

An algorithm with linear expected running time for string editing with substitutions and substring reversals

Abdullah N. Arslan
Department of Computer Science
University of Vermont
Burlington, VT 05405, USA
aarslan@cs.uvm.edu

Abstract

The edit distance between given two strings X and Y is the minimum number of edit operations that transform X into Y without performing multiple operations in the same position. Ordinarily, string editing is based on character insert, delete, and substitute operations. Motivated from the facts that substring reversals are observed in genomic sequences, and it is not always possible to transform a given sequence X into a given sequence Y by reversals alone (e.g. X is all 0's, and Y is all 1s), Muthukrishnan and Sahinalp [7, 8] considered a "simple" well-defined edit distance model in which the edit operations are: replace a character, and reverse and replace a substring. A substring of X can only be reversed if the reversal results in a match in the same position in Y . The cost of each character replacement and substring reversal is 1. The distance in this case is defined only when $|X| = |Y| = n$. There is an algorithm for computing the distance in this model with worst-case time complexity $O(n \log^2 n)$ [8]. We present a dynamic programming algorithm with worst-case time complexity $O(n^2)$ but its expected running-time is $O(n)$. In our dynamic programming solution the weights of edit operations can vary for different substitutions, and the costs of reversals can be a function of the reversal-length.

Keywords: *edit distance, sequence alignment,*

substring (block) reversal.

1 Introduction

Given two strings X and Y computing the edit distance between two strings is an important problem which has many applications such as estimating the evolutionary distances between biological sequences [9], and constructing phylogenetic trees [4]. The edit distance between strings X and Y is the number of edit operations that transform X into Y . Basic edit operations are: insert a symbol, delete a symbol, and substitute one symbol for another. Multiple operations in the same position are not allowed. Motivated from different applications several studies have suggested extending the edit operations by including substring edits that reverse a substring, move or copy a substring from one location to another, and delete a substring (see for example [10, 4]). In its general form when edit distance is computed allowing substring copy, delete, and move operations, the problem is NP-hard [6]. Shapira and Storer [11] give approximation algorithms for several cases of edit distance that use subsets of these substring operations. Arslan [1] proposes extending the basic edit model that includes operations (insert, delete, and substitute) at character level by adding substring reversal operation, and presents an algorithm for computing the edit distance in this model whose

worst-case time complexity is $O(|X|^2|Y|)$ but that runs in $O(|X||Y|)$ expected time where $|X| \leq |Y|$. We note that computing edit distances in these extended models is different from the problem of sorting by reversals which uses reversals repeatedly on a given permutation to generate a sorted order (see for example [3]).

Substring reversals are commonly observed in genomic sequences, and in multimedia data such as music scales [10]. Substring reversals in genomic sequences are usually a consequence of a large scale inter and intra chromosomal genomic duplications. Muthukrishnan and Sahinalp [7, 8] study edit distance computation for an edit model that includes character substitutions, and substring reversals. In this case the edit distance is defined when $|X| = |Y| = n$. A reversal is allowed only if the operation on a substring creates a match in the other string. There can be no multiple operations in the same position. The cost of each edit operation is 1. This is the "simplest" well-defined edit distance that involves substring reversals if we note that not all sequences can be transformed into all other sequences by reversals alone (e.g. X is all 0's, and Y is all 1's). For example in this model string $X = accaagagcg$ can be edited to string $Y = aggaacacgc$ as follows:

- $accaagagcg$ (no operation: a's match)
- $\Rightarrow accaagagcg$ (replace c by g)
- $\Rightarrow agcaagagcg$ (reverse caag)
- $\Rightarrow aggaacagcg$ (no operation: a's match)
- $\Rightarrow aggaacagcg$ (replace g by c)
- $\Rightarrow aggaacaccg$ (reverse cg)
- $\Rightarrow aggaacacgc$

Muthukrishnan and Sahinalp introduced this edit model, and presented a $O(n \log^3 n)$ -time algorithm [7] where n is the common length of the strings X and Y . They later improved this complexity to $O(n \log^2 n)$ [8]. In this paper we consider their model. We first give a dynamic programming solution for computing the distance in this model. The worst-case time complexity of our solution is $O(n + |R|)$ where R is the set of all possible *reversible* substrings (identified by their end-positions and lengths) in X with respect to

Y . A substring of X is reversible with respect to Y if its reversal creates a match in Y in the same position. We present an algorithm that finds set R in time $O(n^2)$ in the worst-case but in $O(n)$ expected time.

The outline of this paper is as follows: in Section 2, we describe our notation, and present our dynamic programming solution for computing the edit distance in the model we consider. This solution assumes information about the end positions and lengths of all reversible substrings. In Section 3, we describe an algorithm that generates this information. We include our final remarks, and summarize our results in Section 4.

2 Edit distance with substitutions and substring reversals

Given two strings X and Y with $n = |X| \leq |Y| = m$ over an alphabet Σ where $|\Sigma| \geq 2$, the edit distance is the total number of operations needed to transform X into Y . Common model of string editing is based on insert, delete, and substitute operations at character level. There are several other models [2, 10, 4, 11]. In general, multiple operations in the same position in X are not allowed. This is necessary to obtain a metric that can be efficiently computed.

We denote by S_i the i th symbol of string S , and by $S_{i..j}$ the substring of S starting at position i , and ending at position j . We use \overleftarrow{S} to denote the *reverse* of string S , i.e. $\overleftarrow{S} = S_{|S|} \dots S_2 S_1$.

In this paper we consider the edit distance model studied by Muthukrishnan and Sahinalp [7, 8]. We say that substring $X_{(i-k+1)..i}$ of length k is *reversible* if $X_{(i-k+1)..i} = \overleftarrow{Y_{(i-k+1)..i}}$. In this model a character can be substituted for a character of X , and a substring of X can be reversed in a single step (if it is reversible). The assumption is that the string lengths are the same: $|X| = |Y| = n$, and multiple edit operations involving the same position in X are not allowed.

Let R_i be the lengths of reversible substrings ending at position i . More formally,

$$R_i = \{ k \mid X_{(i-k+1)..i} = \overleftarrow{Y_{(i-k+1)..i}}, 1 \leq i \leq n, 1 \leq k \leq i \}.$$

If R_i contains $k > 0$ then by a single reversal, $Y_{(i-k+1)..i}$ can be obtained from $X_{(i-k+1)..i}$. We consider all possible reversals ending at position i for every i .

Let D_i denote the edit distance between $X_{1..i}$ and $Y_{1..i}$ in the model we study. We can compute D_i for all i , $1 \leq i \leq n$, as follows:

$$D_i = \min\{D_{i-1} + s(X_i, Y_i), \min_{k \in R_i} D_{i-k} + f(k)\} \quad (1)$$

where $D_0 = 0$, $s(X_i, Y_i)$ is the cost of replacing X_i by Y_i , and $f(k)$ is the cost of a reversal of length k . The functions s and f are given. In the formulation of Muthukrishnan and Sahinalp [7, 8], $s(X_i, Y_i)$ is the Hamming distance between X_i and Y_i , i.e. 0 when $X_i = Y_i$, and 1 otherwise, and $f = 1$. Note that in our formulation the substitution costs can vary, and the cost of reversals can be a function of the reversal-length.

Minimum distance D_i is achieved at position i by either adding $s(X_i, Y_i)$ (the substitution cost if $X_i \neq Y_i$, or else 0) to D_{i-1} , or considering all possible reversals ending at position i of length k for all possible k and choosing an optimal one, i.e. a reversal of length k that yields the minimum total cost at position i when added to D_{i-k} .

Provided that R_i is given for all i , to compute D_n using formulation (1) takes $O(n + |R|)$ time where $R = \bigcup_{i=1}^n R_i$.

Note that if we calculate R_i 's by trying all possible k 's, this would require in total $\Omega(n^2)$ time in the worst case. We will give an algorithm that computes R_i for all i in time $O(n^2)$ in the worst case, but with $O(n)$ expected running time.

3 Computing all reversible substrings

We first give a few observations which we use in our algorithm that finds lengths of all reversible substrings ending for all positions in X .

Observation 1 For any two strings X and Y each of length n where $X = \overleftarrow{Y}$

- if n is odd then $X_{\lceil n/2 \rceil} = \overleftarrow{Y_{\lceil n/2 \rceil}}$, i.e. X and Y have the same symbol in the middle position. For example, for $X = abc$, and $Y = cba$, $X_2 = b = \overleftarrow{b} = \overleftarrow{Y_2}$,
- if n is even then $X_{\lfloor n/2 \rfloor} \dots X_{\lceil n/2 \rceil} = \overleftarrow{X_{\lfloor n/2 \rfloor} \dots X_{\lceil n/2 \rceil}}$, i.e. the substring of length two at the center of X is the reverse of the substring in the same position in Y . For example, for $X = abcd$, and $Y = dcba$, $X_{2..3} = bc = \overleftarrow{cb} = \overleftarrow{Y_{2..3}}$.

Observation 2 For any two strings X and Y each of length $n \geq 1$ where $X = \overleftarrow{Y}$,

- if n is odd, for all k , $0 \leq k \leq \lfloor n/2 \rfloor$,
 $X_{(\lfloor n/2 \rfloor - k) \dots (\lfloor n/2 \rfloor + k)} = \overleftarrow{Y_{(\lfloor n/2 \rfloor - k) \dots (\lfloor n/2 \rfloor + k)}}$,
- if n is even, for all k , $0 \leq k < n/2$,
 $X_{(n/2 - k) \dots (n/2 + k + 1)} = \overleftarrow{Y_{(n/2 - k) \dots (n/2 + k + 1)}}$.

Observation 2 simply says that in a given reversal there is a sequence of shorter reversals centered at the same position. To explain this more formally, we first show in Figure 1 a rule that extends the size of a given reversible substring of X with respect to Y by 2 if certain conditions are true: if $X_{i..j} = \overleftarrow{Y_{i..j}}$, $X_{i-1} = Y_{j+1}$, and $X_{j+1} = Y_{i-1}$ then $X_{(i-1) \dots (j+1)} = \overleftarrow{Y_{(i-1) \dots (j+1)}}$. We can interpret this rule as follows: if we first locate the initial reversible substring (of length 1 if n is odd, and of length 2 if n is even) at the center position of X and Y where $X = \overleftarrow{Y}$, and then extend this reversal by applying the rule in Figure 1 then we obtain longer reversible substrings after each extension until the entire strings are covered. Figure 2 schematically describes all such reversals.

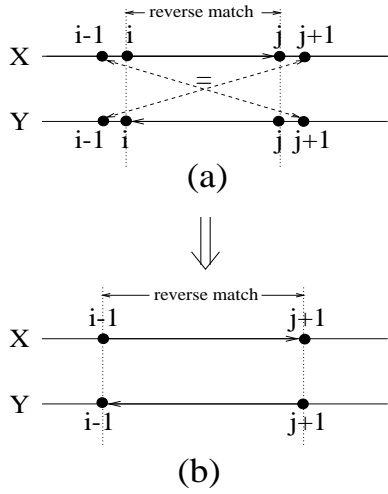


Figure 1: Rule to extend a given reversible substring. Part (a) implies part (b). We say *reverse-match* to mean a reversal that creates a match in strings X and Y .

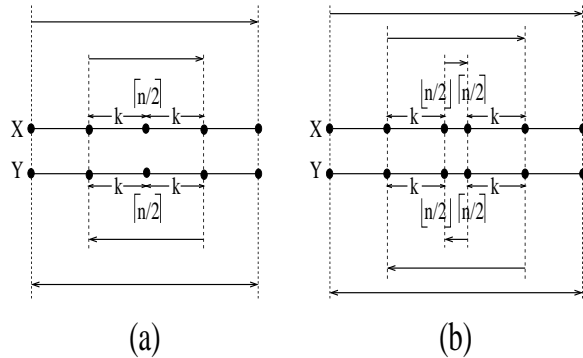


Figure 2: Schematic description of Observation 2 for X and Y where X is the reverse string of Y . The start and end positions on the directed line segments in the figure identify the regions where there are reversible substrings. (a) Reversible substrings of odd length. (b) Reversible substrings of even length.

Corollary 1 For any two given strings X and Y , the maximum number of reversals centered at a given position i is at most the maximum length of the reversible substring centered at position i .

For given two strings X , and Y of length n where $X = \overleftarrow{Y}$, we mean by *seed* an initial reversible substring at the center of X and Y . If n is odd then the seed is of length 1, and if n is even the seed is of length 2 (Observation 1). Starting with the seed and applying the extension rule in Figure 1 incrementally as long as possible we will find all reversible substrings centered at this seed. If we do this for all initial seeds that we find in any given two strings X and Y (not necessarily $X = \overleftarrow{Y}$), we can compute the lengths of all reversible substrings of X with respect to Y centered at all possible positions.

Figure 3 shows our algorithm *FindAll* which computes and returns $R = \bigcup_{i=1}^n R[i]$ where $R[i]$ is the set of all reversible substrings ending at position i . We consider reversible substrings of odd length, and even length separately, in Part 1 and Part 2 of the algorithm, respectively. In Part 3 we combine the results of Part 1 and Part 2 to generate array R .

In Part 1, we first find all reversible substrings of odd length. We index reversals by their midpoints (seed positions). For odd-length reversals the seeds are positions of single symbols. We compute all reversals centered at these seeds, and we keep track of them in array M_1 . We start with examining each position i , $1 \leq i \leq n$, to identify the seeds for reversals of odd length. If X and Y agree on position i then this means that there is a reversible substring of length 1 at i , and it can potentially be a midpoint (seed) of larger reversals. We apply the rule in Figure 1 to see if the current reversible substring can be extended. If the answer is yes then we enlarge the reversal centered at this seed. We continue this process until no more extension is possible. By Observation 2 the last extension yields the longest reversible substring centered at seed position i . At this point array $M_1[i]$ stores the lengths of all reversible substrings of odd-length whose seeds are at position

```

Algorithm FindAll
/* Part 1: find all odd-length
   reversible substrings */
for  $i := 1$  to  $n$  do
  if  $X[i] = Y[i]$  then  $M_1[i] := \{1\}$ 
  else  $M_1[i] := \emptyset$ 
for  $i := 1$  to  $n$  do
  if  $1 \in M_1[i]$  then {
     $k := 1$ 
    while (( $i - k \geq 1$  and  $i + k \leq n$ )
           and ( $X[i - k] = Y[i + k]$ 
                and  $X[i + k] = Y[i - k]$ )) do
      {  $M_1[i] := M_1[i] \cup \{2 * k + 1\}$ ;  $k := k + 1$  }
    }
/* Part 2: find all even-length
   reversible substrings */
for  $i := 1$  to  $n - 1$  do
  if  $X[i] = Y[i + 1]$  and  $X[i + 1] = Y[i]$ 
  then  $M_2[i] := \{2\}$  else  $M_2[i] := \emptyset$ 
for  $i := 1$  to  $n - 1$  do
  if  $2 \in M_2[i]$  then {
     $k := 2$ 
    while (( $i - k \geq 1$  and  $i + k \leq n$ )
           and ( $X[i - k + 1] = Y[i + k]$ 
                and  $X[i + k] = Y[i - k + 1]$ )) do
      {  $M_2[i] := M_2[i] \cup \{2 * k + 2\}$ ;  $k := k + 1$  }
    }
/* Part 3: find  $R$  from  $M_1$  and  $M_2$  */
for  $i := 1$  to  $n$  do  $R[i] := \emptyset$ 
for  $i := 1$  to  $n$  do {
  for every  $r \in M_1[i]$  do
    {  $k := (r - 1)/2$ ;  $R[i + k] := r$  }
  for  $i := 1$  to  $n - 1$  do
    for every  $r \in M_2[i]$  do
      {  $k := r/2 - 1$ ;  $R[i + k + 1] := r$  }
}
return  $R$ 

```

Figure 3: Algorithm *FindAll* which finds all reversible substrings ending at position i in X for all i , $1 \leq i \leq n$.

i . We repeat these steps for all i , $1 \leq i \leq n$.

Part 2 of the algorithm is similar to Part 1. The seeds in this case are length-two reversible substrings. We calculate positions in reversible substrings relative to the position of the first symbol of the seed. After the execution of this part, $M_2[i]$ stores the lengths of all reversible substrings of even-length whose length-two seeds start at position i for all i , $1 \leq i \leq n - 1$.

In the last part, Part 3, of the algorithm we compute R from M_1 and M_2 such that $R[i]$ is the set of all reversible substrings ending at position i in X for all i , $1 \leq i \leq n$. From the seed positions i , and lengths of the reversible substrings in $M_1[i]$ and $M_2[i]$ we identify their end positions, and collect all reversible substring lengths (odd or even) in array $R[i]$. For every $r \in M_1[i]$ there is a reversal ending at position $i + k$ of length r where $r = 2k + 1$. Similarly, for every $r \in M_2[i]$ there is a reversible substring ending at position $i + k + 2$ of length r where $r = 2k + 2$. Clearly set $R[i]$ obtained at the end of Part 3 contains the lengths of all possible reversible substrings ending at position i for all i , $1 \leq i \leq n$.

Next we establish the time complexity of Algorithm *FindAll*. With every iteration of each of the while loops in parts 1 and 2, a new reversible substring is found except for the last iteration. Therefore, the time spent in parts 1 and 2 is $O(\sum_{i=1}^n C_i)$ where C_i is the number of reversible substrings whose seeds start at position i . We note that by Corollary 1, C_i is bounded from above by the length of the longest reversible substring whose seed starts at i . Therefore, $C_i = O(n)$, $|R| = O(n^2)$, and Algorithm *FindAll* in the worst case takes $O(n^2)$ time. However, we show that on average $|R| = O(n)$, and the algorithm runs in $O(n)$ time.

Let $P_{i,k}^{odd}$ be the probability that there is a reversible substring of length $2k + 1$ centered at position i in X , i.e. $X_{(i-k)..(i+k)} = \overline{Y_{(i-k)..(i+k)}}$ for all k , $0 \leq k \leq \lceil n/2 \rceil$. We can easily see that $P_{i,k}^{odd} = \frac{1}{|\Sigma|^{2k+1}}$.

Let $C_{i,k}^{odd}$ be a 0 – 1 random variable such that $C_{i,k}^{odd} = 1$ indicates that there is a reversible sub-

string of length $2k + 1$ centered at position i , otherwise, $C_{i,k}^{odd} = 0$. Define C_i^{odd} as a random variable that denotes the number of reversible substrings whose seeds are at position i . We note that although $P_{i,j,k}$'s are not independent for different i, j, k because of overlapping of substrings, the linearity of expectation does not require independence, and the expected value of C_i^{odd} for $|\Sigma| \geq 2$ is

$$E(C_i^{odd}) = \sum_{k=0}^{\min\{i-1, \lfloor n/2 \rfloor\}} \frac{1}{|\Sigma|^{2k+1}} \quad (2)$$

Similarly, let $P_{i,k}^{even}$ be the probability that there is a reversible substring of length $2k$ whose length-two seed starts at position i in X , i.e. $X_{(i-k)..(i+1+k)} = \overline{Y_{(i-k)..(i+1+k)}}$ for all k , $0 \leq k \leq \lfloor n/2 \rfloor$. We can easily see that $P_{i,k}^{even} = \frac{1}{|\Sigma|^{2k+2}}$.

Let $C_{i,k}^{even}$ be a 0 – 1 random variable that denotes if there is a reversible substring of length $2k$ whose seed starts at position i : $C_{i,k}^{even} = 1$ indicates that there is such a reversible substring, if $C_{i,k}^{odd} = 0$ then there is not such a reversible substring. Define C_i^{even} as the number of reversible substrings whose seeds start at position i . Then for $|\Sigma| \geq 2$, the expected value of C_i^{even} is

$$E(C_i^{even}) = \sum_{k=0}^{\min\{i-1, \lfloor n/2 \rfloor\}} \frac{1}{|\Sigma|^{2k+2}} \quad (3)$$

Define C as a random variable that denotes the number of reversible substrings of X with respect to Y . Then the expected value of C

$$E(C) = E\left(\sum_{i=1}^n C_i^{odd}\right) + E\left(\sum_{i=1}^{n-1} C_i^{even}\right) \quad (4)$$

Using equations (2), (3), and (4) we get

$$\begin{aligned} E(C) &= E\left(\frac{1}{|\Sigma|} \sum_{k=0}^{\min\{i-1, \lfloor n/2 \rfloor\}} \frac{1}{|\Sigma|^k}\right) \\ &< \frac{n}{2} \frac{1}{|\Sigma|} \sum_{k=0}^{\infty} \frac{1}{|\Sigma|^k} \\ &= \frac{n}{2} \frac{1}{|\Sigma| - 1} \end{aligned} \quad (5)$$

Inequality (5) proves that the expected size $|R|$ of the set R of the lengths of all possible reversible substrings of X with respect to Y is $O(n)$, and Algorithm *FindAll* in Figure 3 takes $O(n)$ expected time. We remark that the inequality (5) does not immediately yield a $O(n)$ time algorithm for finding set R without applying the rule in Figure 1. For example, if we naively try all possible reversal lengths at all possible end positions i , $1 \leq i \leq n$, then this takes $\Theta(n^2)$ time to find set R although $|R| = O(n)$ because with respect to the end positions i there can be a sequence of reversal lengths whose lengths are arbitrary (not necessarily consecutive), and we need to check all possible $\Theta(n)$ reversal lengths performing $\Theta(n^2)$ comparisons in total.

4 Conclusion

We consider the edit distance model introduced and studied by Muthukrishnan and Sahinalp [7, 8]. This model allows substitutions at character level, and reversal of substrings in one string when these reversals result in exact matches in the other string. As it is the case in general, multiple edits in the same position are not permitted. The cost of each edit operation is 1. The edit distance under this model can be computed in $O(n \log^2 n)$ time where n is the common length of the given strings X and Y for which the distance is computed. For the problem we give a dynamic programming formulation which uses the set of lengths of all possible reversible substrings of X with respect to Y . In our formulation, the substitution costs can vary, and the costs of reversals can be a function of the reversal-length. We develop an algorithm which generates this information in $O(n^2)$ time. However, we prove that the expected running time of our algorithm is $O(n)$, and our dynamic programming solution for computing the edit distance takes $O(n)$ expected time. The gap between the worst-case time complexity $O(n \log^2 n)$ by Muthukrishnan and Sahinalp [8], and the achievement of $O(n)$ expected time by our algorithm is a motivation for future research

for possible improvement of the worst-case time complexity of the problem.

References

- [1] A. N. Arslan. An algorithm for string edit distance allowing substring reversals. *Proc. IEEE 6th Symposium on Bioinformatics and Bioengineering (BIBE)*, Washington D.C., October 16-18, 2006 (to appear).
- [2] D. Gusfield. Algorithms on strings, trees, and sequences: computer science and computational biology. *Cambridge University Press*, 1997.
- [3] S. Hannenhalli and P. A. Pevzner. Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. *Journal of the ACM*, (46)1:1–27, 1999.
- [4] M. Jackson, T. Strachan and G. Dover. Human genome evolution. *Bios Scientific Publishers*, 1996.
- [5] D. K. Kim, J.-S. Lee, K. Park and Y. Cho. Efficient algorithms for approximate string matching with swaps. *Journal of Complexity*, 15:128-147, 1999.
- [6] D. Lopresti and A. Tomkins. Block edit models for approximate string matching. *Theoretical Computer Science*, 181(1):159-179, 1997.
- [7] S. Muthukrishnan and S. C. Sahinalp. Approximate nearest neighbors and sequence comparison with block operations. *Proc. ACM Symposium on Theory of Computing*, 2000.
- [8] S. Muthukrishnan and S. C. Sahinalp. An improved algorithm for sequence comparison with block reversals. *Theoretical Computer Science*, 2004.
- [9] P. Sellers. The theory and computation of evolutionary distances: pattern recognition. *Journal of Algorithms*, 1, pp. 359-373, 1980.
- [10] D. Sankoff and J. Kruskal. Time warps, string edits, and macromolecules: the theory and practice of sequence comparison. *Addison-Wesley*, Reading, Massachusetts, 1983.
- [11] D. Shapira and J. A. Storer. Large edit distance with multiple block operations. *SPIRE 2003, LNCS 2857*, pp. 369–377, 2003.
- [12] R.A. Wagner and M.J. Fisher. The string-to-string correction problem. *J. ACM*, 21:168–173, 1974.
- [13] M. S. Waterman. Introduction to computational biology. *Chapman & Hall*, 1995.