2007, 104(4):14-20.

10. Ozery-Flato M, Shamir R: Two notes on genome rearrangement. J Bioinf Comp Bio 2003, 1:71-94.

11. Tesler G: Efficient algorithms for multichromosomal genome rearrangements. J Comput Syst Sci 2002, 65(5):607-669.

12. El-Mabrouk N: Sorting signed permutations by reversals and insertions/deletions of contiguous segments. Journal of Discrete Algorithms 2001, 1:105-122.

13. Chen X, Zheng J, Fu Z, Nan P, Zhong Y, Lonardi S, Jiang T: Assignment of orthologous genes via genome rearrangement. ACM/IEEE Trans on Comput Bio & Bioinf 2005, 2(4):302-315.

14. Braga M, Willing E, Stoye J: Genomic distance with DCJ and indels. Algorithms in Bioinformatics 2010, 90-101.

15. Angibaud S, Fertin G, Rusu I, Valette S: A pseudo-boolean framework for computing rearrangement distances between genomes with duplicates. Pcb 2007, 14(4):379-393.

16. Angibaud S, Fertin G, Rusu I, Thévenin A, Valette S, et al: On the approximability of comparing genomes with duplicates. Journal of Graph Algorithms and Applications 2009, 13:19-53.

17. Caprara A, Rizzi R: Improved approximation for breakpoint graph decomposition and sorting by reversals. J of Combin Optimization 2002, 6(2):157-182.

18. Christie D: A 3/2-approximation algorithm for sorting by reversals. Proc 9th Ann ACM/SIAM Symp Discrete Algs (SODA’98) SIAM Press, Philadelphia, 1998, 244-252.

19. Lin G, Jiang T: A further improved approximation algorithm for breakpoint graph decomposition. J of Combin Optimization 2004, 8(2):183-194.

20. Halldórsson M: Approximating discrete collections via local improvements. Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics 1995, 160-169.

21. Hurkens C, Schrijver A: On the size of systems of sets every t of which have an SDR, with an application to the worst-case ratio of heuristics for packing problems. SIAM Journal on Discrete Mathematics 1989, 2:268-72.

22. Kececioglu J, Sankoff D: Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. Algorithmica 1995, 13:180-210.

23. Bailey J, Eichler E: Primate segmental duplications: crucibles of evolution, diversity and disease. Nature Reviews Genetics 2006, 7(7):552-564.

24. York T, Durrett R, Nielsen R: Dependence of paracentric inversion rate on tract length. BMC Bioinformatics 2007, 8(115).

25. Lefebvre F, El-Mabrouk N, Tillier E, Sankoff D: Detection and validation of single gene inversions. Proc 11th Intl Conf on Intelligent Systems for Mol Biol (ISMB’03), Volume 19 of Bioinformatics Oxford U Press, 2003, 1190-1196.

Cite this article as: Shao and Lin: Approximating the edit distance for genomes with duplicate genes under DCJ, insertion and deletion. BMC Bioinformatics 2012 13(Suppl 19):S13.

Submit your next manuscript to BioMed Central and take full advantage of:

• Convenient online submission
• Thorough peer review
• No space constraints or color figure charges
• Immediate publication on acceptance
• Inclusion in PubMed, CAS, Scopus and Google Scholar
• Research which is freely available for redistribution

Submit your manuscript at www.biomedcentral.com/submit