Reversal Distance for Strings with Duplicates: Linear Time Approximation using Hitting Set

Petr Kolman*
Charles University in Prague
Faculty of Mathematics and Physics
Department of Applied Mathematics
kolman@kam.mff.cuni.cz

Tomasz Waleń
Warsaw University
Faculty of Mathematics, Informatics
and Mechanics
walen@mimuw.edu.pl

Abstract

In the last decade there has been an ongoing interest in string comparison problems; to a large extend the interest was stimulated by genome rearrangement problems in computational biology but related problems appear in many other areas of computer science. Particular attention has been given to the problem of *sorting by reversals* (SBR): given two strings, A and B, find the minimum number of reversals that transform the string A into the string B (a *reversal* $\rho(i,j)$, $i < j$, transforms a string $A = a_1 \ldots a_n$ into a string $A' = a_1 \ldots a_{i-1} a_j a_{j-1} \ldots a_i a_{j+1} \ldots a_n$).

Primarily the problem has been studied for strings in which every symbol appears exactly once (that is, for permutations) and only recently attention has been given to the general case where duplicates of the symbols are allowed. In this paper we consider the problem

*Supported by project 1M0021620808 of MŠMT ČR.

k-SBR, a version of SBR in which each symbol is allowed to appear up to k times in each string, for some $k \geq 1$. The main result of the paper is a $\Theta(k)$-approximation algorithm for k-SBR running in time $O(n)$; compared to the previously known algorithm for k-SBR, this is an improvement by a factor of $\Theta(k)$ in the approximation ratio, and by a factor of $\Theta(k)$ in the running time. Crucial ingredients of our algorithm are the suffix tree data structure a linear time algorithm for a special case of a disjoint set union problem.

Key words. Approximation algorithms, String comparison, Sorting by reversals, Minimum common string partition, Suffix trees.

1 Introduction

In the last decade there has been an ongoing interest in string comparison problems. To a large extend the interest was stimulated by genome rearrangement problems in computational biology but related problems appear in many other areas of computer science, in data compression or text processing to name a few. One of the important problems is to measure the similarity of two strings. Particular attention has been given to the problem of *sorting by reversals* (SBR): given two strings, A and B, find the *reversal distance* of A and B, which is the minimum number of reversals that transform the string A into the string B where a *reversal* $\rho(i,j)$, $1 \leq i < j \leq n$, is an operation that transforms a string $A = a_1 \dots a_n$, into a string $A' = a_1 \dots a_{i-1} a_j a_{j-1} \dots a_i a_{j+1} \dots a_n$ (that is, the reversal $\rho(i,j)$ reverses the order of symbols in the substring $a_i \dots a_j$ of A). In the case of signed strings, each symbol is given a sign $+$ or $-$, and the reversal operation also flips the sign of each symbol in the reversed substring.

Primarily the problem has been studied for strings in which every symbol appears exactly once (that is, for permutations); even in this setting the problem is NP-hard for unsigned permutations [2] and, surprisingly, the problem is in P for signed permutations [10]. Only recently attention has been given also to the general case where duplicates of the symbols are allowed. We denote by k-SBR the version of SBR in which each symbol is allowed to appear up to k times in each string, for some $k \geq 1$. Christie and Irving [4] prove that unsigned SBR is NP-hard for binary strings and Chen et al. [3] show that 2-SBR is NP-hard. The best approximation ratio for the general signed SBR is $O(\log n \log^* n)$ (following from the work of Cormode and Muthukrishnan [6]); there are $O(1)$-approximation algorithms for signed 2-SBR and 3-SBR [3, 5, 9]. Kolman [11] describes a greedy-like $O(k^2)$-approximation algorithm for k-SBR running in $O(kn)$ time. Most of the above mentioned algorithms exploit the close relation between the minimum common string partition problem (see below for definition) and the problem of sorting by reversals: they find an approximation for the static problem MCSP and turn it into a solution for SBR; this is also the approach that we take in this paper. For an overview of other related results and for more details about the relation between MCSP and SBR, we refer to the paper [11].

The main results of this paper are $\Theta(k)$-approximation algorithms for k-MCSP and k-SBR running in time $O(n)$; compared to the previously known algorithms for k-MCSP and k-SBR, this is an improvement by a factor of $\Theta(k)$ in the approximation ratio, and by a factor of $\Theta(k)$ in the running time.

On a high level, the algorithm works as follows: given the strings A and B, the algorithm turns them into an instance of the minimum hitting set problem and, exploiting special properties of the instance, it computes an approximation of the minimum hitting set which is in turn transformed into an approximate solution for k-MCSP; a solution for k-SBR is obtained from a solution of the relevant k-MCSP problem by the standard technique mentioned above. Crucial ingredients of the algorithm are a linear time procedure for construction of a suffix tree [7] and a linear time algorithm for a special case of a disjoint set union problem [8].

1.1 Notation

We stick to the notation used in the previous paper on k-SBR [11]. For a (signed or unsigned) string $P = a_1 \dots a_n$, we denote by $-P$ the result of reversal $\rho(1, n)$ of P (e.g., for $P = +a + b - d$, we have $-P = +d - b - a$; for $P = abd$, we have $-P = dba$). We say that two (signed or unsigned) strings $A = a_1 a_2 \dots a_n$ and $B = b_1 b_2 \dots b_n$ are *identical*, $A = B$, if $a_i = b_i$ for each $i \in 1, \dots, n$ (in the case of signed strings, $a_i = b_i$ involves also the equality of the signs), and they are *congruent*, $A \cong B$, if $A = B$ or $A = -B$ (note that for the sake of notational simplicity we overload the sign $\cong$ so that it has a slightly different meaning for signed and unsigned strings).

Throughout the paper we assume that the symbols are represented by integers from the set $\Sigma = \{1, 2, \dots, n\}$. We also assume that each symbol appears the same number of times in A and B (for the signed version, we count together the occurrences of a symbol with positive and negative signs). Clearly, this is a necessary and sufficient condition for A and B to have a finite reversal distance. We call such strings *related*.

The length of a string A is denoted by $|A|$. A *duo* is a string of length two. A *partition* of a string A is a sequence $\mathcal{P} = (P_1, P_2, \ldots, P_m)$ of strings whose concatenation is equal to A, that is, $P_1 P_2 \ldots P_m = A$. The strings P_i are called the *blocks* of $\mathcal{P}$ and their number is the *size* of the partition. Given a partition $\mathcal{P} = (P_1, P_2, \ldots, P_m)$, if $l = \sum_{j=1}^{i} |P_j|$ for some $i \in \{1, 2, \ldots, m-1\}$, we say that the pair $l, l+1$ is a *break* of the partition $\mathcal{P}$ and $a_l a_{l+1}$ is a *broken duo* of the partition $\mathcal{P}$.

For two strings A and B, we say that S is a *common substring with respect to the relation* $=$ if S is a substring of A and a substring of B; we say that S is a *common substring with respect to the relation* $\cong$, if S is a substring of A and there exists a substring R of B such that $S \cong R$, or S is a substring of B and there exists a substring R of A such that $S \cong R$. When not necessary, we will often avoid specifying the relation and will talk only about a common substring.

To *cut* a duo $a_i a_{i+1}$ of a block $P = a_j \ldots a_k$ of a partition of A, for some $j \leq i < k$, means to replace the block P in the partition by two blocks $P_1 = a_j \ldots a_i$ and $P_2 = a_{i+1} \ldots a_k$. For a string $C = c_1, \ldots, c_n$, we denote by $duos(C)$ the set of duos of the string C, that is, $duos(C) = \{c_i c_{i+1} \mid 1 \leq i \leq n-1\}$.

SBR is closely related to the minimum common string partition problem. Given a partition $\mathcal{P}$ of a string A and a partition $\mathcal{Q}$ of a string B, we say that the pair $\pi = (\mathcal{P}, \mathcal{Q})$ is a *common partition* of A and B with respect to the relation $\mathsf{Rel} \in \{=, \cong\}$, if there exists a permutation σ on $1, \ldots, m$ such that for each $i \in 1, \ldots, m$, $(P_i, Q_{\sigma(i)}) \in \mathsf{Rel}$. The *minimum common string partition problem* (MCSP) is to find a common partition of A, B with the minimum size, denoted by $\mathsf{MCSP}(A, B)$. The restricted version of MCSP, where each letter occurs at most k times in each input string, is denoted by k-MCSP. Similarly as for SBR, there is a signed and an unsigned variant of the problem; another variant, important for the unsigned SBR, is called reversed MCSP. Chen et al. [3] observed that for any two related signed strings A and B, the sizes of the optimal solutions of MCSP and SBR differ only by a constant multiplicative factor. An analogous observation applies for related unsigned strings and the problems reversed MCSP and SBR; we refer to the paper [11] for further details.

The rest of the paper is organized as follows. Section 2 is devoted to a simple algorithm for k–MCSP that is based on the Hitting Set problem. In Section 3 we describe how to modify the algorithm to get an $O(k)$ approximation for k-MCSP. In Section 4 we deal with the running time of the algorithm and we show how to implement the algorithm in linear time, using the suffix tree data structure. Finally, Section 5 describes how to modify the algorithm so that it works also for the signed and reversed variants of MCSP and thus, for signed and unsigned SBR.

2 Common partition via hitting set

In *Minimum Hitting Set Problem*, we are given a set U and a collection $\mathcal{S}$ of subsets of U, that is, $\mathcal{S} = \{S_1, \ldots, S_k\}$ such that $S_i \subseteq U$ for $i = 1, \ldots, k$. The task is to find a *minimum hitting set* for $\mathcal{S}$ which is a smallest set $H \subseteq U$ such that $H \cap S_i \neq \emptyset$ for each $i \in 1, \ldots, k$. Minimum Hitting Set problem is equivalent to Minimum Set Cover [1].

We are going to use an algorithm for Minimum Hitting Set Problem as a procedure for MCSP. The idea behind the algorithm is simple. Given the strings A and B and a string P such that the number of occurrences of P in A is larger (or smaller, resp.) than the number of occurrences of P in B, we know that even in the minimum common partition of A and B at least one duo in (an occurrence of) P in A (or in B, resp.) must be broken. The algorithm aims at "hitting" (that is, cutting) all substrings of A and B that have a different number of occurrences. This motivates the following definition.

For two strings A and S, let $\#\mathsf{substr}(A, S)$ be the number of occurrences of the substring S in the string A. For a partition $\mathcal{P} = (P_1, P_2, \ldots, P_m)$ and a string S, we denote by $\#\mathsf{blocks}(\mathcal{P}, S)$ the number of blocks $P_i = S$ in $\mathcal{P}$.

Algorithm HS

input: strings A, B
construct an instance $(U, \mathcal{S})$ of the Hitting Set problem:
  $U \leftarrow duos(A) \cup duos(B)$
  $T \leftarrow \{X \in \Sigma^* \mid \#\mathsf{substr}(A, X) \neq \#\mathsf{substr}(B, X)\}$
  $\mathcal{S} \leftarrow \{duos(X) \mid X \in T\}$
solve (approximately) the Minimum Hitting Set problem:
  $\Phi \leftarrow$ a hitting set for $(U, \mathcal{S})$
transform the hitting set into a common partition:
  $\mathcal{A}, \mathcal{B} \leftarrow$ for each duo $xy \in \Phi$,
    cut all occurrences of xy in the strings A, B
output: $(\mathcal{A}, \mathcal{B})$

Lemma 2.1 *The partition $(\mathcal{A}, \mathcal{B})$ computed by the algorithm* HS *is a common partition of the strings A and B.*

Proof: The proof is by contradiction. Suppose that there exists a block $X \in \mathcal{A}$ such that $\#\mathsf{blocks}(\mathcal{A}, X) \neq \#\mathsf{blocks}(\mathcal{B}, X)$; if there are several such blocks, take as X the longest one. Since the block X is not cut by any duo from Φ we have $duos(X) \cap \Phi = \emptyset$, and since Φ is a correct answer for the Hitting Set problem, it holds that $duos(X) \notin \mathcal{S}$. We conclude that $\#\mathsf{substr}(A, X) = \#\mathsf{substr}(B, X)$. We aim to get a contradiction by inferring an equality for $\#\mathsf{blocks}(\mathcal{A}, X)$ and $\#\mathsf{blocks}(\mathcal{B}, X)$.

Exploiting the fact that X is not cut by any duo from Φ, it is possible to calculate the numbers $\#\mathsf{blocks}(\mathcal{A}, X)$ and $\#\mathsf{blocks}(\mathcal{B}, X)$ by the following formula (by $X \sqsubseteq Y$ we denote that X is a substring of Y and by $X \sqsubset Y$ that X is a proper substring of Y):

$$\#\mathsf{blocks}(\mathcal{A}, X) = \#\mathsf{substr}(A, X) - \sum_{Y \sqsubseteq A, X \sqsubset Y} \#\mathsf{substr}(Y, X) \cdot \#\mathsf{blocks}(\mathcal{A}, Y)$$

$$\#\mathsf{blocks}(\mathcal{B}, X) = \#\mathsf{substr}(B, X) - \sum_{Y \sqsubseteq B, X \sqsubset Y} \#\mathsf{substr}(Y, X) \cdot \#\mathsf{blocks}(\mathcal{B}, Y)$$

By our choice, X is the longest block with $\#\mathsf{blocks}(\mathcal{A}, X) \neq \#\mathsf{blocks}(\mathcal{B}, X)$ (informally, a "wrong" block); therefore for all strings Y satisfying $X \sqsubset Y$ we have $\#\mathsf{blocks}(\mathcal{A}, Y) = \#\mathsf{blocks}(\mathcal{B}, Y)$. We conclude that $\#\mathsf{blocks}(\mathcal{A}, X) = \#\mathsf{blocks}(\mathcal{B}, X)$, which is a contradiction. □

Lemma 2.2 *The algorithm* HS *finds a $2k$-approximation of the minimum common partition (if an procedure for a minimum hiting set is available).*

Proof: Consider any common partition $\mathcal{A}', \mathcal{B}'$ of A and B. Then, every duo in a minimum hitting set for the instance $(U, \mathcal{S})$ must appear as a broken duo in $\mathcal{A}'$ od $\mathcal{B}'$. The claim follows. □

Observe that by replacing the optimal procedure for Minimum Hitting Set by an α-approximation procedure, the algorithm HS finds a $2k\alpha$-approximation of the minimum common partition.

Unfortunately, Minimum Hitting Set problem is hard to approximate; to achieve a good approximation ratio, we need to investigate special properties of the instance $(U, \mathcal{S})$. This is the subject of the next section.

3 $O(k)$-Approximation ratio for MCSP

Let $(\mathcal{A}_o, \mathcal{B}_o)$ denote a minimum common partition of strings A and B (if there are several minimum common partitions, we choose any of them); we say that the breaks in $\mathcal{A}_o$ and $\mathcal{B}_o$ are the *optimal breaks.* There are $2|\mathcal{A}_o|-2$ optimal breaks. We say that a substring $X = a_i \dots a_j$ (resp., $X = b_i \dots b_j$) *goes over* an optimal break if there exists an optimal break $l, l+1$ in $\mathcal{A}_o$ (resp., in $\mathcal{B}_o$) such that $i \leq l < j$.

Recall the definition of the set T

$T = \{X \in \Sigma^* \mid \#\mathsf{substr}(A, X) \neq \#\mathsf{substr}(B, X)\}$;

informally, T is the set of all wrong substrings. Note that in the instance of the Hitting Set problem, most of the substrings in T are obsolete. To be more specific, if $X, Y \in T$ and X is a proper substring of Y, then we can remove Y from the set T and a hitting set for $\{duos(X) \mid X \in T \setminus \{Y\}\}$ will still be a hitting set for $\mathcal{S}$. Using this observation it is possible to substantially reduce the size of the set $\mathcal{S}$. In particular, the relation $\sqsubseteq$ induces a partial order on the set T; let $T_{\min} \subseteq T$ be the set of all minimal elements of T, with respect to the relation $\sqsubseteq$. Then $T_{\min}$ satisfies the desired property

(P) if $X, Y \in T$, and X is a proper substring of Y, then $Y \notin T_{\min}$,

and, at the same time, a hitting set for the set $\mathcal{S}' = \{duos(X) \mid X \in T_{\min}\}$ is a hitting set for $\mathcal{S}$.

Lemma 3.1 *If $X \in T_{\min}$ then there exists an occurrence of X in A or in B that goes over an optimal break.*

Proof: Consider a string $X \in T_{\min}$ and suppose that no occurrence of X in A and B goes over an optimal break. Then every occurrence of X in A or B is a substring of some block in the minimum common partition $(\mathcal{A}_o, \mathcal{B}_o)$. Since $\mathcal{A}_o$ and $\mathcal{B}_o$ consists of the same multiset of blocks and no occurrence of X goes over an optimal break, we have $\#\mathsf{substr}(A, X) = \#\mathsf{substr}(B, X)$. This implies $X \notin T$, which is a contradiction. $\square$

Using the lemma, we assign to each string in $T_{\min}$ an optimal break. In particular, for $X \in T_{\min}$, let $f(X)$ denote the optimal break that an occurrence of X in A or in B goes over; if there is more than one such optimal break, we choose an arbitrary one.

Example: For $A = abaab$, $B = ababa$, the minimum common partition is $(aba, ab), (ab, aba)$, $ba \in T_{min}$ and $f(ba) =$ the break $2, 3$ in the partition of B.

Lemma 3.2 *If $X, Y \in T_{\min}$, $X = x_1, \ldots, x_l$ and $f(X) = f(Y)$, then $duos(Y) \cap \{x_1x_2,\ x_{l-1}x_l\} \neq \emptyset$.*

Proof: Since X and Y go over the same optimal break, their overlap has size at least two. Moreover, since X is not a proper substring of Y and vice versa (by property (P) and the assumptions of the lemma), the claim follows (cf. Figure 1). $\Box$

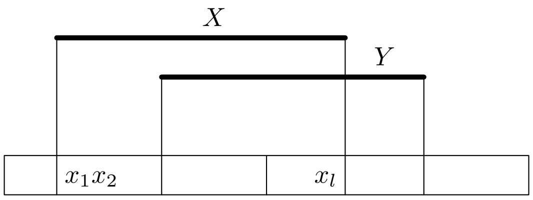

Figure 1: Illustration of Lemma 3.2

The consequence of Lemma 3.2 is the following. Let $\mathcal{A}$ be a partition of A and $\mathcal{B}$ be a partition of B and let $X = x_1 \ldots x_l$ be a common substring of $\mathcal{A}$ and $\mathcal{B}$ such that $X \in T_{\min}$. Then, by cutting all occurrences of x_1x_2 and $x_{l-1}x_l$ in $\mathcal{A}$ and $\mathcal{B}$ we "hit" (that is, we cut) also (a duo in) each string from $T_{\min}$ that goes over the optimal break $f(X)$. Thus, if we choose for each optimal cut one string from T_{min} that goes over it (if there is any such string for the cut; if there is no such string, we ignore this cut) and put together the first and the last duos of each such string, then we get a hitting set for $T_{\min}$ of size at most twice the size of the minimum hitting set. Of course, we do not know the optimal breaks so we have to construct the hitting set in a different way. The following algorithm does it:

Algorithm Fast HS

input: strings A, B
  compute a set T' such that $T_{\min} \subseteq T'$ and T' is of size $O(n)$
  $\Phi \leftarrow \emptyset$
  $\mathcal{A} \leftarrow (A)$, $\mathcal{B} \leftarrow (B)$
  for each $X \in T'$ in order of increasing length **do**
    if $duos(X) \cap \Phi = \emptyset$ **then**
      add the first and last duo of X to Φ
      cut all occurrences of the first and last duo of X
       in the partitions $\mathcal{A}, \mathcal{B}$
output: $(\mathcal{A}, \mathcal{B})$

Lemma 3.3 *If a string X passes the test $duos(X) \cap \Phi = \emptyset$ in the above algorithm, then $X \in T_{\min}$.*

Proof: Suppose, for a contradiction, that X passed the test yet $X \notin T_{min}$. Let Φ' denote the set Φ just before processing the string X. The assumption $X \notin T_{min}$ implies that there exists a string $X' \in T_{\min}$ such that X' is a proper substring of X. Since $|X'| < |X|$, the string X' has been processed before the string X and therefore $duos(X') \cap \Phi' \neq \emptyset$. Moreover, since $duos(X') \subseteq duos(X)$, it holds that $duos(X) \cap \Phi' \neq \emptyset$, and therefore X cannot pass the test, which is a contradiction. □

Theorem 3.4 *The algorithm* Fast HS *computes a $4k$-approximation of the minimum common partition of A and B.*

Proof: If X_1, X_2 are two different strings for which the set Φ was increased then, by Lemma 3.2, $f(X_1) \neq f(X_2)$. Thus, the set Φ was increased at most $|\mathcal{A}_o| + |\mathcal{B}_o| - 2$ times and therefore the final set Φ contains at most $2 \cdot (|\mathcal{A}_o| + |\mathcal{B}_o| - 2)$ duos.

Since we are dealing with an instance of k-MCSP, each duo from the set Φ introduces at most k cuts. It follows that

$$|\mathcal{A}| \leq k \cdot 2 \cdot (|\mathcal{A}_o| + |\mathcal{B}_o| - 2) + 1 \leq 4k \cdot |\mathcal{A}_o| \quad .$$

□

Remark: The approximation ratio applies even if we measure the size of a common partition not by the number of blocks but by the number of breaks.

Lower bound. Let $A = ba\{ab\}^{k-1}$ and $B = \{ab\}^k$. Then the set Φ consists of two duos $\{aa, ab\}$ and the partition computed by the algorithm Fast HS has size $k+1$ while the minimum common partition has size 3.

4 Linear running time

We are going to describe how to implement the algorithm in linear time. The linear implementation heavily uses the *suffix tree* data structure and the fact that a suffix tree of a string of length m can be constructed in time $O(m)$ for constant size alphabets [12] and even for integer alphabets [7].

We start with the construction of the set T'. Let \$ and # be two characters that do not appear in A. We compute the suffix tree τ of the string $C = A\$B\#$. Recall that each leaf of the tree τ corresponds to a suffix of C. We mark by A each leaf of τ that corresponds to a suffix starting in the substring A of C, and we mark by B each leaf of τ that corresponds to a suffix starting in the substring B of C. For each node v of τ we compute the number $numA(v)$ of leaves in the subtree of v marked by A and the number $numB(v)$ of leaves in the subtree of v marked by B; this requires time $O(n)$, for strings A, B of length n. For a node v of τ, let $s(v)$ denote the concatenation of the labels of the edges between the root and the node v and, for $v \neq$root, let $s'(v)$ denote the concatenation of $s(parent(v))$ with the first character of the label of the edge $(parent(v), v)$. If $s'(v)$ does not contain the characters \$ and # we say that v is a *proper* node. Observe that for each proper node v, $numA(v) = \#\mathsf{substr}(A, s'(v))$ and $numB(v) = \#\mathsf{substr}(B, s'(v))$. Thus, if $numA(v) \neq numB(v)$ we know that $s'(v) \in T$. Once we have the suffix tree τ and the values $numA(v)$ and $numB(v)$ for all vertices, we easily compute a set

$$T' = \{s'(v) \mid v \text{ is a proper node and } numA(v) \neq numB(v)\}$$

by traversing the tree τ in, say, breadth first search order. Since the suffix tree consist of $O(n)$ nodes, the size of T' is bounded by $O(n)$. We also note that for each string $X \in T_{\min}$ there is a proper node v such that $s'(v) = X$ and $numA(v) \neq numB(v)$ which guarantees that $T_{\min} \subseteq T'$.

To give an example, consider strings $A = abaab$ and $B = ababa$. The suffix tree of the string $C = A\$B\#$ is given in Figure 2 and the relevant sets are as follows:

$$T' = \{aa, aba, abaa, abab, ba, baa, bab\}$$

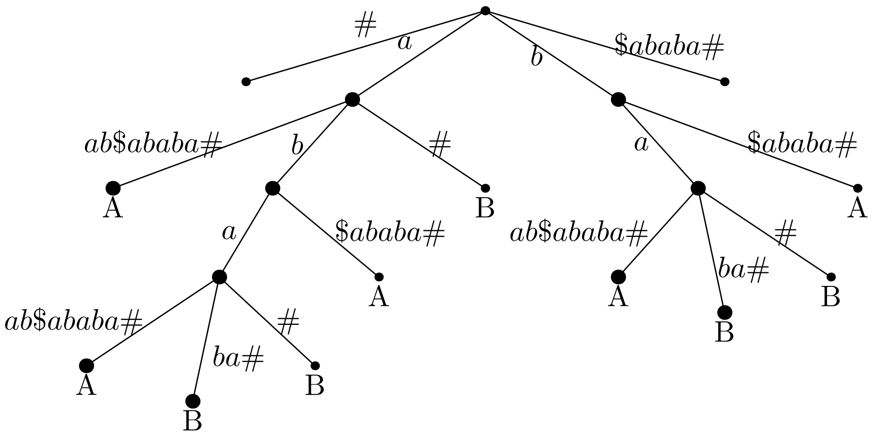

Figure 2: Suffix tree τ of the string $C = abaab\$ababa\#$. The larger dots denote the proper nodes.

$$\begin{aligned} T_{\min} &= \{aa, ba\} \\ \Phi &= \{aa, ba\} \\ \mathcal{A} &= (ab, a, ab) \\ \mathcal{B} &= (ab, ab, a) \end{aligned}$$

To finish the description of the fast implementation of the algorithm, it remains to describe how to maintain the set Φ, how to test the condition $duos(X) \cap \Phi \neq \emptyset$ and how to realize the cuts. We employ a data structure for the *set–splitting problem* [8]. In this problem, we are given a set consisting of the integers $\{1, \ldots, m\}$ and the task is to perform an intermixed sequence of the following two operations:

- $split(i)$ – splits the set containing i into two sets, one with all integers smaller than i and the other with all integers greater than or equal to i,
- $find(i)$ – returns the smallest integer in the set containing i.

Gabow and Tarjan [8] describe a data structure that requires $O(1)$ amortized time for each operation. In our setting, we maintain for each partition $\mathcal{A}$ and $\mathcal{B}$ a separate data structure that stores information about cuts in that partition. Initially, each structure consists of only one set, the set $\{1, \ldots, n\}$.

Each time when we add a duo cd to Φ we perform the cuts of the partitions $\mathcal{A}$ and $\mathcal{B}$ as follows:

for each occurrence of the duo cd in $\mathcal{A}$ **do**
  $A.split(j+1)$, where j is the position of the current occurrence cd in A (i.e., $a_j a_{j+1} = cd$)
for each occurrence of the duo cd in $\mathcal{B}$ **do**
  $B.split(j+1)$, where j is the position of the current occurrence cd in B (i.e., $b_j b_{j+1} = cd$)

Since every duo appearing in A and B is processed at most once by the algorithm the total number of *split* operations is at most $O(n)$.

For an occurrence $a_i \dots a_j = X$ (resp., $b_i, \dots, b_j = X$) of the substring $X \in T'$, it holds that $duos(X) \cap \Phi = \emptyset$ if and only if $A.find(i) = A.find(j)$ (resp., $B.find(i) = B.find(j)$). This provides a way for testing the condition $duos(X) \cap \Phi \neq \emptyset$ in constant time.

Theorem 4.1 *The above implementation of the algorithm* FAST HS *runs in linear time.*

5 Sorting by reversals

One can easily modify the algorithms HS and FAST HS to work also for the relation $\cong$, for both signed and unsigned strings. We redefine $\#\mathsf{substr}(A, S)$ to so that it counts occurrences of both S and $-S$ in A; the definitions of the sets T, T' and T_{min} remain unchanged. The new definition of $\#\mathsf{substr}$ requires a small change in the computation of the set T': we compute a suffix tree of the string $C = A\#B\$(-A)\#(-B)\$$ (the brackets are only used to denote the scope of the reversal operation). We also need a slight change in Lemma 3.1 and Lemma 3.2:

Lemma 3.1a If $X \in T_{\min}$ then there exists an occurrence of X or $-X$ in A or in B that goes over an optimal break.

Lemma 3.2a If $X, Y \in T_{\min}$, $X = x_1, \dots, x_l$ and $f(X) = f(Y)$, then $duos(Y) \cap \{x_1x_2,\ x_{l-1}x_l,\ -(x_1x_2),\ -(x_{l-1}x_l)\} \neq \emptyset$.

Finally, whenever the original algorithm cuts duos xy, the modified algorithm also cuts duos $-(xy)$. This increases the approximation ratio by a factor of two.

Theorem 5.1 *The algorithm* FAST HS *computes in linear time* $\Theta(k)$*-approximation for signed, unsigned and reversed* k-MCSP *and for signed and unsigned* k-SBR*.*

6 Conclusion

We presented $\Theta(k)$-approximation algorithms for signed and unsigned k-MCSP and k-SBR, running in time $O(n)$. A challenging open question is whether it is possible to get a nontrivial approximation ratio independent of the parameter k (or at least less dependent, say an approximation ratio $O(\log k)$).

References

[1] G. Ausiello, A. D'Atri, and M. Protasi. Structure preserving reductions among convex optimization problems. *Journal of Computer and System Sciences*, 21(1):136–153, 1980.

[2] A. Caprara. Sorting by reversals is difficult. In *Proceedings of the First International Conference on Computational Molecular Biology*, pages 75–83. ACM Press, 1997.

[3] X. Chen, J. Zheng, Z. Fu, P. Nan, Y. Zhong, S. Lonardi, and T. Jiang. Assignment of orthologous genes via genome rearrangement. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2(4):302–315, 2005.

[4] D. A. Christie and R. W. Irving. Sorting strings by reversals and by transpositions. *SIAM Journal on Discrete Mathematics*, 14(2):193–206, 2001.

[5] M. Chrobak, P. Kolman, and J. Sgall. The greedy algorithm for the minimum common string partition problem. *ACM Transactions on Algorithms*, 1(2):350–366, 2005.

[6] G. Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. In *Proceedings of the 13th Annual ACM-SIAM Symposium On Discrete Mathematics (SODA)*, pages 667–676, 2002.

[7] M. Farach. Optimal suffix tree construction with large alphabets. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 137–143, 1997.

[8] H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 246–251, 1983.

[9] A. Goldstein, P. Kolman, and J. Zheng. Minimum Common String Partition Problem: Hardness and Approximations. *The Electronic Journal of Combinatorics*, 12(1), Sept. 2005.

[10] 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.

[11] P. Kolman. Approximating reversal distance for strings with bounded number of duplicates. In *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 3618 of *Lecture Notes in Computer Science*, pages 580–590, 2005.

[12] P. Weiner. Linear pattern matching algorithms. In *14th IEEE Symposium on switching and automata theory*, pages 1–11, 1973.