Efficient Algorithms for Analyzing Segmental Duplications with Deletions and Inversions in Genomes

Crystal L. Kahn*[1], Shay Mozes *[1] , Benjamin J. Raphael *[1,2]

[1]Department of Computer Science, Brown University, Providence, RI 02912, USA.
[2]Center for Computational Molecular Biology, Brown University, Providence, RI 02912, USA.

Email: Crystal L. Kahn*- clkahn@cs.brown.edu; Shay Mozes*- shay@cs.brown.edu; Benjamin J. Raphael *- braphael@cs.brown.edu;

*Corresponding author

Abstract

Background: Segmental duplications, or low-copy repeats, are common in mammalian genomes. In the human genome, most segmental duplications are mosaics comprised of multiple duplicated fragments. This complex genomic organization complicates analysis of the evolutionary history of these sequences. One model proposed to explain this mosaic patterns is a model of repeated aggregation and subsequent duplication of genomic sequences.

Results: We describe a polynomial-time exact algorithm to compute duplication distance, a genomic distance defined as the most parsimonious way to build a target string by repeatedly copying substrings of a fixed source string. This distance models the process of repeated aggregation and duplication. We also describe extensions of this distance to include certain types of substring deletions and inversions. Finally, we provide an description of a sequence of duplication events as a context-free grammar (CFG).

Conclusion: These new genomic distances will permit more biologically realistic analyses of segmental duplications in genomes.

1 Introduction

Genomes evolve via many types of mutations ranging in scale from single nucleotide mutations to large genome rearrangements. Computational models of these mutational processes allow researchers to derive similarity measures between genome sequences and to reconstruct evolutionary relationships between genomes. For example, considering chromosomal inversions as the only type of mutation leads to the so-called reversal distance problem of finding the minimum number of inversions/reversals that transform one genome into another [1]. Several elegant polynomial-time algorithms have been found to solve this problem (cf. [2] and references therein). Developing genome rearrangement models that are both biologically realistic *and* computationally tractable remains an active area of research.

Duplicated sequences in genomes present a particular challenge for genome rearrangement analysis and often make the underlying computational problems more difficult. For instance, computing reversal distance in genomes with duplicated segments is NP-hard [3]. Moreover, models that include multiple types of mutations – such as inversions and duplications – often result in similarity measures that cannot

be computed efficiently. Current approaches for duplication analysis rely on heuristics, approximation algorithms, or restricted models of duplication [3–7]. For example, there are efficient algorithms for computing tandem duplication histories [8–11] and whole-genome duplication histories [12, 13].

Here we consider another class of duplications: large segmental duplications (also known as low-copy repeats) that are common in many mammalian genomes [14]. These segmental duplications can be quite large (up to hundreds of kilobases), but their evolutionary history remains poorly understood. The mystery surrounding them is due in part to their complex organization; many segmental duplications are mosaic patterns of smaller repeated segments, or *duplicons*. One hypothesis proposed to explain these mosaic patterns is a two-step model of duplication [14]. In this model, a first phase of duplications copies duplicons from the ancestral genome and aggregates these copies into an array of contiguous duplication blocks. Then in a second phase, portions of these duplication blocks are copied and reinserted into the genome at disparate loci forming secondary duplication blocks.

In [15], we introduced a measure called *duplication distance*, which we used in [16] to find the most parsimonious duplication scenario consistent with the two-step model of segmental duplication. The duplication distance from a source string $\mathbf{x}$ to a target string $\mathbf{y}$ is the minimum number of substrings of $\mathbf{x}$ that can be sequentially copied from $\mathbf{x}$ and pasted into an initially empty string in order to construct $\mathbf{y}$. Note that the string $\mathbf{x}$ does not change during the sequence of duplication events. We derived an efficient exact algorithm for computing the duplication distance between a pair of strings. Here, we extend the duplication distance measure to include certain types of deletions and inversions. These extensions make our model less restrictive and permit the construction of more rich, and perhaps more biologically plausible, duplication scenarios. In particular, our contributions are the following.

Summary of Contributions

Let $\mu(\mathbf{x})$ denote the number of times a character appears in the string $\mathbf{x}$. Let $|\mathbf{x}|$ denote the length of $\mathbf{x}$.

1. We provide an $O(|\mathbf{y}|^2|\mathbf{x}|\mu(\mathbf{x})\mu(\mathbf{y}))$-time algorithm to compute the distance between (signed) strings $\mathbf{x}$ and $\mathbf{y}$ when duplication and certain types of deletion operations are permitted.

2. We provide an $O(|\mathbf{y}|^2\mu(\mathbf{x})\mu(\mathbf{y}))$-time algorithm to compute the distance between (signed) strings $\mathbf{x}$ and $\mathbf{y}$ when duplicated strings may be inverted before being inserted into the target string.

3. We provide an $O(|\mathbf{y}|^2|\mathbf{x}|\mu(\mathbf{x})\mu(\mathbf{y}))$-time algorithm to compute the distance between signed strings $\mathbf{x}$ and $\mathbf{y}$ when duplicated strings may be inverted before being inserted into the target string, and

deletion operations are also permitted.

4. We provide an $O(|\mathbf{y}|^2|\mathbf{x}|^3\mu(\mathbf{x})\mu(\mathbf{y}))$-time algorithm to compute the distance between signed strings $\mathbf{x}$ and $\mathbf{y}$ when any substring of the duplicated string may be inverted before being inserted into the target string. Deletion operations are also permitted.

5. We provide a formal proof of correctness of the duplication distance recurrence presented in [16]. No proof of correctness was previously given.

6. We show how a sequence of duplicate operations that generates a string can be described by a context-free grammar (CFG).

2 Preliminaries

We begin by reviewing some definitions and notation that were introduced in [15] and [16]. Let $\emptyset$ denote the empty string. For a string $\mathbf{x} = x_1 \dots x_n$, let $\mathbf{x}_{i,j}$ denote the substring $x_i x_{i+1} \dots x_j$. We define a *subsequence* S of $\mathbf{x}$ to be a string $x_{i_1} x_{i_2} \dots x_{i_k}$ with $i_1 < i_2 < \cdots < i_k$. We represent S by listing the indices at which the characters of S occur in $\mathbf{x}$. For example, if $\mathbf{x} = abcdef$, then the subsequence $S = (1, 3, 5)$ is the string *ace*. Note that every substring is a subsequence, but a subsequence need not be a substring since the characters comprising a subsequence need not be contiguous. For a pair of subsequences S_1, S_2, denote by $S_1 \cap S_2$ the maximal subsequence common to both S_1 and S_2.

Definition 1. *Subsequences $S = (s_1, s_2)$ and $T = (t_1, t_2)$ of a string $\boldsymbol{x}$ are **alternating** in $\boldsymbol{x}$ if either $s_1 < t_1 < s_2 < t_2$ or $t_1 < s_1 < t_2 < s_2$.*

Definition 2. *Subsequences $S = (s_1, \dots, s_k)$ and $T = (t_1, \dots, t_l)$ of a string $\boldsymbol{x}$ are **overlapping** in $\boldsymbol{x}$ if there exist indices i, i' and j, j' such that $1 \leq i < i' \leq k$, $1 \leq j < j' \leq l$, and $(s_i, s_{i'})$ and $(t_j, t_{j'})$ are alternating in $\boldsymbol{x}$. See Figure 1.*

Definition 3. *Given subsequences $S = (s_1, \dots, s_k)$ and $T = (t_1, \dots, t_l)$ of a string $\boldsymbol{x}$, S is **inside** of T if there exists an index i such that $1 \leq i < l$ and $t_i < s_1 < s_k < t_{i+1}$. That is, the entire subsequence S occurs in between successive characters of T. See Figure 2.*

Definition 4. *A **duplicate operation** from $\boldsymbol{x}$, $\delta_{\boldsymbol{x}}(s, t, p)$, copies a substring $x_s \dots x_t$ of the source string $\boldsymbol{x}$ and pastes it into a target string at position p. Specifically, if $\boldsymbol{x} = x_1 \dots x_m$ and $\boldsymbol{z} = z_1 \dots z_n$, then $\boldsymbol{z} \circ \delta_{\boldsymbol{x}}(s, t, p) = z_1 \dots z_{p-1} x_s \dots x_t z_p \dots z_n$. See Figure 3.*

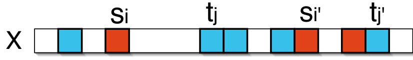

Figure 1: The red subsequence is overlapping with the blue subsequence in $\mathbf{x}$. The indices $(s_i, s_{i'})$ and $(t_j, t_{j'})$ are alternating in $\mathbf{x}$.

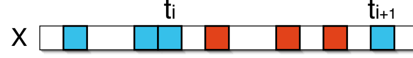

Figure 2: The red subsequence is inside the blue subsequence T. All the characters of the red subsequence occur between the indices t_i and t_{i+1} of T.

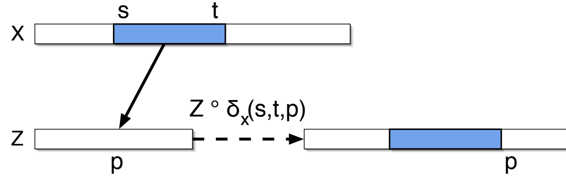

Figure 3: A duplicate operation, $\delta_{\mathbf{x}}(s,t,p)$. A substring $x_s x_{s+1} \ldots x_t$ of the source string $\mathbf{x}$ is copied and inserted into the target string $\mathbf{z}$ at index p.

Definition 5. *The **duplication distance** from a source string $\boldsymbol{x}$ to a target string $\boldsymbol{y}$ is the minimum number of duplicate operations from $\boldsymbol{x}$ that generates $\boldsymbol{y}$ from an initially empty target string* [1]. *That is,* $\boldsymbol{y} = \emptyset \circ \delta_{\boldsymbol{x}}(s_1, t_1, p_1) \circ \delta_{\boldsymbol{x}}(s_2, t_2, p_2) \circ \cdots \circ \delta_{\boldsymbol{x}}(s_l, t_l, p_l)$.

3 Duplication Distance

In this section we review the basic recurrence for computing duplication distance that was introduced in [16]. The recurrence examines the characters of the target string, $\mathbf{y}$, and considers the sets of characters of $\mathbf{y}$ that could have been *generated*, or copied from the source string in a single duplicate operation. Such a set of characters of $\mathbf{y}$ necessarily correspond to a substring of the source $\mathbf{x}$ (see Def .4). Moreover, these characters must be a subsequence of $\mathbf{y}$. This is because, in a sequence of duplicate operations, once a string is copied and inserted into the target string, subsequent duplicate operations do not affect the order of the characters in the previously inserted string. Because every character of $\mathbf{y}$ is generated by exactly one duplicate operation, a sequence of duplicate operations that generates $\mathbf{y}$ partitions the characters of $\mathbf{y}$ into disjoint subsequences, each of which is generated in a single duplicate operation. A more interesting observation is that these subsequences are mutually non-overlapping. We formalize this property as follows.

Lemma 1 (Non-overlapping Property)**.** *Consider a source string $\boldsymbol{x}$ and a sequence of duplicate operations of the form $\delta_{\boldsymbol{x}}(s_i, t_i, p_i)$ that generates the final target string $\boldsymbol{y}$ from an initially empty target string. The substrings x_{s_i,t_i} of $\boldsymbol{x}$ that are duplicated during the construction of $\boldsymbol{y}$ appear as mutually non-overlapping*

[1]We assume that every character in $\mathbf{y}$ appears at least once in $\mathbf{x}$.

subsequences of $\mathbf{y}$*.*

Proof. Consider a sequence of duplicate operations $\delta_{\mathbf{x}}(s_1, t_1, p_1), \ldots, \delta_{\mathbf{x}}(s_k, t_k, p_k)$ that generates $\mathbf{y}$ from an initially empty target string. For $1 \leq i \leq k$, Let Z^i be the intermediate target string that results from $\delta_{\mathbf{x}}(s_1, t_1, p_1) \circ \cdots \circ \delta_{\mathbf{x}}(s_i, t_i, p_i)$. Note that $Z^k = \mathbf{y}$. For $j \leq i$, let S^i_j be the subsequence of Z^i that corresponds to the characters duplicated by the jth operation. We shall show by induction on the length i of the sequence that that $S^i_1, S^i_2, \ldots, S^i_i$ are pairwise non-overlapping subsequences of Z^i. For the base case, when there is a single duplicate operation, there is no non-overlap property to show. Assume now that $S^{i-1}_1, \ldots S^{i-1}_{i-1}$ are mutually non-overlapping subsequences in Z^{i-1}. For the induction step note that, by the definition of a duplicate operation, S_i is inserted as a contiguous substring into Z^{i-1} at location p_i to form Z^i. Therefore, for any $j, j' < i$, if S^{i-1}_j and $S^{i-1}_{j'}$ are non overlapping in Z^{i-1} then S^i_j and $S^i_{j'}$ are non overlapping in Z^i. It remains to show that for any $j < i$ S^i_j and S^i_i are non-overlapping in Z^i. There are two cases: (1) the elements of S^i_j are either all smaller or all greater than the elements of S^i_i or (2) S^i_i is inside of S^i_j in $\mathbf{z}^i$ (Definition 3). In either case, S_j and S_i are not overlapping in $\mathbf{z}^i$ as required. □

The non-overlapping property leads to an efficient recurrence that computes duplication distance. When considering subsequences of the final target string $\mathbf{y}$ that might have been generated in a single duplicate operation, we rely on the non-overlapping property to identify substrings of $\mathbf{y}$ that can be treated as independent subproblems. If we assume that some subsequence S of $\mathbf{y}$ is produced in a single duplicate operation, then we know that all other subsequences of $\mathbf{y}$ that correspond to duplicate operations cannot overlap the characters in S. Therefore, the substrings of $\mathbf{y}$ in between successive characters of S define subproblems that are computed independently.

In order to find the optimal (i.e. minimum) sequence of duplicate operations that can generate $\mathbf{y}$, we must consider all subsequences of $\mathbf{y}$ that could have been generated by a single duplicate operation. The recurrence is based on the observation that y_1 must be the first (i.e. leftmost) character to be copied from $\mathbf{x}$ in some duplicate operation. There are then two cases to consider: either (1) y_1 was the last (or rightmost) character in the substring that was duplicated from $\mathbf{x}$ to generate y_1, or (2) y_1 was not the last character in the substring that was duplicated from $\mathbf{x}$ to generate y_1.

The recurrence defines two quantities: $d(\mathbf{x}, \mathbf{y})$ and $d_i(\mathbf{x}, \mathbf{y})$. We shall show, by induction, that for a pair of strings, $\mathbf{x}$ and $\mathbf{y}$, the value $d(\mathbf{x}, \mathbf{y})$ is equal to the duplication distance from $\mathbf{x}$ to $\mathbf{y}$ and that $d_i(\mathbf{x}, \mathbf{y})$ is equal to the duplication distance from $\mathbf{x}$ to $\mathbf{y}$ under the restriction that the character y_1 is copied from index i in $\mathbf{x}$, i.e. x_i *generates* y_1. $d(\mathbf{x}, \mathbf{y})$ is found by considering the minimum among all characters x_i of $\mathbf{x}$

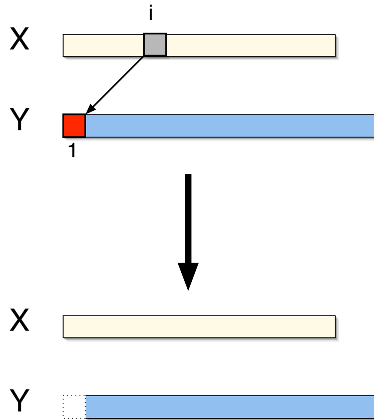

Figure 4: y_1 is generated from x_i in a duplicate operation where y_1 is the last (rightmost) character in the copied substring (Case 1). The total duplication distance is one plus the duplication distance for the suffix $\mathbf{y}_{2,|\mathbf{y}|}$.

that can generate y_1, see Eq. 1.

As described above, we must consider two possibilities in order to compute $d_i(\mathbf{x}, \mathbf{y})$. Either:

Case 1 : y_1 was the last (or rightmost) character in the substring of $\mathbf{x}$ that was copied to produce y_1, (see Fig. 4), or

Case 2 : x_{i+1} is also copied in the same duplicate operation as x_i, possibly along with other characters as well (see Fig. 5).

For case one, the minimum number of duplicate operations is one – for the duplicate that generates y_1 – plus the minimum number of duplicate operations to generate the suffix of $\mathbf{y}$, giving a total of $1 + d(\mathbf{x}, \mathbf{y}_{2,|\mathbf{y}|})$ (Fig. 4). For case two, Lemma 1 implies that the minimum number of duplicate operations is the sum of the optimal numbers of operations for two independent subproblems. Specifically, for each $j > 1$ such that $x_{i+1} = y_j$ we compute: (i) the minimum number of duplicate operations needed to build the substring $\mathbf{y}_{2,j-1}$, namely $d(\mathbf{x}, \mathbf{y}_{2,j-1})$, and (ii) the minimum number of duplicate operations needed to build the string $y_1\mathbf{y}_{j,|\mathbf{y}|}$, given that y_1 is generated by x_i and y_j is generated by x_{i+1}. To compute the latter, recall that since x_i and x_{i+1} are copied in the same duplicate operation, the number of duplicates necessary to generate $y_1\mathbf{y}_{j,|\mathbf{y}|}$ using x_i and x_{i+1} is equal to the number of duplicates necessary to generate $\mathbf{y}_{j,|\mathbf{y}|}$ using x_{i+1}, namely $d_{i+1}(\mathbf{x}, \mathbf{y}_{j,|\mathbf{y}|})$, (see Fig. 5 and Eq. 2).

The recurrence is, therefore:

$$
\begin{aligned}
d(\mathbf{x},\emptyset) &= 0 \\
d(\mathbf{x},\mathbf{y}) &= \min_{\{i:x_i=y_1\}} d_i(\mathbf{x},\mathbf{y}) && (1)\\
d_i(\mathbf{x},\emptyset) &= 0 \\
d_i(\mathbf{x},\mathbf{y}) &= \min\begin{cases} 1+d(\mathbf{x},\mathbf{y}_{2,|\mathbf{y}|}) \\ \min_{\{j:y_j=x_{i+1},j>1\}}\{d(\mathbf{x},\mathbf{y}_{2,j-1})+d_{i+1}(\mathbf{x},\mathbf{y}_{j,|\mathbf{y}|})\}\end{cases} && (2)
\end{aligned}
$$

Theorem 1. *$d(\boldsymbol{x},\boldsymbol{y})$ is the minimum number of duplicate operations that generate $\boldsymbol{y}$ from $\boldsymbol{x}$. For $\{i : x_i = y_1\}$, $d_i(\boldsymbol{x},\boldsymbol{y})$ is the minimum number of duplicate operations that generate $\boldsymbol{y}$ from $\boldsymbol{x}$ such that y_1 is generated by x_i.*

Proof. Let $OPT(\mathbf{x},\mathbf{y})$ denote minimum length of a sequence of duplicate operations that generate $\mathbf{y}$ from $\mathbf{x}$. Let $OPT_i(\mathbf{x},\mathbf{y})$ denote the minimum length of a sequence of operations that generate $\mathbf{y}$ from $\mathbf{x}$ such that y_1 is generated by x_i. We prove by induction on $|\mathbf{y}|$ that $d(\mathbf{x},\mathbf{y}) = OPT(\mathbf{x},\mathbf{y})$ and $d_i(\mathbf{x},\mathbf{y}) = OPT_i(\mathbf{x},\mathbf{y})$. For $|\mathbf{y}| = 1$, since we assume there is at least one i for which $x_i = y_1$, $OPT(\mathbf{x},\mathbf{y}) = OPT_i(\mathbf{x},\mathbf{y}) = 1$. By definition, the recurrence also evaluates to 1. For the inductive step, assume that $OPT(\mathbf{x},\mathbf{y}') = d(\mathbf{x},\mathbf{y}')$ and $OPT_i(\mathbf{x},\mathbf{y}') = d_i(\mathbf{x},\mathbf{y}')$ for any string $\mathbf{y}'$ shorter than $\mathbf{y}$. We first show that $OPT_i(\mathbf{x},\mathbf{y}) \leq d_i(\mathbf{x},\mathbf{y})$. Since $OPT(\mathbf{x},\mathbf{y}) = \min_i OPT_i(\mathbf{x},\mathbf{y})$, this also implies $OPT(\mathbf{x},\mathbf{y}) \leq d(\mathbf{x},\mathbf{y})$. We describe different sequences of duplicate operations that generate $\mathbf{y}$ from $\mathbf{x}$, using x_i to generate y_1:

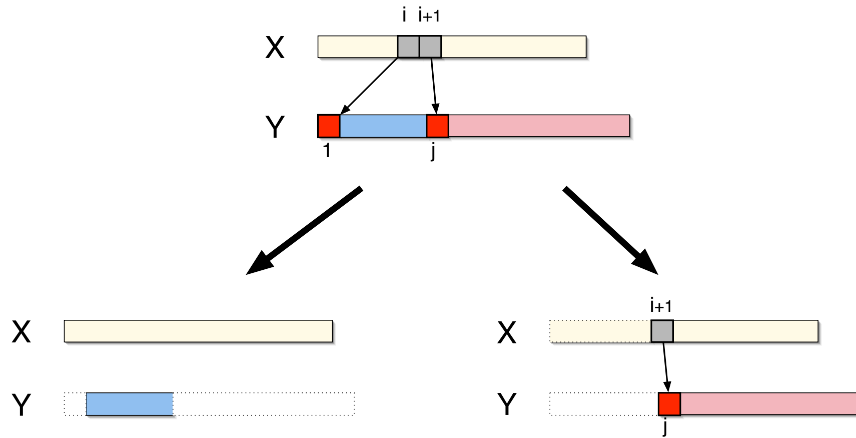

Figure 5: y_1 is generated from x_i in a duplicate operation where y_1 is not the last (rightmost) character in a copied substring (Case 2). In this case, x_{i+1} is also copied in the same duplicate operation (top). Thus, the duplication distance is the sum of $d(\mathbf{x},\mathbf{y}_{2,j-1})$, the duplication distance for $\mathbf{y}_{2,j-1}$ (bottom left), and $d_{i+1}(\mathbf{x},\mathbf{y}_{j,|\mathbf{y}|})$, the minimum number of duplicate operations to generate $\mathbf{y}_{j,|\mathbf{y}|}$ given that x_{i+1} generates y_j (bottom right).

- Consider a minimum-length sequence of duplicates that generates $\mathbf{y}_{2,|\mathbf{y}|}$. By the inductive hypothesis its length is $d(\mathbf{x}, \mathbf{y}_{2,|\mathbf{y}|})$. By duplicating y_1 separately using x_i we obtain a sequence of duplicates that generates $\mathbf{y}$ whose length is $1 + d(\mathbf{x}, \mathbf{y}_{2,|\mathbf{y}|})$.

- For every $\{j : y_j = x_{i+1}, j > 1\}$ consider a minimum-length sequence of duplicates that generates $\mathbf{y}_{j,|\mathbf{y}|}$ using x_{i+1} to produce y_j, and a minimum-length sequence of duplicates that generates $\mathbf{y}_{2,j-1}$. By the inductive hypothesis their lengths are $d_{i+1}(\mathbf{x}, \mathbf{y}_{j,|\mathbf{y}|})$ and $d(\mathbf{x}, \mathbf{y}_{2,j-1})$ respectively. By extending the start index s of the duplicate operation that starts with x_{i+1} to produce y_j to start with x_i and produce y_1 as well, we can produce $\mathbf{y}$ with the same number of duplicate operations.

Since $OPT_i(\mathbf{x}, \mathbf{y})$ is at most the length of any of these options, it is also at most their minimum. Hence,

$$\begin{aligned} OPT_i(\mathbf{x}, \mathbf{y}) &\leq \min \begin{cases} 1 + d(\mathbf{x}, \mathbf{y}_{2,|\mathbf{y}|}) \\ \min_{\{j : y_j = x_{i+1}, j > 1\}} \{d(\mathbf{x}, \mathbf{y}_{2,j-1}) + d_{i+1}(\mathbf{x}, \mathbf{y}_{j,|\mathbf{y}|})\} \end{cases} \\ &= d_i(\mathbf{x}, \mathbf{y}). \end{aligned}$$

To show the other direction (i.e. that $d(x, y) \leq OPT(x, y)$ and $d_i(x, y) \leq OPT_i(x, y)$), consider a minimum-length sequence of duplicate operations that generate $\mathbf{y}$ from $\mathbf{x}$, using x_i to generate y_1. There are a few cases:

- If y_1 is generated by a duplicate operation that only duplicates x_i, then $OPT_i(\mathbf{x}, \mathbf{y}) = 1 + OPT(\mathbf{x}, \mathbf{y}_{2,|\mathbf{y}|})$. By the inductive hypothesis this equals $1 + d(\mathbf{x}, \mathbf{y}_{2,|\mathbf{y}|})$ which is at least $d_i(\mathbf{x}, \mathbf{y})$.

- Otherwise, y_1 is generated by a duplicate operation that copies x_i and also duplicates x_{i+1} to generate some character y_j. In this case the sequence Δ of duplicates that generates $\mathbf{y}_{2,j-1}$ must appear after the duplicate operation that generates y_1 and y_j because $\mathbf{y}_{2,j-1}$ is inside (Definition 3) of (y_1, y_j). Without loss of generality, suppose Δ is ordered after all the other duplicates so that first $y_1 y_j \ldots y_{|\mathbf{y}|}$ is generated, and then Δ generates $y_2 \ldots y_{j-1}$ between y_1 and y_j. Hence, $OPT_i(\mathbf{x}, \mathbf{y}) = OPT_i(\mathbf{x}, y_1 \mathbf{y}_{j,|\mathbf{y}|}) + OPT(\mathbf{x}, \mathbf{y}_{2,j-1})$. Since in the optimal sequence x_i generates y_1 in the same duplicate operation that generates y_j from x_{i+1}, we have $OPT_i(\mathbf{x}, y_1 \mathbf{y}_{j,|\mathbf{y}|}) = OPT_{i+1}(\mathbf{x}, \mathbf{y}_{j,|\mathbf{y}|})$. By the inductive hypothesis, $OPT(\mathbf{x}, \mathbf{y}_{2,j-1}) + OPT_{i+1}(\mathbf{x}, \mathbf{y}_{j,|\mathbf{y}|}) = d(\mathbf{x}, \mathbf{y}_{2,j-1}) + d_{i+1}(\mathbf{x}, \mathbf{y}_{j,|\mathbf{y}|})$ which is at least $d_i(\mathbf{x}, \mathbf{y})$.

□

This recurrence naturally translates into a dynamic programing algorithm that computes the values of $d(\mathbf{x}, \cdot)$ and $d_i(\mathbf{x}, \cdot)$ for various target strings. To analyze the running time of this algorithm, note that both $\mathbf{y}_{2,j}$ and $\mathbf{y}_{j,|\mathbf{y}|}$ are substrings of $\mathbf{y}$. Since the set of substrings of $\mathbf{y}$ is closed under taking substrings, we only encounter substrings of $\mathbf{y}$. Also note that since i is chosen from the set $\{i : x_i = y_1\}$, there are $O(\mu(\mathbf{x}))$ choices for i, where $\mu(\mathbf{x})$ is the maximal multiplicity of a character in $\mathbf{x}$. Thus, there are $O(\mu(\mathbf{x})|\mathbf{y}|^2)$ different values to compute. Each value is computed by considering the minimization over at most $\mu(\mathbf{y})$ previously computed values, so the total running time is bounded by $O(|\mathbf{y}|^2\mu(\mathbf{x})\mu(\mathbf{y}))$, which is $O(|\mathbf{y}|^3|\mathbf{x}|)$ in the worst case. As with most dynamic programming approaches, our algorithm can be extended to reconstruct the optimal sequence of duplicate operations needed to build $\mathbf{y}$. We omit the details.

Extending to Affine Duplication Cost

It is easy to extend the recurrence relations in Eqs. (1), (2) to handle costs for duplicate operations. In the above discussion, the cost of each duplicate operation is 1, so the sum of costs of the operations in a sequence that generates a string $\mathbf{y}$ is just the length of that sequence. We next consider a more general cost model for duplication in which the cost of a duplicate operation $\delta_x(s, t, p)$ is $\Delta_1 + (t - s + 1)\Delta_2$ (i.e., the cost is affine in the number of duplicated characters). Here Δ_1, Δ_2 are some non-negative constants. This extension is obtained by assigning a cost of Δ_2 to each duplicated character, except for the last character in the duplicated string, which is assigned a cost of $\Delta_1 + \Delta_2$. We do that by adding a cost term to each of the cases in Eq. 2. If x_i is the last character in the duplicated string (case 1), we add $\Delta_1 + \Delta_2$ to the cost. Otherwise x_i is not the last duplicated character (case 2), so we add just Δ_2 to the cost. Eq. (2) thus becomes

$$d_i(\mathbf{x}, \mathbf{y}) \quad = \quad \min \begin{cases} \Delta_1 + \Delta_2 + d(\mathbf{x}, \mathbf{y}_{2,|\mathbf{y}|}) \\ \min_{\{j : y_j = x_{i+1}, j > 1\}} \{d(\mathbf{x}, \mathbf{y}_{2,j-1}) + d_{i+1}(\mathbf{x}, \mathbf{y}_{j,|\mathbf{y}|}) + \Delta_2\} \end{cases} \tag{3}$$

The running time analysis for this recurrence is the same as for the one with unit duplication cost.

4 Duplication-Deletion Distance

In this section we generalize the model to include deletions. Consider the intermediate string $\mathbf{z}$ generated after some number of duplicate operations. A deletion operation removes a contiguous substring $z_i, \ldots, z_j$ of $\mathbf{z}$, and subsequent duplicate and deletion operations are applied to the resulting string.

Definition 6. *A **delete operation**, $\tau(s, t)$, deletes a substring $z_s \ldots z_t$ of the target string $\boldsymbol{z}$, thus making $\boldsymbol{z}$ shorter. Specifically, if $\boldsymbol{z} = z_1 \ldots z_s \ldots z_t \ldots z_m$, then $\boldsymbol{z} \circ \tau(s, t) = z_1 \ldots z_{s-1} z_{t+1} \ldots z_m$. See Figure 6.*

The cost associated with $\tau(s,t)$ depends on the number $t-s+1$ of characters deleted and is denoted $\Phi(t-s+1)$.

Figure 6: A delete operation, $\tau(s,t)$. The substring $\mathbf{z}_{s,t}$ is deleted.

Definition 7. *The **duplication-deletion distance** from a source string $\boldsymbol{x}$ to a target string $\boldsymbol{y}$ is the cost of a minimum sequence of duplicate operations from $\boldsymbol{x}$ and deletion operations, in any order, that generates $\boldsymbol{y}$.*

We now show that although we allow arbitrary deletions from the intermediate string, it suffices to consider deletions from the duplicated strings before they are pasted into the intermediate string, provided that the cost function for deletion, $\Phi(\cdot)$ is non-decreasing and obeys the triangle inequality.

Definition 8. *A **duplicate-delete** operation from $\boldsymbol{x}$, $\eta_{\boldsymbol{x}}(i_1, j_1, i_2, j_2, \ldots, i_k, j_k, p)$, for $i_1 \leq j_1 < i_2 \leq j_2 < \cdots < i_k \leq j_k$ copies the subsequence $x_{i_1} \ldots x_{j_1} x_{i_2} \ldots x_{j_2} \ldots \ldots x_{i_k} \ldots x_{j_k}$ of the source string $\boldsymbol{x}$ and pastes it into a target string at position p. Specifically, if $\boldsymbol{x} = x_1 \ldots x_m$ and $\boldsymbol{z} = z_1 \ldots z_n$, then $\boldsymbol{z} \circ \eta_{\boldsymbol{x}}(i_1, j_1, \ldots, i_k, j_k, p) = z_1 \ldots z_{p-1} x_{i_1} \ldots x_{j_1} x_{i_2} \ldots x_{j_2} \ldots x_{i_k} \ldots x_{j_k} z_p \ldots z_n$.*

The cost associated with such a duplication-deletion is $\Delta_1 + (j_k - i_1 + 1)\Delta_2 + \sum_{\ell=1}^{k-1} \Phi(i_{\ell+1} - j_\ell - 1)$. The first two terms in the cost reflect the affine cost of duplicating an entire substring of length $j_k - i_1 + 1$, and the second term reflects the cost of deletions made to that substrings.

Lemma 2. *If the affine cost for duplications is non-decreasing and $\Phi(\cdot)$ is non-decreasing and obeys the triangle inequality then the cost of a minimum sequence of duplicate and delete operations that generates a target string $\boldsymbol{y}$ from a source string $\boldsymbol{x}$ is equal to the cost of a minimum sequence of duplicate-delete operations that generates $\boldsymbol{y}$ from $\boldsymbol{x}$.*

Proof. Since duplicate operations are a special case of duplicate-delete operations, the cost of a minimal sequence of duplicate-delete operations and delete operations that generates $\mathbf{y}$ cannot be more than that of a sequence of just duplicate operations and delete operations. We show the (stronger) claim that an arbitrary sequence of duplicate-delete and delete operations that produces a string $\mathbf{y}$ with cost c can be transformed into a sequence of just duplicate-delete operations that generates $\mathbf{y}$ with cost at most c by induction on the number of delete operations. The base case, where the number of deletions is zero, is

trivial. Consider the first delete operation, τ. Let k denote the number of duplicate-delete operations that precede τ, and let $\mathbf{z}$ be the intermediate string produced by these k operations. For $i = 1, \ldots, k$, let S_i be the subsequence of $\mathbf{x}$ that was used in the ith duplicate-delete operation. By lemma 1, $S_1, \ldots, S_k$ form a partition of $\mathbf{z}$ into disjoint, non-overlapping subsequences of $\mathbf{z}$. Let d denote the substring of $\mathbf{z}$ to be deleted. Since d is a contiguous substring, $S_i \cap d$ is a (possibly empty) substring of S_i for each i. There are several cases:

1. $S_i \cap d = \emptyset$. In this case we do not change any operation.

2. $S_i \cap d = S_i$. In this case all characters produced by the ith duplicate-delete operation are deleted, so we may omit the ith operation altogether and decrease the number of characters deleted by τ. Since $\Phi(\cdot)$ is non-decreasing, this generates $\mathbf{z}$ (and hence $\mathbf{y}$) with a lower cost.

3. $S_i \cap d$ is a prefix (or suffix) of S_i. Assume it is a prefix. The case of suffix is similar. Instead of deleting the characters $S_i \cap d$ we can avoid generating them in the first place. Let r be the smallest index in $S_i \setminus d$ (that is, the first character in S_i that is not deleted by τ). We change the ith duplicate-delete operation to start at r and decrease the number of characters deleted by τ. Since the affine cost for duplications is non-decreasing and $\Phi(\cdot)$ is non-decreasing, the cost of generating $\mathbf{z}$ may only decrease.

4. $S_i \cap d$ is a non-empty substring of S_i that is neither a prefix nor a suffix of S_i. We claim that this case applies to at most one value of i. This implies that after taking care of all the other cases τ only deletes characters in S_i. We can then change the ith duplicate-delete operation to also delete the characters deleted by τ, and omit τ. Since $\Phi(\cdot)$ obeys the triangle inequality, this will not increase the total cost of deletion. By the inductive hypothesis, the rest of $\mathbf{y}$ can be generated by just duplicate-delete operations with at most the same cost. It remains to prove the claim. Recall that the set $\{Si\}$ is comprised of mutually non-overlapping subsequences of $\mathbf{z}$. Suppose that there exist indices $i \neq j$ such that $S_i \cap d$ is a non-prefix/suffix substring of S_i and $S_j \cap d$ is a non-prefix/suffix substring of S_j. There must exist indices of both S_i and S_j in $\mathbf{z}$ that precede d, are contained in d, and succeed d. Let $i_p < i_c < i_s$ be three such indices of S_i and let $j_p < j_c < j_s$ be similar for S_j. It must be the case also that $j_p < i_c < j_s$ and $i_p < j_c < i_s$. Without loss of generality, suppose $i_p < j_p$. It follows that (i_p, i_c) and (j_p, j_s) are alternating in $\mathbf{z}$. So, S_i and S_j are overlapping which contradicts Lemma 1.

□

To extend the recurrence from the previous section to duplication-deletion distance, we must observe that because we allow deletions in the string that is duplicated from $\mathbf{x}$, if we assume character x_i is copied to produce y_1, it may not be the case that the character x_{i+1} also appears in $\mathbf{y}$; the character x_{i+1} may have been deleted. Therefore, we minimize over all possible locations $k > i$ for the next character in the duplicated string that is not deleted. The extension of the recurrence from the previous section to duplication-deletion distance is:

$$\hat{d}(\mathbf{x}, \emptyset) = 0 \quad , \quad \hat{d}(\mathbf{x}, \mathbf{y}) = \min_{\{i: x_i = y_1\}} \hat{d}_i(\mathbf{x}, \mathbf{y}), \tag{4}$$

$$\hat{d}_i(\mathbf{x}, \emptyset) = 0 \quad ,$$

$$\hat{d}_i(\mathbf{x}, \mathbf{y}) = \min \begin{cases} \Delta_1 + \Delta_2 + \hat{d}(\mathbf{x}, \mathbf{y}_{2,|\mathbf{y}|}), \\ \min_{k>i} \min_{\{j: y_j = x_k, j>1\}} \begin{Bmatrix} \hat{d}(\mathbf{x}, \mathbf{y}_{2,j-1}) + \hat{d}_k(\mathbf{x}, \mathbf{y}_{j,|\mathbf{y}|}) \\ (k-i)\Delta_2 + \Phi(k-i-1) \end{Bmatrix}. \end{cases} \tag{5}$$

Theorem 2. *$\hat{d}(\boldsymbol{x}, \boldsymbol{y})$ is the duplication-deletion distance from $\boldsymbol{x}$ to $\boldsymbol{y}$. For $\{i : x_i = y_1\}$, $\hat{d}_i(\boldsymbol{x}, \boldsymbol{y})$ is the duplication-deletion distance from $\boldsymbol{x}$ to $\boldsymbol{y}$ under the additional restriction that y_1 is generated by x_i.*

The proof of Theorem 2 is almost identical to that of Theorem 1 in the previous section and is omitted. However, the running time increases; while the number of entries in the dynamic programming table does not change, the time to compute each entry is multiplied by the possible values of k in the recurrence, which is $O(|\mathbf{x}|)$. Therefore, the running time is $O(|\mathbf{y}|^2|\mathbf{x}|\mu(\mathbf{x})\mu(\mathbf{y}))$, which is $O(|\mathbf{y}|^3|\mathbf{x}|^2)$ in the worst case. We conclude this section by showing, in the following lemma, that if both the duplicate and delete cost functions are the identity function (i.e. one per operation), then the duplication-deletion distance is equal to duplication distance without deletions.

Lemma 3. *Given a source string $\boldsymbol{x}$, a target string $\boldsymbol{y}$, If the cost of duplication is 1 per duplicate operation, and the cost of deletion is 1 per delete operation, then $\hat{d}(\boldsymbol{x}, \boldsymbol{y}) = d(\boldsymbol{x}, \boldsymbol{y})$.*

Proof. First we note that if a target string $\mathbf{y}$ can be built from $\mathbf{x}$ in $d(\mathbf{x}, \mathbf{y})$ duplicate operations, then the same sequence of duplicate operations is a valid sequence of duplicate and delete operations as well, so $d(\mathbf{x}, \mathbf{y})$ is at least $\hat{d}(\mathbf{x}, \mathbf{y})$.

We claim that every sequence of duplicate and delete operations can be transformed into a sequence of duplicate operations of the same length. The proof of this claim is similar to that of Lemma 2. In that proof we showed how to transform a sequence of duplicate and delete operations into a sequence of

duplicate-delete operations of at most the same cost. We follow the same steps, but transform the sequence into an a sequence that consists of just duplicate operations without increasing the number of operations. Recall the four cases in the proof of Lemma 2. In the the first three cases we eliminate the delete operation without increasing the number of duplicate operations. Therefore we only need to consider the last case ($S_i \cap d$ is a non-empty substring of S_i that is neither a prefix nor a suffix of S_i). Recall that this case applies to at most one value of i. Deleting $S_i \cap d$ from S_i leaves a prefix and a suffix of S_i. We can therefore replace the ith duplicate operation and the delete operation with two duplicate operations, one generating the appropriate prefix of S_i and the other generating the appropriate suffix of S_i. This eliminates the delete operation without changing the number of operations in the sequence. Therefore, for any string $\mathbf{y}$ that results from a sequence of duplicate and delete operations, we can construct the same string using only duplicate operations (without deletes) using at most the same number of operations. So, $d(\mathbf{x}, \mathbf{y})$ is no greater than $\hat{d}(\mathbf{x}, \mathbf{y})$. □

As in most dynamic programming approaches, our algorithm to compute duplication-deletion distance can be extended also to reconstruct (through trace-back) the optimal sequence of operations necessary to build a given string.

5 Duplication-Inversion Distance

In this section we extend the duplication-deletion distance recurrence to allow inversions. We now explicitly define characters and strings as having two orientations: forward (+) and inverse (-).

Definition 9. *A* ***signed string*** *of length m over an alphabet Σ is an element of* $(\{+,-\} \times \Sigma)^m$.

For example, $(+b - c - a + d)$ is a signed string of length 4. An inversion of a signed string reverses the order of the characters as well as their signs. Formally,

Definition 10. *The* ***inverse*** *of a signed string $\boldsymbol{x} = x_1 \ldots x_m$ is a signed string $\bar{\boldsymbol{x}} = -x_m \cdots - x_1$.*

For example, the inverse of $(+b - c - a + d)$ is $(-d + a + c - b)$.

In a duplicate-invert operation a substring is copied from $\mathbf{x}$ and *inverted* before being inserted into the target string $\mathbf{y}$. We allow the cost of inversion to be an affine function in the length ℓ of the duplicated inverted string, which we denote $\Theta_1 + \ell\Theta_2$, where $\Theta_1, \Theta_2 \geq 0$. We still allow for normal duplicate operations.

Definition 11. *A* ***duplicate-invert operation*** *from* $\boldsymbol{x}$*,* $\bar{\delta}_{\boldsymbol{x}}(s,t,p)$*, copies an inverted substring* $-x_t,\ldots,-x_s$ *of the source string* $\boldsymbol{x}$ *and pastes it into a target string at position p. Specifically, if* $\boldsymbol{x} = x_1 \ldots x_m$ *and* $\boldsymbol{z} = z_1 \ldots z_n$*, then* $\boldsymbol{z} \circ \bar{\delta}_{\boldsymbol{x}}(s,t,p) = z_1 \ldots z_{p-1}\bar{x}_t\bar{x}_{t-1} \ldots \bar{x}_s z_p \ldots z_n$*.*

The cost associated with each duplicate-invert operation is $\Theta_1 + (t-s+1)\Theta_2$.

Definition 12. *The* ***duplication-inversion distance*** *from a source string* $\boldsymbol{x}$ *to a target string* $\boldsymbol{y}$ *is the cost of a minimum sequence of duplicate and duplicate-invert operations from* $\boldsymbol{x}$*, in any order, that generates* $\boldsymbol{y}$*.*

The recurrence for duplication distance (Eqs. 1, 3) can be extended to compute the duplication-inversion distance. This is done by introducing a term for inverted duplications whose form is very similar to that of the term for regular duplication (Eq. 3). Specifically, when considering the possible characters to generate y_1, we consider characters in $\mathbf{x}$ that match either y_1 or its inverse, $-y_1$. In the former case, then, we use $\bar{d}_i^+(\mathbf{x},\mathbf{y})$ to denote the duplication-inversion distance with the additional restriction that y_1 is generated by x_i without an inversion. The recurrence for $\bar{d}_i^+$ is the same as for d_i in Eq. 3. In the latter case, we consider an inverted duplicate in which y_1 is generated by $-x_i$. This is denoted by $\bar{d}_i^-$, which follows a similar recurrence. In this recurrence, since an inversion occurs, x_i is the *last* character of the duplicated string, rather than the first one. Therefore, the next character in $\mathbf{x}$ to be used in this operation is $-x_{i-1}$ rather than x_{i+1}. The recurrence for $\bar{d}_i^-$ also differs in the cost term, where we use the affine cost of the duplicate-invert operation. The extension of the recurrence to duplication-inversion distance is therefore:

$$
\begin{aligned}
\bar{d}(\mathbf{x},\emptyset) &= 0 \quad , \quad \bar{d}(\mathbf{x},\mathbf{y}) = \min\left\{ \min_{\{i:x_i=y_1\}} \bar{d}_i^+(\mathbf{x},\mathbf{y}), \min_{\{i:x_i=-y_1\}} \bar{d}_i^-(\mathbf{x},\mathbf{y}) \right\}, \\
\bar{d}_i^+(\mathbf{x},\emptyset) &= 0 \quad , \quad \bar{d}_i^-(\mathbf{x},\emptyset) = 0, \\
\bar{d}_i^+(\mathbf{x},\mathbf{y}) &= \min\begin{cases} \Delta_1 + \Delta_2 + \bar{d}(\mathbf{x},\mathbf{y}_{2,|\mathbf{y}|}), \\ \min_{\{j:y_j=x_{i+1},j>1\}} \left\{ \bar{d}(\mathbf{x},\mathbf{y}_{2,j-1}) + \bar{d}_{i+1}^+(\mathbf{x},\mathbf{y}_{j,|\mathbf{y}|}) + \Delta_2 \right\}, \end{cases} \\
\bar{d}_i^-(\mathbf{x},\mathbf{y}) &= \min\begin{cases} \Theta_1 + \Theta_2 + \bar{d}(\mathbf{x},\mathbf{y}_{2,|\mathbf{y}|}), \\ \min_{\{j:y_j=-x_{i-1},j>1\}} \left\{ \bar{d}(\mathbf{x},\mathbf{y}_{2,j-1}) + \bar{d}_{i-1}^-(\mathbf{x},\mathbf{y}_{j,|\mathbf{y}|}) + \Theta_2 \right\}. \end{cases}
\end{aligned}
\tag{6}
$$

Theorem 3. $\bar{d}(\boldsymbol{x},\boldsymbol{y})$ *is the duplication-inversion distance from* $\boldsymbol{x}$ *to* $\boldsymbol{y}$*. For* $\{i : x_i = y_1\}$*,* $\bar{d}_i^+(\boldsymbol{x},\boldsymbol{y})$ *is the duplication-inversion distance from* $\boldsymbol{x}$ *to* $\boldsymbol{y}$ *under the additional restriction that* y_1 *is generated by* x_i*. For* $\{i : x_i = -y_1\}$*,* $\bar{d}_i^-(\boldsymbol{x},\boldsymbol{y})$ *is the duplication-inversion distance from* $\boldsymbol{x}$ *to* $\boldsymbol{y}$ *under the additional restriction that* y_1 *is generated by* $-x_i$*.*

The correctness proof is very similar to that of Theorem 1, only requiring an additional case for handling the case of a duplicate invert operation which is symmetric to the case of regular duplication. The asymptotic running time of the corresponding dynamic programming algorithm is $O(|\mathbf{y}|^2\mu(\mathbf{x})\mu(\mathbf{y}))$. The analysis is identical to the one in section 3. The fact that we now consider either a duplicate or a duplicate-invert operation does not change the asymptotic running time.

6 Duplication-Inversion-Deletion Distance

In this section we extend the distance measure to include delete operations as well as duplicate and duplicate-invert operations. Note that we only handle deletions after inversions of the same substring. The order of operations might be important, at least in terms of costs. The cost of inverting $(+a+b+c)$ and then deleting $-b$ may be different than the cost of first deleting $+b$ from $(+a+b+c)$ and then inverting $(+a+c)$.

Definition 13. *The* ***duplication-inversion-deletion distance*** *from a source string* $\boldsymbol{x}$ *to a target string* $\boldsymbol{y}$ *is the cost of a minimum sequence of duplicate and duplicate-invert operations from* $\boldsymbol{x}$ *and deletion operations, in any order, that generates* $\boldsymbol{y}$*.*

Definition 14. *A* ***duplicate-invert-delete*** *operation from* $\boldsymbol{x}$*,* $\bar{\eta}_{\boldsymbol{x}}(i_1, j_1, i_2, j_2, \ldots, i_k, j_k, p)$*, for* $i_1 \leq j_1 < i_2 \leq j_2 < \cdots < i_k \leq j_k$ *pastes the string* $-x_{j_k} - x_{j_k-1} \cdots - x_{i_k} - x_{j_{k-1}} - x_{j_{k-1}-1} \cdots - x_{i_{k-1}} \cdots \cdots - x_{j_1} - x_{j_1-1} \cdots - x_{i_1}$ *into a target string at position* p*. Specifically, if* $\boldsymbol{x} = x_1 \ldots x_m$ *and* $\boldsymbol{z} = z_1 \ldots z_n$*, then* $\boldsymbol{z} \circ \bar{\eta}_{\boldsymbol{x}}(i_1, j_1, i_2, j_2, \ldots, i_k, j_k, p) =$ $z_1 \ldots z_{p-1} - x_{j_k} - x_{j_k-1} \cdots - x_{i_k} - x_{j_{k-1}} - x_{j_{k-1}-1} \cdots - x_{i_{k-1}} \cdots \cdots - x_{j_1} - x_{j_i-1} \cdots - x_{i_1} z_p \ldots z_n$*.*

The cost of such an operation is $\Theta_1 + (j_k - i_1 + 1)\Theta_2 + \sum_{\ell=1}^{k-1} \Phi(i_{\ell+1} - j_\ell - 1)$. Similar to the previous section, it suffices to consider just duplicate-invert-delete and duplicate-delete operations, rather than duplicate, duplicate-invert and delete operations.

Lemma 4. *If* $\Phi(\cdot)$ *is non-decreasing and obeys the triangle inequality and if the cost of inversion is an affine non-decreasing function as defined above, then the cost of a minimum sequence of duplicate, duplicate-invert and delete operations that generates a target string* $\boldsymbol{y}$ *from a source string* $\boldsymbol{x}$ *is equal to the cost of a minimum sequence of duplicate-delete and duplicate-invert-delete operations that generates* $\boldsymbol{y}$ *from* $\boldsymbol{x}$*.*

The proof of the lemma is essentially the same as that of Lemma 2. Note that in that proof we did not

require all duplicate operations to be from the same string $\mathbf{x}$. Therefore, the arguments in that proof apply to our case, where we can regard some of the duplicates from $\mathbf{x}$ and some from the inverse of $\mathbf{x}$.

The recurrence for duplication-inversion-deletion distance is obtained by combining the recurrences for duplication-deletion (Eq. 5) and for duplication-inversion distance (Eq. 6). We use separate terms for duplicate-delete operations ($\hat{\hat{d}}_i^+$) and for duplicate-invert-delete operations ($\hat{\hat{d}}_i^-$). Those terms differ from the terms in Eq. 6 in the same way Eq. 5 differs from Eq. 2; Because of the possible deletion we do not know that x_{i+1} (x_{i-1}) is the next duplicated character. Instead we minimize over all characters later (earlier) than x_i.

The recurrence for duplication-inversion-deletion distance is therefore:

$$
\begin{aligned}
\hat{\hat{d}}(\mathbf{x},\emptyset) &= 0 \quad , \quad \hat{\hat{d}}(\mathbf{x},\mathbf{y}) = \min\left\{ \min_{\{i:x_i=y_1\}} \hat{\hat{d}}_i^+(\mathbf{x},\mathbf{y}), \min_{\{i:x_i=-y_1\}} \hat{\hat{d}}_i^-(\mathbf{x},\mathbf{y}) \right\}, \\
\hat{\hat{d}}_i^+(\mathbf{x},\emptyset) &= 0 \quad , \quad \hat{\hat{d}}_i^-(\mathbf{x},\emptyset) = 0, \\
\hat{\hat{d}}_i^+(\mathbf{x},\mathbf{y}) &= \min \begin{cases} \Delta_1 + \Delta_2 + \hat{\hat{d}}(\mathbf{x},\mathbf{y}_{2,|\mathbf{y}|}), \\ \min_{k>i} \min_{\{j:y_j=x_k,j>1\}} \left\{ \begin{array}{l} \hat{\hat{d}}(\mathbf{x},\mathbf{y}_{2,j-1}) + \hat{\hat{d}}_k^+(\mathbf{x},\mathbf{y}_{j,|\mathbf{y}|}) \\ (k-i)\Delta_2 + \Phi(k-i-1) \end{array} \right\} \end{cases}, \\
\hat{\hat{d}}_i^-(\mathbf{x},\mathbf{y}) &= \min \begin{cases} \Theta_1 + \Theta_2 + \hat{\hat{d}}(\mathbf{x},\mathbf{y}_{2,|\mathbf{y}|}), \\ \min_{k<i} \min_{\{j:y_j=-x_k,j>1\}} \left\{ \begin{array}{l} \hat{\hat{d}}(\mathbf{x},\mathbf{y}_{2,j-1}) + \hat{\hat{d}}_k^-(\mathbf{x},\mathbf{y}_{j,|\mathbf{y}|}) \\ +(i-k)\Theta_2 + \Phi(i-k-1) \end{array} \right\} \end{cases}.
\end{aligned}
$$

Theorem 4. *$\hat{\hat{d}}(\boldsymbol{x},\boldsymbol{y})$ is the duplication-inversion-deletion distance from $\boldsymbol{x}$ to $\boldsymbol{y}$. For $\{i : x_i = y_1\}$, $\hat{\hat{d}}_i^+(\boldsymbol{x},\boldsymbol{y})$ is the duplication-inversion-deletion distance from $\boldsymbol{x}$ to $\boldsymbol{y}$ under the additional restriction that y_1 is generated by x_i. For $\{i : x_i = -y_1\}$, $\hat{\hat{d}}_i^-(\boldsymbol{x},\boldsymbol{y})$ is the duplication-inversion-deletion distance from $\boldsymbol{x}$ to $\boldsymbol{y}$ under the additional restriction that y_1 is generated by $-x_i$.*

The proof, again, is very similar to the proofs in the previous sections. The running time of the corresponding dynamic programming algorithm is the same (asymptotically) as that of duplication-deletion distance. It is $O(|\mathbf{y}|^2|\mathbf{x}|\mu(\mathbf{y})\mu(\mathbf{x}))$, where the multiplicity $\mu(S)$ is the number of times a character appears in the string S, regardless of its sign.

In comparing the models of the previous section and the current one, we note that restricting the model of rearrangement to allow only duplicate and duplicate-invert operations (Section 5) instead of duplicate-invert-delete operations may be desirable from a biological perspective because each duplicate and duplicate-invert requires only three breakpoints in the genome, whereas a duplicate-invert-delete operation can be significantly more complicated, requiring more breakpoints.

7 Variants of Duplication-Inversion-Deletion Distance

It is possible to extend the model even further. We give here one detailed example which demonstrates how such extensions might be achieved. Other extensions are also possible. In the previous section we handled the model where the duplicated substring of $\mathbf{x}$ may be inverted in its entirety before being inserted into the target string. In the generalized model a substring of the duplicated string may be inverted before the string is inserted into $\mathbf{y}$. For example, we allow $(+a+b+c+d+e+f)$ to become $(+a+b-e-d-c+f)$ before being inserted into $\mathbf{y}$. In this model, the cost of duplicating a string of length m with an inversion of a substring of length ℓ is $\Delta_1 + m\Delta_2 + \Theta(\ell)$, for some non-negative monotonically increasing cost function Θ. The way we extend the recurrence is by considering all possible substring inversions to the original string X. For $1 \leq s \leq t \leq |\mathbf{x}|$, let $\tilde{\mathbf{x}}^{s,t}$ be the string $x_1 \dots x_{s-1} - x_t \cdots - x_s x_{t+1} \dots x_{|\mathbf{x}|}$. That is, the string that is obtained from $\mathbf{x}$ by inverting (in-place) $\mathbf{x}_{s,t}$. For convenience, define also $\tilde{\mathbf{x}}^{0,0} = \mathbf{x}$. We will use $\tilde{d}_i^{st}(\mathbf{x}, \mathbf{y})$ to denote the distance from $\mathbf{x}$ to $\mathbf{y}$ in this model under the additional restriction that y_1 is generated by x_i and that the substring $\mathbf{x}_{s,t}$ was inverted. Note that this does not make much sense unless $s \leq i \leq t$, since otherwise the inverted substring is not used in the duplication. However, restricting the inversion cost $\Theta(\ell)$ to be non-negative and monotonically increasing makes sure that those cases will not contribute to the minimization since inverting a character that is not duplicated will only increase the cost. The recurrence for duplication-deletion with arbitrary-substring-duplicate-inversions distance is given below.

$$
\begin{aligned}
\tilde{d}(\mathbf{x}, \emptyset) &= 0 \quad , \quad \tilde{d}(\mathbf{x}, \mathbf{y}) = \min_{\{s,t : s=0, t=0 \text{ or } 1 \leq s \leq t \leq |\mathbf{x}|\}} \min_{\{i : \tilde{\mathbf{x}}_i^{s,t} = y_1\}} \tilde{d}_i^{st}(\mathbf{x}, \mathbf{y}), \\
\tilde{d}_i(\mathbf{x}, \emptyset) &= 0 \quad , \\
\tilde{d}_i^{st}(\mathbf{x}, \mathbf{y}) &= \min \left\{ \begin{array}{l} \Delta_1 + \Delta_2 + \Theta(t-s+1) + \tilde{d}(\mathbf{x}, \mathbf{y}_{2,|\mathbf{y}|}), \\ \min_{k>i} \min_{\{j : y_j = \tilde{\mathbf{x}}_k^{s,t}, j>1\}} \left\{ \begin{array}{l} \tilde{d}(\mathbf{x}, \mathbf{y}_{2,j-1}) + \tilde{d}_k^{st}(\mathbf{x}, \mathbf{y}_{j,|\mathbf{y}|}) \\ +\Delta_2 + \Phi(k-i-1) \end{array} \right. \end{array} \right\}.
\end{aligned}
$$

The running time is $O(|\mathbf{y}|^2 |\mathbf{x}|^3 \mu(\mathbf{x}) \mu(\mathbf{y}))$. The multiplicative $|\mathbf{x}|^2$ factor in the running time in comparison with that of the previous section arises from considering all possible inverted substrings of $\mathbf{x}$. We note that if we were only interested in handling inversions to just a prefix or a suffix of the duplicated string, then it is possible to extend the duplication-inversion-deletion recurrence without increasing the asymptotic running time.

8 Duplication Distance as a Context-Free Grammar

The generation process of a string $\mathbf{y}$ by repeatedly copying substings of a source string $\mathbf{x}$ and pasting them into an initially empty target string can be very naturally described by a context-free grammar (CFG).

This alternative view might be useful in understanding our algorithms and their correctness. The goal of this section is not to establish formally the equivalence between the two presentations, but rather to provide additional intuition. To describe the idea we consider the most simple variant of duplication distance (no inversions or deletions, the cost of each operation is 1). For a fixed source string $\mathbf{x}$, we construct a grammar $G_{\mathbf{x}}$ in which for every i, j such that $1 \leq i \leq j \leq |\mathbf{x}|$, there is a production rule $S \rightarrow Sx_iSx_{i+1}S \ldots Sx_jS$. These production rules correspond to duplicating the substring $\mathbf{x}_{i,j}$. In addition there is a trivial production rule $S \rightarrow \epsilon$, where ϵ denotes the empty string. It is easy to see that the language described by this grammar is exactly the set of strings that can be duplicated from $\mathbf{x}$. The non-overlapping property (Lemma 1) is now an immediate consequence of the structure of parse trees of CFGs. Finding the duplication distance from $\mathbf{x}$ to $\mathbf{y}$ is equivalent to finding a parse tree with a minimal number of non-trivial productions among all possible parse trees for $\mathbf{y}$.

Consider now the slightly different grammar obtained by removing the leading S to the left of x_i from each of the production rules, so that the new rules are of the form $S \rightarrow x_iSx_{i+1}S \ldots Sx_jS$. It is not difficult to see that both grammars produce the same language and have the same minimal size parse tree for every string $\mathbf{y}$. The change only restricts the order in which rules are applied. For example, y_1 is always produced by the first production rule.

The recurrence for $d_i(\mathbf{x}, \mathbf{y})$ naturally arises by observing that if T is an optimal parse tree for $\mathbf{y}$ in which the first production rule generates y_1 by x_i and y_j by x_{i+1}, then the subtree T_1 of T that generates $\mathbf{y}_{2,j-1}$ is a valid parse tree which is optimal for $\mathbf{y}_{2,j-1}$. Similarly, the tree T_2 obtained by deleting x_i and T_1 from T is a valid parse tree which is optimal for $\mathbf{y}_{j,|\mathbf{y}|}$ under the restriction that y_j must be generated by x_{i+1} (see Fig. 7). Moreover, T_1 and T_2 are disjoint trees which contain all non trivial productions in T. This explains the term $d(\mathbf{x}, \mathbf{y}_{2,j-1}) + d_{i+1}(\mathbf{x}, \mathbf{y}_{j,|\mathbf{y}|})\}$ in Eq. 2, which is the heart of the recursion. The minimization over $\{j : y_j = x_{i+1}, j > 1\}$ simply enumerates all of the possibilities for constructing T. The term $1 + d(\mathbf{x}, \mathbf{y}_{2,|\mathbf{y}|})$ handles the possibility that y_1 is generated by a duplicate operation that ends with x_i. In this case the tree T_2 is empty, so we only consider T_1. We add one to account for the production rule at the root of T which is not part of T_1. This is illustrated in Fig. 8

9 Conclusion

We have shown how to generalize duplication distance to include certain types of deletions and inversions and how to compute these new distances efficiently via dynamic programming. In earlier work [15, 16], we used duplication distance to derive phylogenetic relationships between human segmental duplications. We

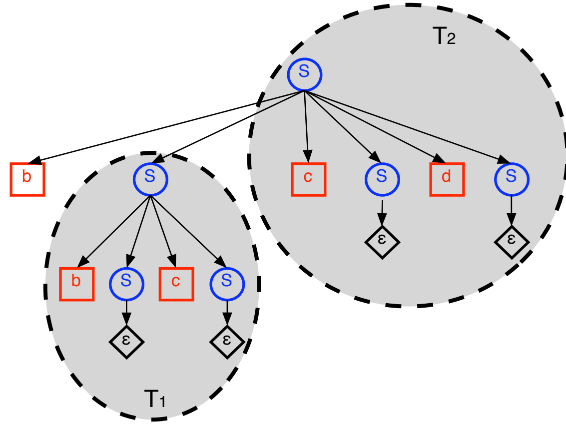

Figure 7: An optimal parse tree T for $\mathbf{y}$ =bbccd where $\mathbf{x}$ =abcd. The root production duplicates $\mathbf{x}_{2,4}$ =bcd. x_2 generates y_1 and x_3 generates y_4. The trees T_1 and T_2 are indicated. T_1 is an optimal parse tree for $\mathbf{y}_{2,4-1}$ =bc. T_2 is an optimal parse tree for $\mathbf{y}_{4,|\mathbf{y}|}$ =cd.

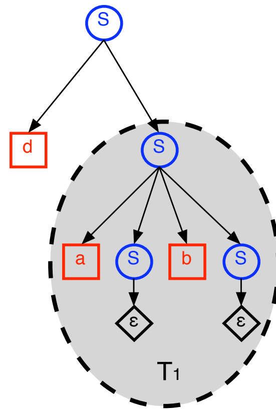

Figure 8: An optimal parse tree T for $\mathbf{y}$ =dab where $\mathbf{x}$ =abcd. The root production duplicates just x_4 =d. The tree T_1 is indicated. T_2 is empty (not indicated). The root production is not part of T_1.

plan to apply the generalized distances introduced here to the same data to determine if these richer computational models yield new biological insights.

Competing interests

The authors declare that they have no competing interests.

Authors contributions

All authors contributed equally. All authors read and approved the final manuscript.

Acknowledgements

SM was supported by NSF Grant CCF-0635089. BJR is supported by a Career Award at the Scientific Interface from the Burroughs Wellcome Fund and by funding from the ADVANCE Program at Brown University, under NSF Grant No. 0548311.

References

1. Sankoff D, Leduc G, Antoine N, Paquin B, Lang B, Cedergren R: **Gene Order Comparisons for Phylogenetic Inference: Evolution of the Mitochondrial Genome**. *Proc. Natl. Acad. Sci. U.S.A.* 1992, **89**(14):6575–6579.
2. Pevzner P: *Computational molecular biology : an algorithmic approach*. Cambridge, Mass.: MIT Press 2000.
3. Chen X, Zheng J, Fu Z, Nan P, Zhong Y, Lonardi S, Jiang T: **Assignment of Orthologous Genes via Genome Rearrangement**. *IEEE/ACM Trans. Comp. Biol. Bioinformatics* 2005, **2**(4):302–315.
4. Marron M, Swenson KM, Moret BME: **Genomic Distances Under Deletions and Insertions**. *TCS* 2004, **325**(3):347–360.
5. El-Mabrouk N: **Genome Rearrangement by Reversals and Insertions/Deletions of Contiguous Segments**. In *In Proc. 11th Ann. Symp. Combin. Pattern Matching (CPM)*, *Volume 1848*, Berlin: Springer-Verlag 2000:222–234.
6. Zhang Y, Song G, Vinar T, Green ED, Siepel AC, Miller W: **Reconstructing the Evolutionary History of Complex Human Gene Clusters**. In *Proc. 12th Int'l Conf. on Research in Computational Molecular Biology (RECOMB)* 2008:29–49.
7. Ma J, Ratan A, Raney BJ, Suh BB, Zhang L, Miller W, Haussler D: **DUPCAR: Reconstructing Contiguous Ancestral Regions with Duplications**. *Journal of Computational Biology* 2008, **15**(8):1007–1027, [http://www.liebertonline.com/doi/abs/10.1089/cmb.2008.0069].
8. Bertrand D, Lajoie M, El-Mabrouk N: **Inferring Ancestral Gene Orders for a Family of Tandemly Arrayed Genes**. *J Comp Biol* 2008, **15**(8):1063–1077, [http://www.liebertonline.com/doi/abs/10.1089/cmb.2008.0025].
9. Chaudhuri K, Chen K, Mihaescu R, Rao S: **On the Tandem Duplication-Random Loss Model of Genome Rearrangement**. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, New York, NY, USA: ACM 2006:564–570.
10. Elemento O, Gascuel O, Lefranc MP: **Reconstructing the Duplication History of Tandemly Repeated Genes**. *Mol Biol Evol* 2002, **19**(3):278–288, [http://mbe.oxfordjournals.org/cgi/content/abstract/19/3/278].

11. Lajoie M, Bertrand D, El-Mabrouk N, Gascuel O: **Duplication and Inversion History of a Tandemly Repeated Genes Family**. *J. Comp. Bio.* 2007, **14**(4):462–478.

12. El-Mabrouk N, Sankoff D: **The Reconstruction of Doubled Genomes**. *SIAM J. Comput.* 2003, **32**(3):754–792.

13. Alekseyev MA, Pevzner PA: **Whole Genome Duplications and Contracted Breakpoint Graphs**. *SICOMP* 2007, **36**(6):1748–1763.

14. Bailey J, Eichler E: **Primate Segmental Duplications: Crucibles of Evolution, Diversity and Disease**. *Nat. Rev. Genet.* 2006, **7**:552–564.

15. Kahn CL, Raphael BJ: **Analysis of Segmental Duplications via Duplication Distance**. *Bioinformatics* 2008, **24**:i133–138.

16. Kahn CL, Raphael BJ: **A Parsimony Approach to Analysis of Human Segmental Duplications**. In *Pacific Symposium on Biocomputing* 2009:126–137.

Figures

Figure 1 - Overlapping

The red subsequence is overlapping with the blue subsequence in $\mathbf{x}$. The indices $(s_i, s_{i'})$ and $(t_j, t_{j'})$ are alternating in $\mathbf{x}$.

Figure 2 - Inside

The red subsequence is inside the blue subsequence T. All the characters of the red subsequence occur between the indices t_i and t_{i+1} of T.

Figure 3 - A duplicate operation

A duplicate operation, denoted $\delta_{\mathbf{x}}(s,t,p)$. A substring $x_s x_{s+1} \ldots x_t$ of the source string $\mathbf{x}$ is copied and inserted into the target string $\mathbf{z}$ at index p.

Figure 4 - Recurrence: Case 1

y_1 is generated from x_i in a duplicate operation where y_1 is the last (rightmost) character in the copied substring (Case 1). The total duplication distance is one plus the duplication distance for the suffix $\mathbf{y}_{2,|\mathbf{y}|}$.

Figure 5 - Recurrence: Case 2

y_1 is generated from x_i in a duplicate operation where y_1 is not the last (rightmost) character in a copied substring (Case 2). In this case, x_{i+1} is also copied in the same duplicate operation (top). Thus, the duplication distance is the sum of $d(\mathbf{x}, \mathbf{y}_{2,j-1})$, the duplication distance for $\mathbf{y}_{2,j-1}$ (bottom left), and $d_{i+1}(\mathbf{x}, \mathbf{y}_{j,|\mathbf{y}|})$, the minimum number of duplicate operations to generate $\mathbf{y}_{j,|\mathbf{y}|}$ given that x_{i+1} generates y_j (bottom right).

Figure 6 - A delete operation

A delete operation, denoted $\tau(s,t)$. The substring $\mathbf{z}_{s,t}$ is deleted.

Figure 7 - Example parse tree

An optimal parse tree T for $\mathbf{y}$ =bbccd where $\mathbf{x}$ =abcd. The root production duplicates $\mathbf{x}_{2,4}$ =bcd. x_2 generates y_1 and x_3 generates y_4. The trees T_1 and T_2 are indicated. T_1 is an optimal parse tree for $\mathbf{y}_{2,4-1}$ =bc. T_2 is an optimal parse tree for $\mathbf{y}_{4,|\mathbf{y}|}$ =cd.

Figure 8 - Example parse tree

An optimal parse tree T for $\mathbf{y}$ =dab where $\mathbf{x}$ =abcd. The root production duplicates just x_4 =d. The tree T_1 is indicated. T_2 is empty (not indicated). The root production is not part of T_1.