

# Efficient Algorithms for Analyzing Segmental Duplications, Deletions, and Inversions in Genomes

Crystal L. Kahn<sup>\*1</sup>, Shay Mozes<sup>\*\*1</sup>, and Benjamin J. Raphael<sup>\*1,2</sup>

<sup>1</sup> Department of Computer Science, Brown University, and

<sup>2</sup> Center for Computational Molecular Biology, Brown University,  
Providence, RI 02912, USA.

{clkahn,shay,braphael}@cs.brown.edu

**Abstract.** Segmental duplications, or low-copy repeats, are common in mammalian genomes. In the human genome, most segmental duplications are mosaics consisting of pieces of multiple other segmental duplications. This complex genomic organization complicates analysis of the evolutionary history of these sequences. Earlier, we introduced a genomic distance, called duplication distance, that computes the most parsimonious way to build a target string by repeatedly copying substrings of a source string. We also showed how to use this distance to describe the formation of segmental duplications according to a two-step model that has been proposed to explain human segmental duplications. Here we describe polynomial-time exact algorithms for several extensions of duplication distance including models that allow certain types of substring deletions and inversions. These extensions 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 the mutational process 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.

---

\* Work supported by funding from the ADVANCE Program at Brown University, under NSF Grant No. 0548311, and by a Career Award at the Scientific Interface from the Burroughs Wellcome Fund to BJR.

\*\* Work supported by NSF Grant CCF-0635089.

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 [8–11] and whole-genome duplication [12, 13] histories.

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 maximal 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  $O(|\mathbf{y}|^2|\mathbf{x}|\mu(\mathbf{x})\mu(\mathbf{y}))$ -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. Deletion operations are also permitted.
3. 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.

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

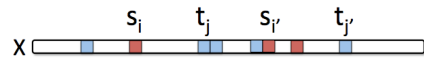
## 2 Preliminaries

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  $i_1 < i_2 < \dots < i_k$ . 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  $S_2$ .

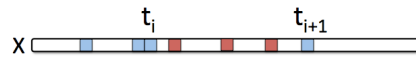
**Definition 1.** Subsequences  $S = (s_1, s_2)$  and  $T = (t_1, t_2)$  of a string  $\mathbf{x}$  are **alternating** in  $\mathbf{x}$  if either  $s_1 < t_1 < s_2 < t_2$  or  $t_1 < s_1 < t_2 < s_2$ .

**Definition 2.** Subsequences  $S = (s_1, \dots, s_k)$  and  $T = (t_1, \dots, t_l)$  of a string  $\mathbf{x}$  are **overlapping** in  $\mathbf{x}$  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  $\mathbf{x}$ . See Figure 1.

**Definition 3.** Given subsequences  $S = (s_1, \dots, s_k)$  and  $T = (t_1, \dots, t_l)$  of a string  $\mathbf{x}$ ,  $S$  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.



**Fig. 1.** Two blue (light) and red (dark) subsequences are overlapping in  $\mathbf{x}$ . The indices  $(s_i, s_{i'})$  and  $(t_j, t_{j'})$  are alternating in  $\mathbf{x}$ .

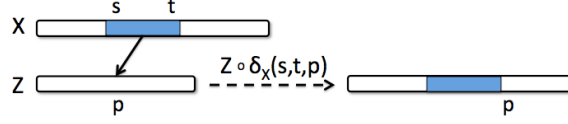


**Fig. 2.** The red (dark) subsequence is inside the blue (light) subsequence  $T$ . All the characters of the red subsequence occur between the indices  $t_i$  and  $t_{i+1}$  of  $T$ .

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

The cost associated with a duplicate operation is 1.<sup>3</sup>

<sup>3</sup> For simplicity of exposition we fix the cost of each duplicate operation to be one (i.e. the duplication distance just counts the number of duplicate operations). It is not difficult to extend the cost to an affine function of the length of the duplicated substring. The technique is similar to that of handling the affine cost of inversion in section 5. We omit the details.



**Fig. 3.** A duplicate operation,  $\delta_x(s, t, p)$ . A substring of the source string  $\mathbf{x}$  ranging between indices  $s$  and  $t$  is copied and inserted into the target string  $\mathbf{z}$  at index  $p$ .

**Definition 5.** The **duplication distance** from a source string  $\mathbf{x}$  to a target string  $\mathbf{y}$  is the minimum cost of a sequence of duplicate operations from  $\mathbf{x}$ ,  $\delta_x(s_1, t_1, p_1), \delta_x(s_2, t_2, p_2), \delta_x(s_3, t_3, p_3), \dots$ , that generates  $\mathbf{y}$  from an initially empty target string<sup>4</sup> (that is,  $\mathbf{y} = \emptyset \circ \delta_x(s_1, t_1, p_1) \circ \delta_x(s_2, t_2, p_2) \circ \dots$ ).

**Lemma 1 (Non-overlapping Property).** Consider a source string  $\mathbf{x}$ , an initial target string  $\mathbf{z}$ , and a sequence of duplicate operations of the form  $\delta_x(s_i, t_i, p_i)$  that transforms  $\mathbf{z}$  into the final target string  $\mathbf{y}$ . The substrings of  $\mathbf{x}$  that are duplicated during the construction of  $\mathbf{y}$  appear as mutually non-overlapping subsequences of  $\mathbf{y}$ .

*Proof.* (Sketch) First note that a substring  $S$  that is duplicated from  $\mathbf{x}$  and inserted into a target string  $\mathbf{z}$  is initially a substring of the newly augmented target, but subsequent duplicate operations may insert other strings in between the characters of  $S$  transforming it into a subsequence of the final target string  $\mathbf{y}$ . Consider now two strings  $S_1$  and  $S_2$  that are copied from  $\mathbf{x}$  and inserted into an intermediate target string in succession yielding an augmented target string  $\mathbf{z}'$ . There are two cases: (1)  $S_1 \cap S_2 = \emptyset$  or (2)  $S_2$  is inside of  $S_1$  in  $\mathbf{z}'$  (Definition 3). In either case,  $S_1$  and  $S_2$  are not overlapping in  $\mathbf{z}'$ . This argument can be extended for any sequence of duplicate operations; as each string is inserted into an intermediate target string  $\mathbf{z}'$ , it does not overlap any of the subsequences of  $\mathbf{z}'$  that correspond to previously-inserted strings; we maintain the invariant that the subsequences of  $\mathbf{z}'$  corresponding to characters inserted in individual duplicate operations are mutually non-overlapping in  $\mathbf{z}'$ , and therefore also in the final string  $\mathbf{y}$ .  $\square$

### 3 Duplication Distance

In this section we review the basic recurrence for computing duplication distance that was introduced in [16]. 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. 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  $\mathbf{x}$  that can generate  $y_1$ , see Eq. 1. To compute

<sup>4</sup> We assume that every character in  $\mathbf{y}$  appears at least once in  $\mathbf{x}$ .

$d_i(\mathbf{x}, \mathbf{y})$ , there are two possible cases to consider: either (1)  $y_1$  was duplicated as a substring of  $\mathbf{x}$  of length one, namely just  $x_i$ , or (2)  $x_{i+1}$  is also copied in the same duplicate operation. In the former case the minimum cost is 1 (for this duplicate operation) plus the minimum cost to generate the remaining suffix of  $\mathbf{y}$ , namely  $1 + d(\mathbf{x}, \mathbf{y}_{2,|\mathbf{y}|})$ . In the latter case, the set of candidate characters in  $\mathbf{y}$  to be generated by  $x_{i+1}$  is  $\{j : y_j = x_{i+1}, j > 1\}$ . For each such  $j$ , the minimum cost of generating  $\mathbf{y}$  provided that  $y_1$  is generated by  $x_i$  and  $y_j$  is generated by  $x_{i+1}$ , is the minimum cost of generating  $\mathbf{y}_{2,j-1}$  (i.e.,  $d(\mathbf{x}, \mathbf{y}_{2,j-1})$ ) plus the minimum cost of generating the string  $y_1 \mathbf{y}_{j,|\mathbf{y}|}$  using  $x_i$  and  $x_{i+1}$  to generate  $y_1$  and  $y_j$ . Since  $x_i$  and  $x_{i+1}$  are copied in the same duplicate operation, the cost of generating  $y_1 \mathbf{y}_{j,|\mathbf{y}|}$  using  $x_i$  and  $x_{i+1}$  is equal to the cost of generating just  $\mathbf{y}_{j,|\mathbf{y}|}$  using  $x_{i+1}$ , namely  $d_{i+1}(\mathbf{x}, \mathbf{y}_{j,|\mathbf{y}|})$ , see Eq. 2. The recurrence is, therefore:

$$\begin{aligned} d(\mathbf{x}, \emptyset) &= 0 \\ d(\mathbf{x}, \mathbf{y}) &= \min_{\{i: x_i = y_1\}} d_i(\mathbf{x}, \mathbf{y}) \end{aligned} \quad (1)$$

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

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

*Proof.* Let  $OPT(\mathbf{x}, \mathbf{y})$  denote minimum cost of a sequence of duplicate operations that generate  $\mathbf{y}$  from  $\mathbf{x}$ . Let  $OPT_i(\mathbf{x}, \mathbf{y})$  denote the minimum cost 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  $y_1$ :

- Consider a minimum-cost sequence of duplicates that generates  $\mathbf{y}_{2,|\mathbf{y}|}$ . By the inductive hypothesis its cost 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 cost is  $1 + d(\mathbf{x}, \mathbf{y}_{2,|\mathbf{y}|})$ .
- For every  $\{j : y_j = x_{i+1}, j > 1\}$  consider a minimum-cost sequence of duplicates that generates  $\mathbf{y}_{j,|\mathbf{y}|}$  using  $x_{i+1}$  to produce  $y_j$ , and a minimum-cost sequence of duplicates that generates  $\mathbf{y}_{2,j-1}$ . By the inductive hypothesis their costs 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, and hence the same cost.

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

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

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-cost 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  $\Delta$  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  $\Delta$  is ordered after all the other duplicates so that first  $y_1 y_j \dots y_{|\mathbf{y}|}$  is generated, and then  $\Delta$  generates  $y_2 \dots y_{j-1}$  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  $d_i(\mathbf{x}, \mathbf{y})$ .  $\square$

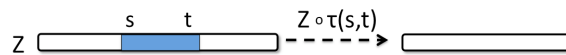
This recurrence naturally translates into a dynamic programming algorithm that computes the values of  $d(\mathbf{x}, \cdot)$  and  $d_i(\mathbf{x}, \cdot)$  for various target strings. To analyze the running time of this algorithm, note that both  $\mathbf{y}_{2,j}$  and  $\mathbf{y}_{j,|\mathbf{y}|}$  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.

## 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  $z_i, \dots, z_j$  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  $z_s \dots z_t$  of the target string  $\mathbf{z}$ , thus making  $\mathbf{z}$  shorter. Specifically, if  $\mathbf{z} = z_1 \dots z_s \dots z_t \dots z_m$ , then  $\mathbf{z} \circ \tau(s, t) = z_1 \dots z_{s-1} z_{t+1} \dots z_m$ . See Figure 4.

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



**Fig. 4.** A delete operation,  $\tau(s, t)$ . The substring of  $\mathbf{z}$  that ranges between indices  $s$  and  $t$  is deleted.

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

We now show that although we allow arbitrary deletions from the intermediate string, it suffices to consider just 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  $\mathbf{x}$ ,  $\eta_{\mathbf{x}}(i_1, j_1, i_2, j_2, \dots, i_k, j_k, p)$ , for  $i_1 \leq j_1 < i_2 \leq j_2 < \dots < i_k \leq j_k$  copies the subsequence  $x_{i_1} \dots x_{j_1} x_{i_2} \dots x_{j_2} \dots x_{i_k} \dots x_{j_k}$  of the source string  $\mathbf{x}$  and pastes it into a target string at position  $p$ . Specifically, if  $\mathbf{x} = x_1 \dots x_m$  and  $\mathbf{z} = z_1 \dots z_n$ , then  $\mathbf{z} \circ \eta_{\mathbf{x}}(i_1, j_1, \dots, i_k, j_k, p) = z_1 \dots z_{p-1} x_{i_1} \dots x_{j_1} x_{i_2} \dots x_{j_2} \dots x_{i_k} \dots x_{j_k} z_p \dots z_n$ .

The cost associated with such a duplication-deletion is  $1 + \sum_{\ell=1}^{k-1} \Phi(i_{\ell+1} - j_{\ell} - 1)$ .

**Lemma 2.** If  $\Phi(\cdot)$  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  $\mathbf{y}$  from a source string  $\mathbf{x}$  is equal to the cost of a minimum sequence of duplicate-delete operations that generates  $\mathbf{y}$  from  $\mathbf{x}$ .

*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,  $\tau$ . Let  $k$  denote the number of duplicate-delete operations that precede  $\tau$ , and let  $\mathbf{z}$  be the intermediate string produced by these  $k$  operations. For  $i = 1, \dots, k$ , let  $S_i$  be the subsequence of  $\mathbf{x}$  that was used in the  $i$ th duplicate-delete operation.  $S_1, \dots, S_k$  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  $i$ th duplicate-delete operation are deleted, so we may omit the  $i$ th operation altogether and decrease the number of characters deleted by  $\tau$ . 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  $\tau$ ). We change the  $i$ th duplicate-delete operation to start at  $r$  and decrease the number of characters deleted by  $\tau$ . Since  $\Phi(\cdot)$  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  $\tau$  only deletes characters in  $S_i$ . We can then change the  $i$ th duplicate-delete operation to also delete the characters deleted by  $\tau$ , and omit  $\tau$ . 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  $\{S_i\}$  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.  $\square$

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:

$$\hat{d}(\mathbf{x}, \emptyset) = 0 \quad , \quad \hat{d}(\mathbf{x}, \mathbf{y}) = \min_{\{i: x_i = y_1\}} \hat{d}_i(\mathbf{x}, \mathbf{y}), \quad (3)$$

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

$$\hat{d}_i(\mathbf{x}, \mathbf{y}) = \min \left\{ \begin{array}{l} 1 + \hat{d}(\mathbf{x}, \mathbf{y}_{2,|\mathbf{y}|}), \\ \min_{k>i} \min_{\{j: y_j = x_k, j>1\}} \left\{ \begin{array}{l} \hat{d}(\mathbf{x}, \mathbf{y}_{2,j-1}) + \hat{d}_k(\mathbf{x}, \mathbf{y}_{j,|\mathbf{y}|}) \\ + \Phi(k - i - 1) \end{array} \right\} \end{array} \right\}. \quad (4)$$

**Theorem 2.**  $\hat{d}(\mathbf{x}, \mathbf{y})$  is the duplication-deletion distance from  $\mathbf{x}$  to  $\mathbf{y}$ . For  $\{i : x_i = y_1\}$ ,  $\hat{d}_i(\mathbf{x}, \mathbf{y})$  is the duplication-deletion distance from  $\mathbf{x}$  to  $\mathbf{y}$  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. Note that if the cost function is the identity  $\Phi(\cdot) = 1$ , then the duplication-deletion distance is equal to duplication distance without deletions. We omit the proof.

## 5 Duplication-Inversion-Deletion 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 (-). An inversion of the signed string  $(+1 + 2 + 3)$  yields  $(-3 - 2 - 1)$ . 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  $\ell$  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 and delete operations from the target string at any time during the generation process. 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  $(+1 + 2 + 3)$  and then deleting  $-2$  may be different than the cost of first deleting  $+2$  from  $(+1 + 2 + 3)$  and then inverting  $(+1 + 3)$ .

**Definition 9.** A *duplicate-invert operation* from  $\mathbf{x}$ ,  $\bar{\delta}_{\mathbf{x}}(s, t, p)$ , copies an inverted substring  $-x_t, \dots, -x_s$  of the source string  $\mathbf{x}$  and pastes it into a target string at position  $p$ . Specifically, if  $\mathbf{x} = x_1 \dots x_m$  and  $\mathbf{z} = z_1 \dots z_n$ , then  $\mathbf{z} \circ \bar{\delta}_{\mathbf{x}}(s, t, p) = z_1 \dots z_{p-1} \bar{x}_t \bar{x}_{t-1} \dots \bar{x}_s z_p \dots z_n$ .

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

**Definition 10.** The *duplication-inversion-deletion distance* from a source string  $\mathbf{x}$  to a target string  $\mathbf{y}$  is the cost of a minimum sequence of duplicate and duplicate-invert operations from  $\mathbf{x}$  and deletion operations, in any order, that generates  $\mathbf{y}$ .

**Definition 11.** A *duplicate-invert-delete operation* from  $\mathbf{x}$ ,  $\bar{\eta}_{\mathbf{x}}(i_1, j_1, i_2, j_2, \dots, i_k, j_k, p)$ , for  $i_1 \leq j_1 < i_2 \leq j_2 < \dots < i_k \leq j_k$  pastes the string  $-x_{j_k} - x_{j_k-1} \dots - x_{i_k} - x_{j_k-1} - x_{j_k-1-1} \dots - x_{i_{k-1}} \dots \dots - x_{j_1} - x_{j_1-1} \dots - x_{i_1}$  into a target string at position  $p$ . Specifically, if  $\mathbf{x} = x_1 \dots x_m$  and  $\mathbf{z} = z_1 \dots z_n$ , then  $\mathbf{z} \circ \bar{\eta}_{\mathbf{x}}(i_1, j_1, i_2, j_2, \dots, i_k, j_k, p) = z_1 \dots z_{p-1} - x_{j_k} - x_{j_k-1} \dots - x_{i_k} - x_{j_k-1} - x_{j_k-1-1} \dots - x_{i_{k-1}} \dots \dots - x_{j_1} - x_{j_1-1} \dots - x_{i_1} z_p \dots z_n$ .

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

**Lemma 3.** *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  $\mathbf{y}$  from a source string  $\mathbf{x}$  is equal to the cost of a minimum sequence of duplicate and duplicate-invert-delete operations that generates  $\mathbf{y}$  from  $\mathbf{x}$ .*

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  $\mathbf{x}$ .

There are a few differences between the recurrence for duplication-inversion-deletion distance and the one for duplication-deletion distance (Eqs. 3, 4). First, 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-deletion 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  $\hat{d}_i$  in Eq. 4. 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, when considering  $k$ , the next character in  $\mathbf{x}$  to be used in this operation, we look for an index smaller than  $i$  rather than larger than  $i$ . The recurrence for  $\bar{d}_i^-$  also differs in the cost term. If  $x_k$  is the next character used by this operation, then all the characters between  $x_k$  and  $x_i$  are part of the inversion. We therefore add  $(i - k)\Theta_2$  to the cost. If  $x_i$  is the last character used by this operation, we add  $\Theta_2 + \Theta_1$  to the cost. The extension of the recurrence from the previous section to duplication-inversion-deletion distance is therefore:

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

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

We omit the proof. The running time of the corresponding dynamic programming algorithm remains  $O(|\mathbf{y}|^2|\mathbf{x}|\mu(\mathbf{y})\mu(\mathbf{x}))$ , where the multiplicity  $\mu(S)$  is maximal number of times a character appears in the string  $S$ , regardless of its sign.

We note here without further explanation that if a sequence of duplicate and duplicate-invert operations is desired (without any deletions), it suffices to modify the recurrence that computes duplication distance in section 3, increasing its running time by only a constant factor. Restricting the model of rearrangement to allow only duplicate and duplicate-invert operations (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.

**Other Variants of Duplication with Inversions** Above we handled the model where the duplicated substring of  $\mathbf{x}$  may be inverted in its entirety before being inserted into the target. We now generalize to models in which only a substring of the duplicated string is inverted before being inserted into  $\mathbf{y}$ . If the length of the inverted substring is  $\ell$ , then the cost of this operation is  $1 + \Theta(\ell)$ . For example, we allow the substring  $(+1 + 2 + 3 + 4 + 5 + 6)$  to become  $(+1 + 2 - 5 - 4 - 3 + 6)$  before being inserted into  $\mathbf{y}$ .

It is possible to extend the duplication-inversion-deletion recurrence to handle an inversion of any prefix or suffix of the duplicated substring, without changing the asymptotic running time of the recurrence. We omit the details.

Another variant allows for inversion of an arbitrary substring of the duplicated string at the price of asymptotically longer running time. For  $1 \leq s \leq t \leq |\mathbf{x}|$ , let  $\tilde{\mathbf{x}}^{s,t}$  be the string  $x_1 \dots x_{s-1} - x_t \dots - x_s x_{t+1} \dots x_{|\mathbf{x}|}$ . That is, the string that is obtained from  $\mathbf{x}$  by inverting (in-place)  $\mathbf{x}_{s,t}$ . For convenience, define also  $\tilde{\mathbf{x}}^{0,0} = \mathbf{x}$ . The duplication-deletion with arbitrary-substring-duplicate-inversions distance satisfies the following recurrence. The running time is  $O(|\mathbf{y}|^2|\mathbf{x}|^3\mu(\mathbf{x})\mu(\mathbf{y}))$ .

$$\begin{aligned} \tilde{d}(\mathbf{x}, \emptyset) &= 0 \quad , \quad \tilde{d}(\mathbf{x}, \mathbf{y}) = \min_{\{s,t:s=0,t=0 \text{ or } 1 \leq s \leq t \leq |\mathbf{x}|\}} \min_{\{i:\tilde{\mathbf{x}}_i^{s,t}=\mathbf{y}_1\}} \tilde{d}_i^{st}(\mathbf{x}, \mathbf{y}), \\ \tilde{d}_i(\mathbf{x}, \emptyset) &= 0 \quad , \\ \tilde{d}_i^{st}(\mathbf{x}, \mathbf{y}) &= \min \left\{ \begin{array}{l} 1 + \Theta(t - s + 1) + \tilde{d}(\mathbf{x}, \mathbf{y}_{2,|\mathbf{y}|}), \\ \min_{k>i} \min_{\{j:y_j=\tilde{\mathbf{x}}_k^{s,t}, j>1\}} \left\{ \begin{array}{l} \tilde{d}(\mathbf{x}, \mathbf{y}_{2,j-1}) + \tilde{d}_k^{st}(\mathbf{x}, \mathbf{y}_{j,|\mathbf{y}|}) \\ + \Phi(k - i - 1) \end{array} \right\} \end{array} \right\}. \end{aligned}$$

## 6 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

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

## 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.* **89**(14) (1992) 6575–6579
2. Pevzner, P.: *Computational molecular biology : an algorithmic approach*. MIT Press, Cambridge, Mass. (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* **2**(4) (2005) 302–315
4. Marron, M., Swenson, K.M., Moret, B.M.E.: Genomic Distances Under Deletions and Insertions. *TCS* **325**(3) (2004) 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 (CPM00)*. Volume 1848., Berlin, Springer-Verlag (2000) 222–234
6. Zhang, Y., Song, G., Vinar, T., Green, E.D., Siepel, A.C., 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, B.J., Suh, B.B., Zhang, L., Miller, W., Haussler, D.: Dupcar: Reconstructing contiguous ancestral regions with duplications. *Journal of Computational Biology* **15**(8) (2008) 1007–1027
8. Bertrand, D., Lajoie, M., El-Mabrouk, N.: Inferring Ancestral Gene Orders for a Family of Tandemly Arrayed Genes. *J Comp Biol* **15**(8) (2008) 1063–1077
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, M.P.: Reconstructing the Duplication History of Tandemly Repeated Genes. *Mol Biol Evol* **19**(3) (2002) 278–288
11. Lajoie, M., Bertrand, D., El-Mabrouk, N., Gascuel, O.: Duplication and Inversion History of a Tandemly Repeated Genes Family. *J. Comp. Bio.* **14**(4) (2007) 462–478
12. El-Mabrouk, N., Sankoff, D.: The reconstruction of doubled genomes. *SIAM J. Comput.* **32**(3) (2003) 754–792
13. Alekseyev, M.A., Pevzner, P.A.: Whole Genome Duplications and Contracted Breakpoint Graphs. *SICOMP* **36**(6) (2007) 1748–1763
14. Bailey, J., Eichler, E.: Primate Segmental Duplications: Crucibles of Evolution, Diversity and Disease. *Nat. Rev. Genet.* **7** (Jul 2006) 552–564
15. Kahn, C.L., Raphael, B.J.: Analysis of Segmental Duplications via Duplication Distance. *Bioinformatics* **24** (2008) i133–138
16. Kahn, C.L., Raphael, B.J.: A Parsimony Approach to Analysis of Human Segmental Duplications. In: *Pacific Symposium on Biocomputing*. (2009) 126–137