Duplication Distance
Research Comprehensive Project

Crystal Kahn
Advised by Ben Raphael

December 17, 2007

1 Introduction

Genome sequences mutate and rearrange through the course of evolution. Genome rearrangement is an important and rich area of computational biology, and has many applications, including inferring phylogenetic trees. Some genome rearrangement problems, such as reversal distance [6], [1] and translocation distance [5], have been studied. Reversal distance, for example, measures the number of substring inversions necessary to transform one signed permutation into another. Translocation distance measures the number of substring "cut-paste" operations that are necessary to transform one signed permutation into another. See [7] for a survey of rearrangement problems. However, there remain many genomic rearrangements that are not well understood, and there does not exist a unified genome rearrangement metric that can be used to quantify the similarity of two arbitrary genomes.

Recently, it has become possible to study mutations and rearrangements in cancer. The types of rearrangement operations that have been observed in cancer development include previously studied operations, such as reversals and translocations, but also include new classes of rearrangement operations. One such operation is duplication via amplisome, a genetic extrachromosomal plasmid or molecule that can replicate autonomously and integrate itself into a chromosome [8], [2], [9], [11]. As a step toward developing a model for this type of genome rearrangement, we define the *amplisome distance* between a healthy genome G and a tumor genome T to be the minimum number of rearrangement operations needed to create an extrachromosomal string A from substrings of G and then to transform G into T by inserting substrings of A into G. The `Amplisome Distance Problem` is to determine the shortest sequence of operations necessary to build the string A and then to transform G into T. We believe that the `Amplisome Distance Problem` is difficult, in general.

In Section 2 we describe a subproblem of the Amplisome Distance Problem, the Duplication Distance Problem. Computing the duplication distance between strings can be seen as a subproblem of both computing the minimum number of rearrangements to build the string A and of computing the minimum number of rearrangements to transform G into T, given the extrachromosomal element A. In Section 3, we describe a polynomial-time algorithm for the Duplication Distance Problem. In Section 4 we describe a closely related problem, the Reversal/Duplication Transposition Distance Problem. We demonstrate that the the current best-known algorithm for the problem is flawed and show that Duplication Distance is a lower bound on Duplication Transposition Distance and

thus yields a more parsimonious rearrangement scenario.

2 The Duplication Distance Problem

Given a string T, a *substring* of T is a contiguous set of characters of T. For example, given a string $T = abcd$, the string $S' = bc$ is a substring of T, but the string $S'' = ac$ is not a substring of T. We let $|S|$ denote the length of the substring S. For a string $Y = y_1 \dots y_n$, let $Y_{s,t}$ be the substring $y_s y_{s+1} \dots y_{t-1} y_t$.

Given two strings X and Z, a *duplicate operation* copies a substring of X and pastes it into Z, resulting in a new, longer string.

Example 2.1.
Consider the strings:
$X = abcdefgh$,
$Z = abhhefhgh$.

Using X as a source string, we can transform Z into the the string $abh\mathbf{cd}hfhgh$ by copying the substring cd of X and inserting it into Z at the fourth position. Formally, we have the following definition.

Definition 2.2 (*Duplicate* Operation)**.** A *duplicate* operation, $\delta_{s,t,p}(X)$, copies a substring $x_s \dots x_t$ of a source string X and pastes it into a target string at position p. Specifically, if $X = x_1 \dots x_m$ and $Z = z_1 \dots z_n$, then $Z \circ \delta_{s,t,p}(X) = z_1 \dots z_{p-1} x_s \dots x_t z_p \dots z_n$. (Figure 1).

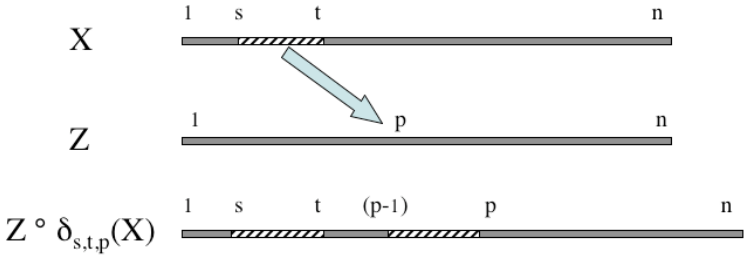

Figure 1: A duplicate operation in which a substring of X is copied and pasted into Z.

Given a string X, we define the *duplication distance* between strings Y and Z as the minimum number of duplicate operations with source X and target Z that transforms Z into Y. The duplication distance, therefore, quantifies the similarity of the strings Z and Y, given the existence of an exterior string X. Note that we use the term "distance" loosely throughout this paper; in particular, duplication distance is not symmetric.

Example 2.3.
Consider the strings:
$X = abcdefgh$,
$Y = abhcdhefhgh$,
$Z = \emptyset$,

where Z is the empty string, denoted $\emptyset$. Note that the substrings ab, h, cd, h, ef, h, gh of Y are also substrings of X. Therefore, we can copy each of these substrings from X and paste them into Z, giving a sequence of seven individual duplicate operations. Specifically, $Y = Z \circ \delta_{1,2,1}(X) \circ \delta_{8,8,3}(X) \circ \delta_{3,4,4}(X) \circ \delta_{8,8,6}(X) \circ \delta_{5,6,7}(X) \circ \delta_{8,8,9}(X) \circ \delta_{7,8,10}(X)$, where the duplicate operations are applied to Z in a left-to-right order. The question remains: is it possible to construct Y by a sequence of fewer than seven duplicate operations? This question is answered by computing the duplication distance.

To define the `Duplication Distance Problem` formally, we require the following definition.

Definition 2.4 (Ambiguous)**.** A string S is ambiguous if there exists a character c in S such that c appears in multiple instances in S. A string S is non-ambiguous provided it contains at most one instance of every character.

For example, the string $T = abcadaefb$ is ambiguous because it contains multiple instances of the characters 'a' and 'b.'

We now state the `Duplication Distance Problem`.

`Input`: Strings X, Y, Z, where X is non-ambiguous, Y is ambiguous, Z is a non-contiguous subsequence of Y, and $\{X\}$ the set of characters comprising X, is a superset of $\{Y\} \setminus \{Z\}$.
`Output`: A minimum sequence $\Delta = (\delta_{s_1,t_1,p_1}(X), \delta_{s_2,t_2,p_2}(X), \ldots, \delta_{s_r,t_r,p_r}(X))$ of duplicate operations such that $Z \circ \delta_{s_1,t_1,p_1}(X) \circ \delta_{s_2,t_2,p_2}(X) \circ \cdots \circ \delta_{s_r,t_r,p_r}(X) = Y$.

We let $D_X(Z, Y)$ denote the length of a minimum sequence Δ. Returning to Example 2.3, we have already seen that it is possible to transform Z into Y in seven operations. However, an alternative approach to constructing Y would be to copy all of X in one duplicate operation, and then insert the substring h three times into the third, fifth, and seventh indices of Z, respectively. Thus, we can build Y in a total of four duplicate operations. Note that one of the duplicated substrings of X is not a substring of Y: namely, $abcdefgh$. Thus, to compute the minimum duplication distance between Z and Y it will not suffice to consider only substrings of Y that are also substrings of X. However, we will show that considering non-contiguous subsequences of Y and the ways in which they overlap leads to an efficient algorithm for duplication distance.

2.1 Preliminaries

We define some terminology.

Definition 2.5 (Subsequence)**.** A string $S = s_1 s_2 \ldots s_m$ is a *subsequence* of a string $T = t_1 t_2 \ldots t_n$ if there exists an integer sequence $I_T(S) = (i_1, i_2, \ldots, i_m)$ satisfying $1 \leq i_1 < i_2 < \cdots < i_m \leq n$ such that $s_k = t_{i_k}$ for $k = 1, \ldots, m$.

For example, given a string $T = abcd$, the string $S = abd$ is a subsequence of T and the corresponding integer sequence is $I_T(S) = (1, 2, 4)$. For a subsequence S, we denote the index of its first character in Y by $I_T(S)[1]$, the index of its second character by $I_T(S)[2]$, the index of its third character by $I_T(S)[3]$, and so on. We let $|S|$ denote the length of the subsequence. Also, for a character x, let $I_Y(x) = \{k \mid Y_k = x\}$, i.e. the set of indices k such that $y_k = x$.

Note that, given a string T, a substring of T is also a subsequence of T, but a subsequence need not be a substring.

Definition 2.6 (Inside)**.** Given two increasing integer sequences, $\sigma = (\sigma_1, \ldots, \sigma_m)$ and $\tau = (\tau_1, \ldots, \tau_n)$, σ is *inside* τ if there exists an integer k such that $\sigma_1 > \tau_k$ and $\sigma_m < \tau_{k+1}$. If R and S are subsequences of a string T, then R is *inside* S if $I_T(R)$ is inside $I_T(S)$.

For example, in the string $T = abcdef$, the subsequence $\sigma = bd = (2, 4)$ is inside the subsequence $\tau = aef = (1, 5, 6)$.

Definition 2.7 (Internal Strings)**.** Given a string $T = t_1 \ldots t_n$ and an integer sequence $\sigma = (\sigma_1, \ldots, \sigma_m)$ corresponding to a subsequence of T, we denote by $T \setminus \sigma$, the set of maximal substrings of T that are inside σ. That is, $T \setminus \sigma = \{t_{\sigma_i+1} t_{\sigma_i+2} \ldots t_{\sigma_{i+1}-1} \mid 1 \leq i < m,\ \sigma_i + 1 \neq \sigma_{i+1}\}$. We call these the set of *internal strings* of σ in T.

For example, given the string $T = abcdefghijk$ and the subsequence $\sigma = (1, 2, 4, 6, 9)$, the internal strings of σ are $T \setminus \sigma = \{c, e, gh\}$.

Definition 2.8 (Overlap)**.** Given two integer sequences σ and τ, σ *overlaps* τ if either (i) there exist integers k and j such that $\sigma_k = \tau_j$; or (ii) there exist integers k_1, k_2, j_1, j_2, satisfying $k_1 < k_2$ and $j_1 < j_2$ such that $\sigma_{k_1} < \tau_{j_1} < \sigma_{k_2} < \tau_{j_2}$. We say that σ and τ are *overlapping* if either σ overlaps τ or τ overlaps σ. If R and S are subsequences of a string T, we say that R overlaps S if $I_T(R)$ overlaps $I_T(S)$.

For example, in the string $T = abcdef$, the subsequence $R = abd$ overlaps the subsequence $S = ce$ because $I_T(R) = (1, 2, 4)$ overlaps $I_T(S) = (3, 5)$. The subsequences R and $R' = bc$ are also overlapping because $I_T(R) \cap I_T(R')$ contains 2.

3 A Polynomial Algorithm for Duplication Distance

We shall give an algorithm for the Duplication Distance problem that is polynomial in the lengths of the input strings. Our main result is stated in the following theorem.

Theorem 3.1. *Let X, Y and Z be strings such that X is non-ambiguous, Z is a subsequence of Y, and $\{Z\} \setminus \{Y\} \subseteq \{X\}$. There is an algorithm to compute a minimum sequence $\Delta = (\delta_{s_1,t_1,p_1}(X), \delta_{s_2,t_2,p_2}(X), \ldots, \delta_{s_r,t_r,p_r}(X))$ of duplicate operations such that $Z \circ \delta_{s_1,t_1,p_1}(X) \circ \delta_{s_2,t_2,p_2}(X) \circ \cdots \circ \delta_{s_r,t_r,p_r}(X) = Y$ that is polynomial in $|Y|$.*

In general, the input strings represent genomes and the characters of the strings represent conserved segments, i.e. genes. For each character c in Z that appears multiple times in Y, exactly one of those instances in Y corresponds to the "original" copy of c, i.e. the copy of c that did not result from a duplicate operation. Therefore, we can define a mapping from the characters (or genes) in Z and the indices of their respective original copies in Y. Sometimes, this mapping is known or can be inferred from biological data; however, sometimes it cannot. In Sections 3.1 and 3.2, we give an algorithm for computing the minimum duplication distance $D_X(Z, Y)$ where we suppose that we are given the mapping from indices of Z to the indices of the original copies in Y as part of the input. Then, in Section 3.3, we show how to compute such a mapping if it is not known.

3.1 Computing Minimum Duplication Distance

Suppose we are given a mapping from the characters of Z to the indices of their respective original copies in $Y = y_1 \ldots y_n$. Let $\sigma = \sigma_1 \ldots \sigma_m$ denote the subsequence of Y to which all the characters

of Z are mapped. Then the problem of computing the duplication distance $D_X(Z, Y)$ reduces to a number of smaller duplication distance problems of the form $D_X(\emptyset, Y_i)$ where $Y_i \in Y \setminus \sigma \cup \{y_1 \dots y_{\sigma_1 - 1}, y_{\sigma_m + 1} \dots y_n\}$. Note that $\sum_{Y_i} |Y_i| \leq n$. In this section, we show how to compute $D_X(\emptyset, Y_i)$ where Y_i is some substring of Y. Therefore, in this section, we assume that Z is the empty string unless otherwise stated. We demonstrate, in Section 3.3, that there is an efficient algorithm for computing the mapping from characters of Z to indices of Y that yields the minimum duplication distance.

We define $d(Y) = D_X(\emptyset, Y)$. A sequence of duplicate operations to build the string Y defines a set of substrings of X that are copied and pasted into Z. Let $S = \{\sigma_1, \dots, \sigma_t\}$ be such a set. Every element of S is both a substring of X and a subsequence of the final string Y. Moreover, the corresponding integer sequences $I_Y(\sigma_1), \dots, I_Y(\sigma_t)$ are a partition of the set $\{1, \dots, |Y|\}$. We have the following definition.

Definition 3.2 (Candidate Set)**.** Let X and Y be strings where X is non-ambiguous. A set $S = \{\sigma_1, \dots, \sigma_t\}$ of subsequences of Y is a candidate set provided each σ_i is a substring of X and the corresponding integer sequences $I_Y(\sigma_i)$ partition $\{1, \dots, |Y|\}$.

Example 3.3.
Consider the following strings:
$X = abcdef$,
$Y = adbecf$,
$Z = \emptyset$.

Consider the set $S = \{abc, def\}$ of subsequences of Y. It is clear that both elements of S are substrings of X and that $\bigcup_{\sigma \in S} I_Y(\sigma) = \{1, 3, 5\} \cup \{2, 4, 6\} = \{1, 2, 3, 4, 5, 6\}$. Therefore, S is a candidate set of subsequences. However, there is no order on S of duplicate operations corresponding to copying an element of S from X and producing Y. This is because the subsequences abc and def overlap in Y. On the other hand, the sets $S' = \{abc, d, e, f\}$, $S'' = \{def, a, b, c\}$ and $S''' = \{a, b, c, d, e, f\}$ are all candidates sets of subsequences for which there do exist corresponding sequences of duplicate operations to produce Y. For example, the elements of S' can be duplicated in the following order to create Y: $Y = \emptyset \circ \delta_{1,3,1}(X) \circ \delta_{4,4,2}(X) \circ \delta_{5,5,4}(X) \circ \delta_{6,6,6}(X)$.

Lemma 3.4. *Given a candidate set $S = \{\sigma_1, \dots, \sigma_k\}$ of subsequences of Y, there exists a sequence $\Delta_S = \delta_{s_1,t_1,p_1}(X) \circ \dots \circ \delta_{s_k,t_k,p_k}(X)$ of duplicate operations such that $Y = Z \circ \delta_{s_1,t_1,p_1}(X) \circ \dots \circ \delta_{s_k,t_k,p_k}(X)$ and the copied substrings of X are $\{X_{s_1,t_1}, \dots, X_{s_k,t_k}\} = \{\sigma_1, \dots, \sigma_k\}$ if and only if σ_i and σ_j are non-overlapping for all $\sigma_i, \sigma_j \in S, i \neq j$.*

Proof. Let $S = \{\sigma_1, \dots, \sigma_t\}$ be a candidate set where the elements of S are all mutually non-overlapping in Y. We shall exhibit an ordering on S that corresponds to a sequence of duplicate operations that builds Y.

Let $\sigma_i, \sigma_j \in S$ with $i \neq j$. Since the elements of S are mutually non-overlaping, either σ_i is inside σ_j, σ_j is inside σ_i, or the convex hulls of $I_Y(\sigma_i), I_Y(\sigma_j)$ are disjoint. We define the following relation on elements of S. $\sigma_i \prec \sigma_j$ if σ_j is inside σ_i or if the convex hulls of $I_Y(\sigma_i)$ and $I_Y(\sigma_j)$ are disjoint and $I_Y(\sigma_i)[1] < I_Y(\sigma_j)[1]$.

We now show that $\prec$ defines an ordering on S, i.e. we show that for all $\sigma_i, \sigma_j, \sigma_k \in S$, if $\sigma_i \prec \sigma_j$ and $\sigma_j \prec \sigma_k$, then $\sigma_i \prec \sigma_k$. Let $\sigma_i, \sigma_j, \sigma_k \in S$ be ordered such that $\sigma_i \prec \sigma_j$ and $\sigma_j \prec \sigma_k$. There are two cases to consider for how σ_i and σ_j are related: (1) σ_j is inside σ_i. (2) The convex hulls of

$I_Y(\sigma_i)$ and $I_Y(\sigma_j)$ are disjoint and $I_Y(\sigma_i)[1] < I_Y(\sigma_j)[1]$. Similarly, there are two cases to consider for σ_j and σ_k: (A) σ_k is inside σ_j. (B) The convex hulls of $I_Y(\sigma_j)$ and $I_Y(\sigma_k)$ are disjoint and $I_Y(\sigma_j)[1] < I_Y(\sigma_k)[1]$.

There are a total of four cases for how all three subsequences are related: (1A) σ_k will be inside σ_i, so $\sigma_i \prec \sigma_k$ in the ordering. (1B) Either (i) σ_k is inside σ_i or (ii) the convex hulls of $I_Y(\sigma_i)$ and $I_Y(\sigma_k)$ are disjoint and $I_Y(\sigma_i)[1] < I_Y(\sigma_k)[1]$ because σ_i and σ_k are non-overlapping and $I_Y(\sigma_j)[1] < I_Y(\sigma_k)[1]$. In either case, σ_i will be ordered before σ_k. (2A) The convex hulls of σ_i and σ_k will be disjoint and $I_Y(\sigma_i)[1] < I_Y(\sigma_k)[1]$, so σ_i will be ordered before σ_k. (2B) The convex hulls of σ_i and σ_k will be disjoint and $I_Y(\sigma_i)[1] < I_Y(\sigma_k)[1]$, so σ_i will be ordered before σ_k.

Conversely, let $C = \{\sigma_1, \ldots, \sigma_t\}$ be a candidate set for which there exists an order on C corresponding to a sequence of duplicate operations to build Y. Let $\sigma_i, \sigma_j \in C$ and suppose, without loss of generality, that σ_i is before σ_j in the ordering. Consider the operation, δ_j, that copies σ_j into the target string. Because σ_i is before σ_j in the ordering, σ_i will already be a subsequence of the target string. The operation, δ_j, will insert the string σ_j into the target string as a contiguous substring. Therefore, σ_j can either be inserted at some index before the first character of σ_i, after the last character of σ_i or inside σ_i. In any case, σ_i and σ_j will be non-overlapping. □

A result of Lemma 3.4 is that a candidate set S of subsequences of Y for which all the elements of S are mutually non-overlapping corresponds to a sequence of duplicate operations to build Y whose length equals the cardinality of S. We call such sets *feasible sets of subsequences*.

Definition 3.5 (Feasible Set of Subsequences)**.** Let X and Y be strings where X is non-ambiguous. A *feasible set* $\mathcal{F} = \{\sigma_1, \ldots, \sigma_k\}$ is a candidate set where σ_i and σ_j are non-overlapping for all $\sigma_i, \sigma_j \in \mathcal{F}$ with $i \neq j$.

Claim 3.6. *Let X and Y be strings where X is non-ambiguous and $\{Y\} \subseteq \{X\}$ and let $n = |Y|$. Given a feasible set S of minimum cardinality, there is an $O(n^2)$ algorithm to compute Δ, a minimum-length sequence of duplicate operations to build Y from the empty string. Moreover, $d(Y) = |S|$.*

Proof. By Lemma 3.4 and Definition 3.5, every sequence of duplicate operations that builds Y from the empty string corresponds to a feasible set, and the length of a sequence of duplicate operations that builds Y is equal to the cardinality of its corresponding feasible set. Therefore, a minimum-cardinality feasible set of subsequences of Y will corresponding to the minimum-length sequence of duplicate operations.

Moreover, in the proof of Lemma 3.4, we defined an ordering on the elements of a feasible set S that corresponds to an order of duplicate operations that builds Y. The ordering can be computed by first ordering every pair of elements of S with respect to each other. Then, we compute the total ordering on elements of S using an insertion-sort-like algorithm. This computation takes time $O(s^2)$ where $s = |S|$. If $|Y| = n$, $s \in O(n)$. □

Therefore, given a feasible set of subsequences of Y with minimum cardinality, we can compute the duplication distance and the corresponding order of duplicate operations. In the next section, we shall describe how to compute a feasible set of subsequences of minimum cardinality.

3.2 Computing a Minimum-Cardinality Feasible Set of Subsequences

We present here an efficient dynamic programming algorithm to compute the cardinality of the minimum feasible set of subsequences of Y which, by Claim 3.6, gives a minimum sequence of duplicate operations. It is straightforward to augment the algorithm to compute also the feasible set itself.

Given a string $s = x_1 \dots x_r$ that appears as a subsequence of Y with multiple corresponding integer sequences, i.e. $s = \sigma_i = \sigma_j$, but $I_Y(\sigma_i) \neq I_Y(\sigma_j)$, we call each integer sequence $I_Y(\sigma_i)$ a *placement* of s.

For integers $1 \leq s \leq t \leq |X|$, let $X^{s,t}$ denote the substring of X that begins with the character y_s and ends with the character y_t. Note that $X^{s,t}$ may not be defined, but if it is defined, then it is unique because X is non-ambiguous. Define the set $\mathcal{S}(X^{s,t}) = \{\sigma \mid I_Y(\sigma)$ is a placement of $X^{s,t}$ in $Y_{s,t}$, with $I_Y(\sigma)[1] = s, I_Y(\sigma)[|\sigma|] = t\}$.

For example, let $X = abcdefghij$, $Y = bcdedefg$. The substring $cdef$ of X appears as a subsequence of $Y_{2,7}$ in multiple placements. The set $\mathcal{S}(X^{2,7})$ contains the subsequences with corresponding integer sequences $\{(2,3,4,7),(2,5,6,7),(2,3,6,7)\}$.

For completeness, define $d(Y_{s,t}) = 0$ for indices $s > t$.

Theorem 3.7. *Given a string $Y_{s,t}$ and indices $s \leq t$, the value $d(Y_{s,t})$ satisfies the following recurrence.*

$$d(Y_{s,t}) = \begin{cases} 1 & \text{if } s = t, \\ \min\ \{a_{s,t}, b_{s,t}\} & \text{otherwise,} \end{cases} \tag{1}$$

where

$$a_{s,t} = \begin{cases} \infty & \text{if } X^{s,t} \text{ is undefined} \\ \min_{\tau \in \mathcal{S}(X^{s,t})} \left(1 + \sum_{Y_{i,j} \in Y_{s,t} \setminus \tau} d(Y_{i,j})\right) & \text{otherwise,} \end{cases} \tag{2}$$

and

$$b_{s,t} = \min_{r=s,\dots,t-1} \left(d(Y_{s,r}) + d(Y_{r+1,t})\right). \tag{3}$$

Proof. As a base case, if $s = t$, then $d(Y_{s,t}) = 1$ because a string of one character requires exactly one duplicate operation for its construction.

We assume, inductively, that we have already computed $d(Y_{s,t})$ for every substring $Y_{s,t}$ with $|Y_{s,t}| = t - s + 1 \leq i - 1$.

Now consider a string $Y_{s,t}$ with $|Y_{s,t}| = t - s + 1 = i$, and consider all feasible sets of subsequences of $Y_{s,t}$. In every feasible set, there must exist some element σ such that $s \in I_{Y_{s,t}}(\sigma)$. There are two classes of feasible sets: either $t \in I_{Y_{s,t}}(\sigma)$ or $t \notin I_{Y_{s,t}}(\sigma)$.

Suppose S is a feasible set of subsequences with $\sigma \in S$ and $s, t \in I_{Y_{s,t}}(\sigma)$. Because σ contains both s and t, it corresponds to a placement of the substring $X^{s,t}$ of X ranging between indices $I_X(Y_s)$ and $I_X(Y_t)$. As noted above, the substring $X^{s,t}$ is uniquely defined. Given σ, the indices of $\{s, s+1, \ldots, t-1, t\}$ not covered by σ consist of substrings of $Y_{s,t}$ that are inside σ. Each of these substrings are mutually non-overlapping and, moreover, do not overlap with σ. Therefore, the minimum-cardinality feasible set that includes an element σ such that $s, t \in I_{Y_{s,t}}(\sigma)$ will be comprised of: the element σ and the minimum-cardinality feasible sets for each $Y_{i,j} \in Y_{s,t} \setminus \sigma$. Thus, the feasible set of minimum cardinality is obtained by considering the minimum sum of the cardinalities of the minimum feasible sets for each $Y_{i,j} \in Y_{s,t} \setminus \tau$ for every possible placement τ of $X^{s,t}$ in $Y_{s,t}$. The placement τ^* of $X^{s,t}$ that minimizes this sum is the optimal placement. The value of $a_{s,t}$ is the value of this sum plus one for the subsequence τ^*.

Now suppose S is a feasible set of subsequences with $\sigma \in S$, $s \in I_{Y_{s,t}}(\sigma)$ but $t \notin I_{Y_{s,t}}(\sigma)$. Then the element σ contains some maximal index $r < t$, and S can be partitioned into two subsets of elements: a set of feasible subsequences of $Y_{s,r}$, consisting of σ and some set of subsequences inside σ, and a feasible set of subsequences of $Y_{r+1,t}$. The elements in the union of these two subsets are mutually non-overlapping. Therefore, the value of $d(Y_{s,t})$ where $t \notin I_{Y_{s,t}}(\sigma)$ will be the minimum value of $d(Y_{s,r}) + d(Y_{r+1,t})$, taken over all values $s \leq r < t$. This value is $b_{s,t}$. □

Note that in computing the recurrence in Equation 1, the value $b_{s,t}$ can be computed in time linear in the size of $Y_{s,t}$.

Now we describe how to compute the value $a_{s,t}$ efficiently.

Let $a'_{s,t}(X^{s,t}) = a_{s,t} - 1$.

Lemma 3.8. *Given a substring $X^{s,t}$ of X whose first and last characters are located at indices s and t in Y, the value $a'_{s,t}(X^{s,t})$ satisfies the following recurrence.*

$$a'_{s,t}(X^{s,t}) = \begin{cases} \infty & \text{if } X^{s,t} \text{ is undefined,} \\ d(Y_{s+1,t-1}) & \text{if } |X^{s,t}| = 2 \text{ (base case)} \\ \min_{r \in I_{Y_{s,t}}(X_2^{s,t})} \left(d(Y_{s+1,r-1}) + a'_{r,t}(X_{2,*}^{s,t}) \right) & \text{otherwise} \end{cases} \tag{4}$$

where $X_2^{s,t}$ is the second character of $X^{s,t}$, and $X_{2,*}^{s,t}$ is the substring of $X^{s,t}$ containing all but the first character.

Moreover, for fixed values of s and t, we compute $a'_{s,t}(X^{s,t})$ in a total of $O(l^2)$ time, where $l = |Y_{s,t}|$.

Proof. Recall that $a_{s,t} = \min_{\sigma \in \mathcal{S}(X^{s,t})} \left(1 + \sum_{Y_{i,j} \in Y_{s,t} \setminus \sigma} d(Y_{i,j})\right)$ where $s, t \in I_{Y_{s,t}}(\sigma)$. And $a_{s,t} = 1 + a'_{s,t}(X^{s,t})$.

Consider the substring $Y_{s,t}$ of Y and the substring $X^{s,t}$ of X such that the first character of $X^{s,t}$ is Y_s and the last character is Y_t. First, we note that if $X^{s,t}$ is undefined, $a_{s,t}$ should equal infity, and thus, $a'_{s,t}(X^{s,t})$ should return infinity.

Next we show that the base case is correct. If $X^{s,t}$ has only two characters, then there is only one placement $\sigma = (s, t)$ of $X^{s,t}$ in $Y_{s,t}$, and $Y_{s,t} \setminus \sigma = \{Y_{s+1,t-1}\}$. So, $a'_{s,t}(X^{s,t}) = d(Y_{s+1,t-1})$.

Finally, if $|X^{s,t}| > 2$, then there could exist more than one placement of $X^{s,t}$ in $Y_{s,t}$. And, for a placement σ, the set $Y_{s,t} \setminus \sigma$ may contain more than one element. Consider the suffix $X_{i,*}^{s,t}$ of $X^{s,t}$

for $1 < i < |X^{s,t}|$. Assume, inductively, that for all $i < k \leq |X^{s,t}|$ and for all $r_k \in I_{Y_{s,t}}(X^{s,t}_k)$ the value $a'_{r_k,t}(X^{s,t}_{k,*})$ has already been computed and equals the sum $\sum_{Y_{i,j} \in Y_{r,t} \setminus \sigma_k} d(Y_{i,j})$ where σ_k is the optimal placement of $X^{s,t}_{k,*}$ in $Y_{r_k,t}$. For every placement r_{i+1} of $X^{s,t}_{i+1}$ in $Y_{s+1,t}$, we compute $d(Y_{s+1,r_{i+1}-1})$ and add it to $a'_{r_{i+1},t}(X^{s,t}_{i+1,*})$.

The recurrence, therefore, considers all possible placements of $X^{s,t}$, and for each placement, it sums the values of $d(Y_{i,j})$ of the maximal substrings $Y_{i,j}$ between successively placed characters of $X^{s,t}$. The placement that minimizes this sum is the placement that optimizes the sum $a'_{s,t}(X^{s,t})$.

Consider a substring $X^{s,t}$ of X and some suffix $X^{s,t}_{i,*}$ including all but the first $i-1$ characters. We assume, by induction, that the value of $a'_{s,t}(X^{s,t}_{k,*})$ has already been computed for every value of $i < k \leq |X^{s,t}|$ and that it is equal to the sum of the values $d(Y_{i,j})$ for each element in $\{Y_{i,j} | Y_{i,j} \in Y_{s,t} \setminus \sigma, \sigma \text{ is optimal placement of } X^{s,t}_{k,*} \text{ in } Y_{s,t}\}$. For a fixed i, the time to compute $a'_{s,t}(X^{s,t}_{i,*})$ is bounded by $|I_{Y_{s+1,t}}(X^{s,t}_i)| * |I_{Y_{s+1,t}}(X^{s,t}_{i+1})|$ where $I_{Y_{s+1,t}}(X^{s,t}_i)$ is the number of placements of the character $X^{s,t}_i$ in the substring $Y_{s+1,t}$. Therefore, the total time to compute $a'_{s,t}(X^{s,t})$ is bounded by $\sum_{i=1}^{|X^{s,t}|} \left[|I_{Y_{s,t}}(X^{s,t}_i)| * |I_{Y_{s,t}}(X^{s,t}_{i+1})| \right]$. However, every character of the substring $Y_{s+1,t}$ can correspond to at most one character of $X^{s,t}$ because $X^{s,t}$ is non-ambiguous. Therefore, this sum is bounded by $\sum_{j=s}^{t} |I_{Y_{j+1,t}}(X_{i+1})|$ where $y_j = x_i$ which is, in turn, bounded by $O(l^2)$. □

Let $n = |Y|$. Note that $l \in O(n)$. The running time for computing $d(Y_{s,t})$ is dominated by the time to compute $a_{s,t}$. The value $a_{s,t}$ is computed once for every substring of Y, of which there are $O(n^2)$, so the total running time to compute the cardinality of the smallest feasible set of subsequences of Y takes $O(n^4)$ time. Combining Theorem 3.7 and Lemma 3.8 obtains:

Theorem 3.9. *Let X and Y be strings such that X is non-ambiguous and $\{Y\} \subseteq \{X\}$. Let $n = |Y|$. There is an $O(n^4)$ algorithm for computing $D_X(\emptyset, Y)$.*

Furthermore, as stated above, the algorithm described by the recurrences in Equations 1 and 4 can be augmented to output, not only the value of $D_X(\emptyset, Z)$, but also the elements of the minimum-cardinality feasible set of subsequence of Y. Then by Claim 3.6, the corresponding sequence of duplicate operations to build Y can be computed in $O(n^2)$ time.

3.3 Computing the Mapping from Z to Y

Suppose we have two strings Z and Y where Z is a subsequence of Y, and we would like to understand the process by which Z evolved or mutated into Y. Also suppose that Y is ambiguous. A particular gene in Z may exhibit multiple *homologs* or copies in Y. Sometimes, we may have biological information about which genes in Y correspond to the original or *exemplar* copy of each gene in Z. That is, sometimes we may have a mapping from the characters of Z to indices in Y. However, sometimes this mapping is not known.

Consider the following strings:

Example 3.10.
$Z = abcd$
$Y = aaabbbcccddd$

In the example, there are 3^4 possible mappings from genes in Z to exemplar genes in Y. If the true mapping is not known, we would like to find the most parsimonious mapping, the one that minimizes duplication distance. We call this mapping *optimal*.

Given a mapping, finding the minimum sequence of duplicate operations necessary to build Y is equivalent to finding the minimum duplication distance $D_X(\emptyset, Y_i)$ for each substring Y_i in $Y \backslash I_Y(\sigma_Z)$, where σ_Z is the subsequence in Y comprised of indices to which characters of Z are mapped and $\emptyset$ denotes the empty string. Note that the sum of the sizes of the substrings in $Y \setminus I_Y(\sigma_Z)$ is bounded by the size of Y.

Let $Z = z_1 \dots z_m$ and $Y = y_1 \dots y_n$ and $m \leq n$. And let $d(Y_{s,t}) = D_X(\emptyset, Y_{s,t})$ as before. We shall also define $f(i,j)$ as the value of the duplication distance $D_X(Z_{i,m}, Y_{j,n})$ of the suffix $z_i z_{i+1} \dots z_m$ of Z and the suffix $y_j y_{j+1} \dots y_n$, given that the character z_i is mapped to the index j in Y. Finally, we shall define the mapping L from indices of Z to indices of Y. We shall represent it as a list. Let $L(i,j)$ be the optimal mapping of the suffix $z_i z_{i+1} \dots z_m$ to indices $j \dots n$ of Y given that character z_i is mapped to index j in Y.

Lemma 3.11. *The values of $f(i,j)$ and $L(i,j)$ satisfy the following recurrence.*

$$f(i,j) = \begin{cases} d(Y_{j+1,n}) & \text{if } i = m, \\ \min_{k \in I_{Y_{j+1,n}}(z_{i+1})} \left(f(i+1,k) + d(Y_{j+1,k-1}) \right) & \text{otherwise.} \end{cases}$$

$$L(i,j) = \begin{cases} (j) & \text{if } i = m, \\ \mathit{prepend}\,(k^*, L(i+1,k^*)) & \text{otherwise.} \end{cases}$$

where

$$k^* = \mathit{argmin}_{k \in I_{Y_{j+1,n}}(z_{i+1})} \left(f(i+1,k) + d(Y_{j+1,k-1}) \right)$$

and

$$\mathit{prepend}\,(x, L) \text{ prepends } x \text{ to the list } L.$$

Proof. Recall that $f(i,j)$ is the value of $D_X(Z_{i,m}, Y_{j,n})$, given that z_i is mapped to index j in Y. We define a base case when $i = m$, i.e. $|Z_{i,m}| = 1$. In this case, since z_m is mapped to index j, the remaining characters of Y need to be inserted and thus $D_X(\emptyset, Y_{j+1,n}) = d(Y_{j+1,n})$.

Now, suppose that $1 \leq i < m$ and $1 \leq j < n$. Let us assume, by induction, that, for all values of k satisfying $j < k \leq n$, the value $f(i+1,k)$ is equal to the value of $D_X(Z_{i+1,m}, Y_{k,n})$. Suppose we want to evaluate $f(i,j)$ when the character z_{i+1} is mapped to index $k > j$ in Y. The characters y_j and y_k do not contribute to the duplication distance $D_X(Z_{i,m}, Y_{j,n})$, however the characters in $Y_{j+1,k-1}$ remain to be inserted via duplicate operations. Therefore, the value of $D_X(Z_{i,m}, Y_{j,n})$ given that character z_i is mapped to index j in Y and z_{i+1} is mapped to index k in Y is given by $f(i+1,k) + d(Y_{j+1,k-1})$. In computing $f(i,j)$, we must consider all possible mappings of z_{i+1}, and

therefore, the value of $f(i,j)$ is computed by taking the minimum of $f(i+1,k) + d(Y_{j+1,k-1})$ over all possible indices k to which z_{i+1} could be mapped in Y. □

The value of $D_X(Z,Y)$ given the best overall mapping of characters of Z to indices in Y is given by $\min_{j \in I_Y(z_1)} (d(Y_{1,j-1}) + f(1,j))$. The corresponding optimal mapping is given by $L(1,j^*)$ where $j^* = argmin_{j \in I_Y(z_1)} (d(Y_{1,j-1}) + f(1,j))$.

Given the value of $D_X(\emptyset, Y_{s,t})$ for all values $1 \leq s \leq t \leq |Y|$, the total time for computing the recurrence in Equation 3.11 is $O(n^3)$. The values of $f(i,j)$ and $L(i,j)$ must be computed for each of the $O(n^2)$ possible combinations of $1 \leq i \leq m$ and $1 \leq j \leq n$ where $m \in O(n)$ because Z is a subsequence of Y. We assume, inductively, that for particular values of i and j, we have already computed $f(i+1,k)$ for all $j < k \leq n$. Therefore, computing $f(i,j)$ and $L(i,j)$ for fixed i and j takes time $O(n)$ because the size of $I_{Y_{j+1,n}}$ is linear in n.

Therefore, as a corollary to Lemma 3.11, we have the following.

Corollary 3.12. *Given the value $d(Y_{s,t}) = D_X(\emptyset, Y_{s,t})$ for all values $1 \leq s \leq t \leq |Y|$, $D_X(Z,Y)$ can be computed in $O(n^3)$ time where $n = |Y|$.*

We conclude the discussion of the algorithm for computing duplication distance with the following summary of our algorithm.

Algorithm 1: Algorithm to Compute Duplication Distance

Input: Strings X, Y,Z where X is non-ambiguous, Z is a subsequence of Y, and $\{Y\} \setminus \{Z\} \subseteq \{X\}$

Output: Value of duplication distance, $D_X(Z,Y)$, and corresponding sequence of duplicate operations, Δ

1 Compute $d(Y_{s,t}) = D_X(\emptyset, Y_{s,t})$ for all $1 \leq s \leq t \leq |Y|$.

2 Compute $D_X(Z,Y)$ and optimal mapping from characters of Z to indices in Y and output S^*, min-cardinality feasible set.

3 Compute sequence of duplicate operations, Δ .

If $n = |Y|$, the first step takes $O(n^4)$ time by Theorem 3.9, the second step takes $O(n^3)$ time by Corollary 3.12, and the third step takes $O(n^2)$ time by Claim 3.6. Therefore, the overall running time is $O(n^4)$. This completes our proof of Theorem 3.1.

3.4 A Note on Duplicate Reversal Operations

In the genome rearrangement literature, the notion of string orientation plays an important role. Many rearrangement operations act on signed strings in which each gene has an orientation, denoted: + or -. Distinguishing between orientations of genes is important when operations allowing substring reversals are considered. Most famously, it has been shown that computing the distance between a pair of unsigned permutations when only substring reversals are allowed is NP-Hard [3], but the reversal distance between signed permutations can be computed in linear time [6], [1].

In the previous sections, we described the duplication distance problem for unsigned strings. However, our algorithm also works in the presence of duplicate reversals on signed strings, i.e. duplicate operations in which the copied substring of X is inverted before being inserted into Z. Now we

shall suppose that our input strings X, Y, and Z are signed strings. We still require that X be non-ambiguous, that Z be a subsequence of Y, and that $\{Y\} \setminus \{Z\} \subseteq \{X\}$ where now $\{X\}$ denotes the the set of all signed characters in X and their complements.

We shall extend the definition of *substring* to include the notion of orientation: a string $S = s_1 s_2 \dots s_m$ is a substring of a string $T = t_1 t_2 \dots t_n$ if there exists an integer j such that either (i) $S = t_j t_{j+1} \dots t_{j+m-1}$ and S has a *forward orientation* or (ii) $S = (-t_j)(-t_{j-1}) \dots (-t_{j-m+1})$ and S has a *backward orientation*. Note that with this new definition of substring, the definitions of a candidate set and a feasible set are extended implicitly; given a string Y and a non-ambiguous string X, a candidate or feasible set of Y is comprised of subsequences of Y that are also substrings of X with either orientation.

We shall also distinguish between duplicate operations in which a reversal occurs and those in which no reversal occurs, denoted by $\delta^{-}_{s,t,p}(X)$ and $\delta^{+}_{s,t,p}(X)$, respectively. For example, if $X = x_1 \dots x_m$ and $Z = z_1 \dots z_n$, then $Z \circ \delta^{-}_{s,t,p}(X) = z_1 \dots z_{p-1}(-x_t) \dots (-x_s) z_p \dots z_n$. $\delta^{+}_{s,t,p}(X)$ is defined as in Definition 2.2.

Given these new definitions, it is not hard to verify that Lemma 3.4 and Claim 3.6 still hold; changing the definition of a substring does not affect the way subsequences of Y overlap. Moreover, the recurrences given in Theorem 3.7 and Lemma 3.8 are still satisfied because the notions of subsequence placement and internal strings do not change. Therefore, our algorithm for computing duplication distance will also correctly compute duplication distance on signed strings in the presence of duplicate reversals.

4 Duplication Distance and the Reversal/Duplication Transposition Distance Problem

The duplicate operation is closely related to another string operation, the duplication transposition [10]. A duplication transposition augments a string G of length n by copying part of G and "pasting" it into another location in the same string G, resulting in a string of length greater than n. A duplication transposition is denoted $\rho_{s,t,p}$, where s and t are the beginning and ending indices of the substring of G to be copied and p is the index into which the beginning of the duplicated substring is to be inserted, satisfying $1 \leq s < t < p \leq n+1$ or $1 \leq p < s < t \leq n$ (see [10], for example). For example, given a string $G = abcdef$, the duplication transposition operation $\rho_{1,3,5}$ on G yields $G \circ \rho_{1,3,5} = abcd\mathbf{abc}ef$.

Definition 4.1 (Duplication Transposition Distance)**.** Given two strings G, T where G is a subsequence of T, the *duplication transposition distance* $DT(G, T)$ is equal to the minimum number of duplication transpositions needed to transform G into T.

Note that $G \circ \rho_{s,t,p} = G \circ \delta_{s,t,p}(G)$ where $\delta_{s,t,p}$ is the duplicate operation (Definition 2.2). However, a duplication transposition requires that s, t, and p satisfy $s < t < p$ or $p < s < t$, whereas a duplicate operation does not have this restriction; a duplicate operation might copy a substring of G ranging between indices s and t and paste it into the middle of that range (Figure 3).

In the next section we describe a problem, formulated by El-Mabrouk in [4], of computing the reversal/duplication transposition distance between two strings. We show that the algorithm presented in [4] does not correctly compute the minimum distance. Moreover, we show that duplication distance can be used in place of duplication transposition distance in the computation.

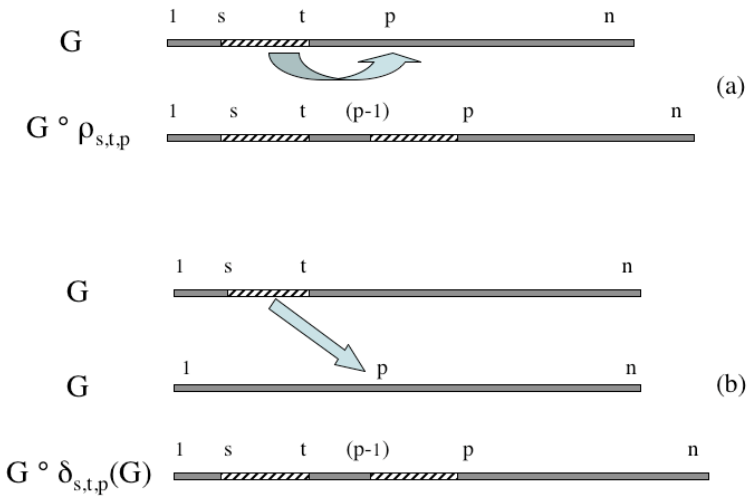

Figure 2: (a) A duplication transposition operation in which a segment of G is copied and pasted into itself. (b) A duplicate operation in which a segment of G is copied and pasted into another, identical string. Both operations result in the same final string.

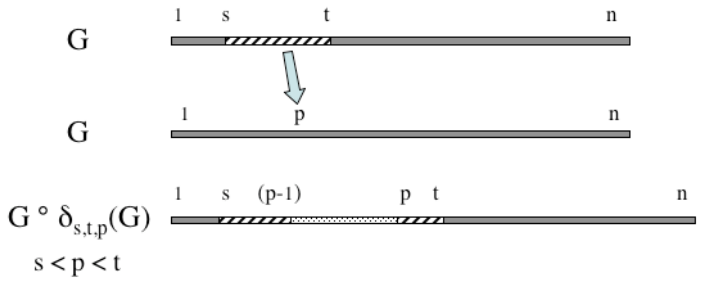

Figure 3: A duplicate operation in which the string from which the substring is copied and the string into which the copied substring is pasted are identical. In this example, $s < p < t$.

4.1 A Discussion of El-Mabrouk's Algorithm

The problem of computing the minimum number of duplication transposition and reversal operations (where a substring of G is inverted) has been studied by El-Mabrouk in [4]. To simplify the problem, El-Mabrouk [4] devises a model of evolution in which she recasts the general problem of computing minimum reversal/duplication transposition distance as an exponential number of polynomially solvable subproblems.

The subproblem considered in [4] is that of computing the minimum reversal/duplication transposition distance between a present-day semi-ambiguous genome G and an ancestral, non-ambiguous genome H. A *semi-ambiguous* genome is an ambiguous string in which no character appears more than twice. It is shown that the minimum reversal/duplication transposition distance of G can be computed by assuming that it evolved from H in two phases. First, by a series of duplication transpositions transformed G into an intermediate ancestor I having the same set of gene blocks (or characters) as G, but not necessarily in the same order. Then, the intermediate ancestor I evolved through a series of reversal operations, resulting in the present-day genome G.

The problem of finding the reversal/duplication transposition distance of a semi-ambiguous genome G then is merely the problem of reconstructing a semi-ambiguous intermediate ancestor I that minimizes the sum of the duplication transposition distance between I and any non-ambiguous ancestor H and the reversal distance between I and G. Moreover, Hannenhalli and Pevzner give a polynomial algorithm for computing the reversal distance between two signed permutations in [6]. Therefore, the remaining work is to compute the duplication transposition distance of the semi-ambiguous intermediate ancestor genome I and the non-ambiguous ancestor genome H.

In [4], the problem of computing the reversal/duplication transposition distance of a semi-ambiguous genome $RD(G)$ is formalized as

$$RD(G) = \min_{I} [R(G, I) + D(I)] \tag{5}$$

where $R(G, I)$ is the reversal distance between G and I, $D(I)$ is defined as the number of maximal, repeated substrings of I, and the minimum is taken over all possible permutations I of the set of characters comprising G.

The algorithm given in [4], therefore, implies that the minimum duplication transposition distance between H and I is equal to the number of maximal repeated substrings of I. However, this simplification is incorrect.

Claim 4.2. *Let H be a non-ambiguous string and let I be a semi-ambiguous string that evolved from H through a series of duplication transpositions. $D(I)$, the number of maximal, repeated substrings of I, is not necessarily equal to the minimum number of duplication transposition operations needed to transform H into I.*

Proof. We provide the following counterexample. Consider the strings:

$$H = abcdefg$$
$$I = ab\mathbf{de}cd\mathbf{bc}efg$$

In this example, the maximal repeated substrings of I are $\{d, e, b, c\}$, so $D(I) = 4$. However, a sequence of three duplication transposition operations would suffice to create I: first, duplicate bc in one operation, then duplicate d and e in two separate operations. □

Note that in the example above, the duplication distance, $D_H(H, I)$ is two. The only cases considered in [4] are cases in which the ancestor genome H is non-ambiguous and the intermediate ancestor I is semi-ambiguous. In these cases, the duplication distance is no greater than the duplication transposition distance.

Claim 4.3. *If Y is semi-ambiguous and X is non-ambiguous, X is a subsequence of Y, and $\{Y\} \setminus \{X\} = \emptyset$ then $D_X(X, Y) \leq DT(X, Y)$.*

Proof. Because Y is semi-ambiguous, no character of X is copied more than once in any sequence of duplication transposition or duplicate operations building Y from X. Therefore, all the strings that are copied in a minimum sequence of either duplication transposition or duplicate operations are substrings of the original input string X.

Consider a minimum-length sequence of duplication transpositions that transforms X into Y. Because all the duplication transpositions in such a sequence copy substrings of the original string X, there are corresponding duplicate operations, copying the same sequence of substrings of X, giving rise to Y. However, as we noted above, the optimal sequence of duplicate operations transforming X into Y may have strictly smaller length than that of the optimal sequence of duplication tranpsosition operations yielding the same Y. Thus, $D_X(X, Y) \leq DT(X, Y)$. □

It follows that duplication distance may provide a simpler explanation than duplication transposition distance in this subproblem of computing the minimum reversal/duplication transposition distance between a non-ambiguous string and a semi-ambiguous string. Moreover, the computation implied in [4] for retrieving the ancestor genome H from the intermediate, semi-ambiguous genome, I, is to delete exactly one copy of every maximal repeated substring of I. However, this may not yield an ancestor genome H such that $DT(H, I)$ is minimal. Consider, the example given in the proof of Claim 4.2. If we had deleted the unbold copies of each of the repeated characters, we would construct the ancestor genome $H' = adebcfg$. In this example, the duplication transposition distance $DT(H', I)$ is equal to four, the number of repeats. However, H' is not the ancestor genome that minimizes the duplication transposition distance. Therefore, the algorithm in [4] not only incorrectly computes the optimal value of the duplication transposition distance between I and some non-ambiguous H, it might also output an ancestor genome that is non-optimal.

Now we return to the reversal/duplication transposition distance problem posed in [4]. Under the assumption that a semi-ambiguous genome G evolved from a non-ambiguous genome H by a series of duplication transpositions followed by a series of reversals, the formulation of Equation 5 can be corrected if we change the definition of $D(I)$ so that it no longer equals the number of repeats in I. Instead, let

$$D(I) = \min_{H \in \mathcal{A}} DT(H, I) \tag{6}$$

where $\mathcal{A}$ is defined as the set of all non-ambiguous subsequences of I that are obtained by deleting exactly one copy of every duplicated gene. Thus, if I has d duplicated genes, $\mathcal{A}$ will contain 2^d elements.

Unfortunately, there is no known algorithm for computing duplication transposition distance. The problem's difficulty results from the fact that the set of substrings that can be duplicated after the k^{th} duplication transposition operation depends on the first k operations. On the other hand, Claim 4.3 states that the duplication distance $D_H(H, I)$ for any $H \in \mathcal{A}$ is a lower bound for the duplication transposition distance.

5 The Amplisome Distance Problem

As mentioned in the introduction, we studied the duplication distance problem primarily as a precursor to computing the amplisome distance between a pair of genomes. We now describe the amplisome distance problem in more detail.

In computing the amplisome distance between a healthy genome G and a tumor genome T, we assume the following model of mutation. The mutation occurs in two phases. In the first phase, called the *copy phase*, an extrachromosomal element, i.e. amplisome, is constructed from a series of duplicate operations in which substrings are copied from G, and the target string is initialized as the empty string. Let the A denote an amplisome. The total number of duplicate operations required to build A from G in the copy phase is $D_G(\emptyset, A)$. At the end of the copy phase, A becomes fixed. In the second phase, the *insertion phase*, G is transformed into T by a sequence of duplicate operations where substrings are copied from A and the target string is initialized as G. The total number of duplicate operations required to build T from G in the insertion phase is $D_A(G, T)$.

The *amplisome distance* between G and T is defined as:

$$AD(G, T) = \min_A \left(D_G(\emptyset, A) + D_A(G, T)\right) \tag{7}$$

where the minimum is taken over all possible amplisome strings A.

It is clear that in order to design an efficient algorithm for the amplisome distance problem, we must first limit the types of strings that can be considered as candidates for the amplisome string A. Moreover, it would be useful to compute some upper and lower bounds on the amplisome distance between two strings and to compute whether, for a particular pair of strings, the amplisome distance will be less than the duplication transposition distance. While this work has not yet been completed, we note here an example of a pair of strings for which amplisome distance is less than duplication transposition distance and, therefore, provides a more compelling model of mutation.

Example 5.1.
$G = \ldots ab \ldots cd \ldots ef \ldots gh \ldots uv \ldots wx \ldots yz \ldots$
$T = \ldots auwb \ldots cwyd \ldots ezxf \ldots gxvh \ldots$

The amplisome distance between G and T in Example 5.1 is seven: first, the amplisome $A = uwyzxv$ can be constructed in three duplicate operations in the copy phase by first copying uv, then inserting wx in between u and v, and then inserting yz in between w and x. The amplisome, A, can be used to construct T from G in four duplicate operations in the insertion phase. However, the duplication distance between G and T is eight.

References

[1] David A. Bader, Bernard M. E. Moret, and Mi Yan. A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. *Lecture Notes in Computer Science*, 2125:365–??, 2000.

[2] Graham R. Bignell, Thomas Santarius, Jessica C.M. Pole, Adam P. Butler, Janet Perry, Erin Pleasance, Chris Greenman, Andrew Menzies, Sheila Taylor, Sarah Edkins, Peter Campbell,

Michael Quail, Bob Plumb, Lucy Matthews, Kirsten McLay, Paul A.W. Edwards, Jane Rogers, Richard Wooster, P. Andrew Futreal, and Michael R. Stratton. Architectures of somatic genomic rearrangement in human cancer amplicons at sequence-level resolution. *Genome Res.*, page gr.6522707, 2007.

[3] Alberto Caprara. Sorting by reversals is difficult. In *RECOMB '97: Proceedings of the first annual international conference on Computational molecular biology*, pages 75–83, New York, NY, USA, 1997. ACM Press.

[4] Nadia El-Mabrouk. Reconstructing an ancestral genome using minimum segments duplications and reversals. *J. Comput. Syst. Sci.*, 65(3):442–464, 2002.

[5] S. Hannenhalli. Polynomial-time algorithm for computing translocation distance between genomes. In Z. Galil and E. Ukkonen, editors, *Proceedings of the 6th Annual Symposium on Combinatorial Pattern Matching*, number 937, pages 162–176, Espoo, Finland, 1995. Springer-Verlag, Berlin.

[6] Sridhar Hannenhalli and Pavel A. Pevzner. Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. *J. ACM*, 46(1):1–27, 1999.

[7] Pavel Pevzner. *Computational molecular biology : an algorithmic approach.* MIT Press, Cambridge, Mass., 2000.

[8] B. J. Raphael and P. A. Pevzner. Reconstructing tumor amplisomes. *Bioinformatics*, 20 Suppl 1:I265–I273, 2004.

[9] G.R. Stark, M. Debatisse, E. Giulotto, and G. M. Wahl. Recent progress in understanding mechanisms of mammalian dna amplification. *Cell*, 57(6):901–908, 1989.

[10] Maria Emilia Telles Walter, Zanoni Dias, and João Meidanis. Reversal and transposition distance of linear chromosomes. In *SPIRE*, pages 96–102, 1998.

[11] B Windle, B W Draper, Y X Yin, S O'Gorman, and G M Wahl. A central role for chromosome breakage in gene amplification, deletion formation, and amplicon integration. *Genes Dev.*, 5(2):160–174, 1991.