# Approximating Reversal Distance for Strings with Bounded Number of Duplicates in Linear Time

Petr Kolman *

February 17, 2005

**Abstract**

For a string $A = a_1 \dots a_n$, a *reversal* $\rho(i,j)$, $1 \le i < j \le n$, transforms the string $A$ 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 a case of signed strings, where each symbol is given a sign + or −, the reversal operation also flips the sign of each symbol in the reversed substring. Given two strings, $A$ and $B$, signed or unsigned, *sorting by reversals* (SBR) is the problem of finding the minimum number of reversals that transform the string $A$ into the string $B$.

Traditionally, the problem was studied for permutations, that is, for strings in which every symbol appears exactly once. We consider a generalization of the problem, $k$-SBR, and allow each symbol to appear at most $k$ times in each string, for some $k \ge 1$. The main result of the paper is a simple $O(k^2)$-approximation algorithm running in time $O(k \cdot n)$. For instances with $3 < k \le O(\sqrt{\log n \log^* n})$, this is the best known approximation algorithm for $k$-SBR and, moreover, it is faster than the previous best approximation algorithm. In particular, for $k = O(1)$ which is of interest for DNA comparisons, we have a linear time $O(1)$-approximation algorithm.

**Key words.** Approximation algorithms, String comparison, Edit distance, Sorting by reversals.

## 1 Introduction

For a string $A = a_1 \dots a_n$, a *reversal* $\rho(i,j)$, $1 \le i < j \le n$, transforms the string $A$ 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 a case of signed strings, where each symbol is given a sign + or −, the reversal operation also flips the sign of each symbol in the reversed substring. Given two strings, $A$ and $B$, signed or unsigned, *sorting by reversals* (SBR) is the problem of finding the minimum number of reversals that transform the string $A$ into the string $B$; this number, denoted by $\mathsf{SBR}(A,B)$, is called the *reversal distance* of $A$ and $B$.

A necessary and sufficient condition for $A$ and $B$ to have a finite reversal distance is that each letter appears the same number of times in $A$ and $B$ (for the signed version, we count together the occurrences of a letter with positive and negative signs). We call such strings *related*.

To give an example, $A = abcabc$ and $B = bcbaac$ are related strings and $\rho(3,6), \rho(1,4)$ is a sequence of reversals that turns $A$ to $B$, therefore $\mathsf{SBR}(A,B) \le 2$. Similarly, $\rho(1,4), \rho(4,4)$ turns $A' = +a-c-b-a+b+c$ to $B' = +a+b+c+a+b+c$ and thus, $\mathsf{SBR}(A',B') \le 2$.

*Institute for Theoretical Computer Science, Charles University, Malostranské nám. 25, 118 00 Praha 1, Czech Republic. kolman@kam.mff.cuni.cz. Research done while visiting University of California at Riverside. Supported by project LN00A056 of MŠMT ČR and NSF grants CCR-0208856 and ACI-0085910.

In this paper we study a variant of the problem, denoted by $k$-SBR, in which each symbol is allowed to appear at most $k$ times in each string. Our particular interest is in the case that $k = O(1)$. The main contribution is a simple $O(k^2)$-approximation algorithm for $k$-SBR running in time $O(k \cdot n)$. Thus, for $k = O(1)$, we have a linear time $O(1)$-approximation algorithm.

### 1.1 Terminology

For notational simplicity, we allow a few symbols to have slightly different meanings for signed and unsigned strings. For a 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 $-P = +d - b - a$). We use two different equivalence relations. Two strings $A = a_1 a_2 \dots a_n$ and $B = b_1 b_2 \dots b_n$, signed or unsigned, are *identical*, $A = B$, if $a_i = b_i$ for each $i \in [n]$. In a case of signed strings, by $a_i = b_i$ we mean also equality of the signs. Signed or unsigned strings $A$ and $B$ are *congruent*, $A \cong B$, if $A = B$ or $A = -B$.

The length of a string $A$ is denoted by $|A|$. A *partition* of a string $A$ is a sequence $\mathcal{P} = (P_1, P_2, \dots, P_m)$ of strings whose concatenation is equal to $A$, that is, $P_1 P_2 \dots P_m = A$. The strings $P_i$ are called the *blocks* of $\mathcal{P}$ and their number is the *size* of the partition. Given a partition $\mathcal{P} = (P_1, P_2, \dots, P_m)$, of a string $A$, a pair $l, l+1$ is a *break* of the partition $\mathcal{P}$ if $l = \sum_{j=1}^{i} |P_j|$ for some $i \in [m-1]$. Informally, a break of a partition $\mathcal{P}$ of $A$ is a pair of letters that are consecutive in $A$ but are not consecutive in $\mathcal{P}$.

For two strings $A$ and $B$, we say that $S$ is a *common substring with respect to the relation* $=$ or $\cong$, respectively, if $S$ is a substring of $A$ and there exists a substring $R$ of $B$ such that $S = R$ or $S \cong R$, respectively. When not necessary, we will often avoid specifying the relation and will talk only about a common substring. If $S$ is a common substring of $A$, $B$, we use notations $S^A$ and $S^B$ to distinguish between the occurrences of $S$ (or $-S$) in $A$ and $B$. Given two partitions $\mathcal{A} = (A_1, \dots, A_m)$ and $\mathcal{B} = (B_1, \dots, B_{m'})$, a common substring of $\mathcal{A}$ and $\mathcal{B}$ is a string $S$ such that $S$ is a common substring of $A_i$ and $B_j$, for some indices $i, j$.

### 1.2 Related work

String comparison is a fundamental problem in computer science with applications in text processing, data compression or computational biology. The problem of sorting by reversals drew a lot of attention in the last years as a useful tool for DNA comparison [4, 12, 6, 1]. In that application, the letters in the strings represent different genes and the reversal distance measures the similarity of two genomic sequences. A common assumption that a genome contains only one copy of each gene is unwarranted for genomes with multi-gene families such as the human genome [14]. On the other hand, a weaker assumption that a genome contains at most $k = O(1)$ copies of each gene is often warranted (cf. [9]). That is why $k$-SBR is of interest. In this subsection we will briefly mention the most relevant known results.

Under the assumption that every symbol appears in each input string exactly once, we have the well known problem of permutation sorting by reversals. The problem 1-SBR is solvable in polynomial time for strings with signs [12, 1] but is NP-hard [4] and even MAX-SNP hard [3] for strings without signs; the best known approximation ratio for the unsigned 1-SBR is 1.375 by an algorithm of Berman et al. [2]. A recent result of Chen et al. [5] shows that the signed $k$-SBR is NP-hard even for $k = 2$ (the unsigned $k$-SBR is obviously NP-hard for all $k \geq 2$). There are $O(1)$-approximation algorithms for signed 2-SBR and 3-SBR [5, 7, 11]. The best approximation ratio for 2-SBR is 2.2074 and the algorithm relies on semidefinite programming [11]; the algorithm for 3-SBR runs in linear time and has an approximation ratio 8 [11]. The best approximation ratio

for the general signed SBR is $O(\log n \log^* n)$, using an $O(n \log^* n)$-time algorithm of Cormode and Muthukrishnan [8].

Instead of bounding the number of duplicates, there is another way to restrict the general problem of sorting by reversals with duplicates: bound the size of the alphabet. Unsigned SBR with unary alphabet is trivial; the NP-hardness of unsigned SBR with binary alphabet was proved by Christie and Irving [6].

Closely related is a *minimum common string partition* problem (MCSP). Given a partition $\mathcal{P}$ of a string $A$ and a partition $\mathcal{Q}$ of a string $B$, we say that the pair $\pi = \langle \mathcal{P}, \mathcal{Q} \rangle$ is a *common partition* of $A$ and $B$ with respect to the relation $\mathsf{Rel} \in \{=, \cong\}$, if there exists a permutation $\sigma$ on $[m]$ such that for each $i \in [m]$, $(P_i, Q_{\sigma(i)}) \in \mathsf{Rel}$. The minimum common string partition problem 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. In *unsigned* MCSP, the input consists of two unsigned strings, and the relation $=$ is used; in *signed* MCSP, the input consists of two signed strings and the relation $\cong$ is used. For unsigned strings, we define yet another variant of the problem, *reversed* MCSP (RMCSP), in which the (unsigned) strings are compared by the relation $\cong$.

The signed MCSP problem was introduced by Chen et al. [5] as a tool for dealing with SBR; they observed that for any two related signed strings $A$ and $B$, $\lceil (\mathsf{MCSP}(A, B) - 1)/2 \rceil \leq \mathsf{SBR}(A, B) \leq \mathsf{MCSP}(A, B) - 1$ (note that $\mathsf{MCSP}(A, B) - 1$ is the number of breaks in a minimum common partition). Analogously, it is possible to show that for any two related unsigned strings $A$ and $B$, $\lceil (\mathsf{RMCSP}(A, B) - 1)/2 \rceil \leq \mathsf{SBR}(A, B) \leq 2(\mathsf{RMCSP}(A, B) - 1)$. For $k \geq 2$, $k$-MCSP is NP-hard, and even APX-hard [11]. Due to the close relation between signed SBR and signed MCSP, the known approximation ratios for signed MCSP are within a factor of 2 of the approximation ratios for signed SBR: $O(1)$ approximation ratios for 2-MCSP and 3-MCSP [7, 11], $O(\log n \log^* n)$ approximation ratio for the general MCSP [8].

Chrobak et al. [7] analyzed the behavior of a natural greedy heuristic for MCSP: start with the two strings $A$ and $B$ and iteratively, find the longest common substring of $A$ and $B$ that does not overlap previously marked substrings, and mark this substring. They showed that though GREEDY is a 3-approximation algorithm for 2-MCSP, even for 4-MCSP its approximation ratio is $\Omega(\log n)$. For general MCSP, both signed and unsigned, the approximation ratio is between $\Omega(n^{0.43})$ and $O(n^{0.67})$. It is worth noting that the algorithms described in this paper are simple modifications of GREEDY, yet their approximation ratios for $k$-MCSP are better, namely $O(k^2)$, in contrast to the $\Omega(\log n)$ of GREEDY for $k \geq 4$.

In the *edit distance* problem, a set of string operations is given (e.g., DELETE, INSERT or CHANGE a character, SUBSTRING_MOVE or SUBSTRING_REVERSAL) and the task is to find the minimum number of operations needed to convert one string to the other. SBR can be also viewed as an edit distance problem where the only operation is SUBSTRING_REVERSAL and the input strings are related. For any two related strings $A$ and $B$, $\mathsf{MCSP}(A, B)$ differs by a constant multiplicative factor from the edit distance of $A$ and $B$ with only SUBSTRING_MOVE operations, and the edit distance using only SUBSTRING_MOVE operations differs also by a constant multiplicative factor from the edit distance with operations {INSERT, DELETE a character, SUBSTRING_MOVE } [15]. For the later problem, Cormode and Muthukrishnan [8] describe an $O(n \log^* n)$-time $O(\log n \log^* n)$-approximation algorithm which yields, by the relations described above, the $O(\log n \log^* n)$-approximation for SBR mentioned earlier in this subsection.

The edit distance problem with a different set of string operations was studied by Ergun et al. [10]. For several edit distance problems that allow SUBSTRING_DELETION, they describe an

$O(1)$ approximation algorithm. This is in contrast to the above mentioned known approximations of edit distance *without* SUBSTRING DELETION where the best approximation ratio is of order $\Omega(\log n \log^* n)$.

The rest of the paper is organized as follows. In Section 2.1 we describe how to modify GREEDY to get the $O(k^2)$ approximation for (reversed) $k$-MCSP and thus, for $k$-SBR. Section 2.2 explains how to implement the algorithm in time $O(k \cdot n)$.

# 2 Algorithms

## 2.1 REFINED GREEDY: $O(k^2)$-approximation

In the previous section, we briefly described GREEDY algorithm and we recalled that its approximation ratio for $k$-MCSP and $k$-SBR, for any $k \geq 4$, is $\Omega(\log n)$. In this section, we show that a simple modification of GREEDY, called REFINED GREEDY, has an $O(k^2)$ approximation ratio for $k$-MCSP, which implies also an $O(k^2)$ approximation ratio for $k$-SBR.

A few more terms are needed. A *duo* is a string of length two. To *cut* a duo $a_i a_{i+1}$ of a block $P = a_j \dots a_k$ of a partition of $A$, for some $j \leq i < k$, means to replace the block $P$ in the partition by two blocks $P_1 = a_j \dots a_i$ and $P_2 = a_{i+1} \dots a_k$. For a substring $S = a_i \dots a_j$ of $A = a_1 \dots a_n$, if $i > 1$ we say that $a_{i-1}a_i$ is a *(left) boundary* duo of $S$, and similarly, if $j < n$ $a_j a_{j+1}$ is a *(right) boundary* duo of $S$.

For unsigned $k$-MCSP the algorithm is the following:

**Algorithm** REFINED GREEDY
**Input:** two related strings $A$ and $B$
$\mathcal{A} \leftarrow (A)$, $\mathcal{B} \leftarrow (B)$
**while** there are unmarked blocks in $\mathcal{A}$ and $\mathcal{B}$ **do**
$S \leftarrow$ longest common substring of $\mathcal{A}$, $\mathcal{B}$ that does not overlap previously marked blocks
cut the boundary duos of $S^A$ in $\mathcal{A}$ and the boundary duos of $S^B$ in $\mathcal{B}$
mark $S^A$ in $\mathcal{A}$ and $S^B$ in $\mathcal{B}$
cut in unmarked blocks of $\mathcal{A}$ and $\mathcal{B}$ *all* occurrences of duos $\delta \in \Phi$, where $\Phi$ is the set of boundary duos of $S^A$ and $S^B$
**Output:** $(\mathcal{A}, \mathcal{B})$

To extend the algorithm for signed $k$-MCSP and for $k$-RMCSP, apart from considering common substrings with respect to the other equivalence relation $\cong$, the difference is that in the cutting steps, we cut not only all occurrences of $\delta \in \Phi$ but also all occurrences of $-\delta$.

To give an example, consider an instance of MCSP,

$$A = abcccca f ccccddddIe f ccccebccccgggg \ , \ B = abccccddddaf ccccIe f ccccggggebcccc \ .$$

REFINED GREEDY first marks substring $S_1 = ccccdddd$ (we use overline to denote marking in this example) and cuts all unmarked occurrences of duos from $\Phi = \{fc, dI, bc, da\}$. In the second iteration, REFINED GREEDY looks for the longest unmarked substring in partitions $\mathcal{A} = (ab, ccccaf, \overline{ccccdddd}, Ief, cccceb, ccccgggg)$ and $\mathcal{B} = (ab, \overline{ccccdddd}, af, \ ccccIef, ccccggggeb, cccc)$, marks substring $S_2 = ccccgggg$ and cuts duos from $\Phi = \{ge\}$. In the third iteration, REFINED GREEDY looks for the longest unmarked substring in partitions $\mathcal{A} = (ab, ccccaf, \overline{ccccdddd}, Ief, cccceb, \overline{ccccgggg})$

and $\mathcal{B} = (ab, \overline{ccccdddd}, af, ccccIef, \overline{ccccgggg}, eb, cccc)$, marks substring $S_3 = cccc$ and cuts duos from $\Phi = \{ca, cI\}$. Eventually, Refined Greedy outputs the common partition

$$\mathcal{P} = \langle (ab, cccc, af, ccccdddd, Ief, cccc, eb, ccccgggg), (ab, ccccdddd, af, cccc, Ief, ccccgggg, eb, cccc) \rangle \ .$$

The optimal common partition has six blocks:

$$\mathcal{P}_{\text{OPT}} = \langle (abcccc, afcccc, dddd, Iefcccc, ebcccc, gggg), (abcccc, dddd, afcccc, Iefcccc, gggg, ebcccc) \rangle \ .$$

Before analyzing Refined Greedy, let us briefly look on the behavior of Greedy on the same instance. The longest common substrings of $A$ and $B$ are *ccccdddd* and *ccccgggg*, therefore Greedy starts by matching these substrings in the first two iterations. We observe that there exists a common partition of $A$ and $B$ that uses *ccccdddd* and *ccccgggg* as blocks:

$$\mathcal{P}' = \langle (ab, cccc, af, ccccdddd, Ief, cccc, eb, ccccgggg), (ab, ccccdddd, af, cccc, Ief, ccccgggg, eb, cccc) \rangle \ .$$

Every common partition induces a matching between the letters (positions) of $\mathcal{A}$ and $\mathcal{B}$. We note that the common partition $\mathcal{P}'$ matches many of the letters of $\mathcal{A}$ and $\mathcal{B}$ in the same way as the optimal partition $\mathcal{P}_{\text{OPT}}$ does. However, after several steps Greedy will find another common partition:

$$\mathcal{P}_{\text{GR}} = \langle (a, bcccc, a, f, ccccdddd, Ie, fcccc, e, b, ccccgggg), (a, b, ccccdddd, a, fcccc, Ie, f, ccccgggg, e, bcccc) \rangle \ .$$

Intuitively, the problem of Greedy is that a wrong decision in *one* iteration can force the use of *several* additional iterations, and in each of them Greedy may do another wrong decision, and so on. In other words, a deviation from the optimal solution in one iteration encourages deviations in later iterations. In our instance, after the first two iterations, it is still desirable, for example, to match the first $b$ from $A$ with the first $b$ from $B$, as the common partition $\mathcal{P}'$ does. However, since *bcccc* is the longest common substring at this point, Greedy will decide to use the wrong match between the first $b$ from $A$ and the third $b$ from $B$.

To improve the performance of the algorithm, the idea is to prevent it from propagating "mistakes" from one iteration to later iterations. In our example, the first mistake was to use the substrings *ccccddd*; a consequence of this was the use of the substrings *bcccc*, another mistake. Refined Greedy attempts to suppress this problem by cutting a few additional duos that are related to the current longest common substring, in each iteration. These breaks will constrain later iterations and will confine the propagation of mistakes.

**Theorem 2.1** Refined Greedy *is a* $2k^2$*-approximation algorithm for unsigned and signed* $k$-MCSP *and* $2(2k-1)^2$*-approximation for* $k$-RMCSP.

*Proof:* The output of the algorithm is clearly a common partition. We only have to prove the bound on its quality. For simplicity of the presentation, we prove the claim in detail for the unsigned $k$-MCSP and then we briefly outline the necessary modifications for signed $k$-MCSP and for $k$-RMCSP.

For technical reasons, it will be convenient to extend the notions of a partition and a common partition from strings to multisets of strings. A *partition of the multiset* of strings $\mathcal{A} = \{A_1, \ldots, A_l\}$ is a sequence of strings $A_{1,1}, \ldots, A_{1,k_1}, A_{2,1}, \ldots, A_{2,k_2}, \ldots, A_{l,1}, \ldots, A_{l,k_l}$, such that $A_i = A_{i,1} \ldots, A_{i,k_i}$ for $i \in [l]$. For two multisets of strings, the common partition is defined analogously as for pairs of strings.

**Observation 2.2** *Let $(\mathcal{Q}, \mathcal{R})$ be a common partition of multisets of strings $\mathcal{A}$ and $\mathcal{B}$, and let $\delta$ be any duo that appears in $\mathcal{Q}$ and $\mathcal{R}$. Let $\mathcal{Q}'$ denote the partition of $\mathcal{A}$ that is obtained from $\mathcal{Q}$ by cutting all occurrences of the duo $\delta$, and let $\mathcal{R}'$ denote the partition of $\mathcal{B}$ that is obtained from $\mathcal{R}$ by cutting all occurrences of the duo $\delta$. Then, $(\mathcal{Q}', \mathcal{R}')$ is a common partition of $\mathcal{A}$ and $\mathcal{B}$.*

*Proof:* Since $\mathcal{Q}$ is a permutation of $\mathcal{R}$, every block $P$ from $\mathcal{Q}$ that contains $\delta$ appears also in $\mathcal{R}$, and vice versa. Thus, if we cut all occurrences of $\delta$ in $\mathcal{Q}$ and $\mathcal{R}$, the resulting new partitions $\mathcal{Q}'$ and $\mathcal{R}'$ will be again permutations of each other. □

Let $\pi = (\mathcal{P}, \mathcal{Q})$ be a minimum common partition of $A$ and $B$, $m$ be its size and let $\Delta$ be the set of all boundary duos of blocks in $\mathcal{P}$ and in $\mathcal{Q}$. We are going to iteratively construct common partitions $\pi_i$ of $A$ and $B$ that will help us to estimate the size of the common partition found by Refined Greedy. We define $\pi_1$ as the common partition derived from $\pi$ by cutting *all* occurrences of all duos in $\Delta$ (the fact that $\pi_1$ is a partition follows from Observation 2.2). For $k$-MCSP instances, the number of blocks increases at most $k$ times. The breaks in $\pi_1$ are called *initial* breaks. Let $S_i$ denote the substring that Refined Greedy used in iteration $i$ and let $\Phi_i$ be the set of boundary duos of $S_i^A$ and $S_i^B$. For iteration $i \geq 1$ of Refined Greedy, we define $\pi_{i+1}$ as the common partition derived from $\pi_i$ by cutting all occurrences of all duos in $\Phi_i$.

We are going to compare the blocks used by Refined Greedy with the blocks in $\pi_i$. For ease of reference, we denote the sets $\mathcal{A}$ and $\mathcal{B}$ at the beginning of iteration $i$ by $\mathcal{A}_i$ and $\mathcal{B}_i$, and by $s_i$ the first position of $S_i^A$ in $A$, by $t_i$ the last position of $S_i^A$ in $A$, by $s_i'$ the first position of $S_i^B$ in $B$, and by $t_i'$ the last position of $S_i^B$ in $B$.

**Observation 2.3** *For every iteration $i$ and for every $0 \leq l < |S_i|$: the pair $s_i + l, s_i + l + 1$ is an initial break of $A$ if and only if the pair $s_i' + l, s_i' + l + 1$ is an initial break of $B$.*

*Proof:* The observations follow from the definition of $\pi_1$: if one occurrence of a duo is cut in $\pi_1$, then all occurrences of this duo are cut. □

Given a break $l, l+1$ of a partition of $A$, and a substring $S = a_i \dots a_j$ of $A$, we say that the substring $S$ *goes over the break* $l, l+1$ if $i \leq l < j$. Observation 2.3 can be informally stated like this: If the block $S_i^A$ goes over one or more initial breaks, then the block $S_i^B$ goes over the same number of initial breaks, and, moreover, the relative positions of the initial breaks in $S_i^A$ and $S_i^B$ are the same.

Let $\mathcal{A}_i' \subseteq \mathcal{A}_i$ and $\mathcal{B}_i' \subseteq \mathcal{B}_i$ denote the subsets of unmarked strings of $\mathcal{A}_i$ and $\mathcal{B}_i$ ,resp., at the beginning of phase $i$, and let $\pi_i'$ denote the restriction of $\pi_i$ to $\mathcal{A}_i'$ and $\mathcal{B}_i'$. Observation 2.3 implies the following important claim.

**Observation 2.4** *For every $i$, $\pi_i'$ is a common partition of $\mathcal{A}_i'$ and $\mathcal{B}_i'$.*

*Proof:* The proof is by induction. For $i = 1$, nothing is marked, $\mathcal{A}_1' = \{A\}$, $\mathcal{B}_1' = \{B\}$, $\pi_1' = \pi_1$ and the claim is obvious. For $i > 1$, Observations 2.3 and 2.2 imply that the blocks from $\pi_i$ corresponding to the newly marked block $S_{i-1}^A$ are the same as the blocks from $\pi_i$ corresponding to the newly marked block $S_{i-1}^B$. Observing that outside $S_{i-1}^A$ and $S_{i-1}^B$, cuts of the same duos (i.e., duos from $\Phi_{i-1}$) are used to obtain $\pi_i'$ from $\pi_{i-1}'$ and $(\mathcal{A}_i', \mathcal{B}_i')$ from $(\mathcal{A}_{i-1}', \mathcal{B}_{i-1}')$, the proof is completed. □

**Lemma 2.5** *For every $i$,*

- *the block $S_i = a_{s_i} \dots a_{t_i}$ is an entire block in $\mathcal{A}_i'$ and $\mathcal{B}_i'$, or*

- *$S_i$ goes over an initial break or*
- *$s_i - 1, s_i$ is an initial break or $s_i = 1$, and $t_i, t_i + 1$ is an initial break or $t_i = n$.*

*Proof:* The lemma follows from Observation 2.4 and from the greedy nature of Refined Greedy: for every common substring $S$ of $\mathcal{A}'_i$ and $\mathcal{B}'_i$ not satisfying any of the conditions in the lemma, there exists another common longer substring $S'$ of $\mathcal{A}'_i$ and $\mathcal{B}'_i$ such that $S$ is a proper substring of $S'$. □

We are ready to finish the proof of Theorem 2.1. In every iteration, the number of duos in $\mathcal{A}$ that Refined Greedy cuts, is at most $2k$. If Refined Greedy chooses for $S$ an entire block of $\mathcal{A}'_i$, then there are no new cuts introduced in this iteration. If Refined Greedy chooses for $S$ a string that is not an entire block of $\mathcal{A}'_i$, then, by Lemma 2.5, $S$ either goes over an initial break or (roughly) $S$ starts and ends at an initial break. In the former case, we charge all cuts done by Refined Greedy in this iteration to this initial break; in the later case, we charge half of the new cuts to each of these two new breaks (in the special case that $s_i = 1$ or $t_i = n$, we charge all new cuts to the only initial break). In this way each cut done by Refined Greedy is charged to one initial break, and the total number of breaks charged to one initial break is not more than $2 \cdot k$. Since there are at most $k \cdot (m-1)$ initial breaks, there are at most $2 \cdot k^2 \cdot (m-1)$ breaks in the final partition found by Refined Greedy. The total number of blocks used by Refined Greedy is at most $2 \cdot k^2 \cdot (m-1) + 1 = 2 \cdot k^2 \cdot m$.

For signed $k$-MCSP and $k$-RMCSP we only need to adjust the proof to reflect the thing that now a substring $S$ from $A$ can be matched with a substring $R$ from $B$ even if $S \neq R$ but $S = -R$. Thus, in Observation 2.2 we cut not only all occurrences of duo $\delta$ but also all occurrences of duo $-\delta$. To get the common partition $\pi_1$ from $\pi$, for each $\delta \in \Delta$ we cut all occurrences of $\delta$ as well as all occurrences of $-\delta$; for signed $k$-MCSP the number of breaks in $\pi_1$ increases again at most $k$ times, for $k$-RMCSP it increases at most $2k - 1$ times. In Observation 2.3, we distinguish whether $S_i^A = S_i^B$ or $S_i^A = -S_i^B$. In the later case, we count the relative positions of the initial breaks in $S_i^B$ backwards (i.e., the claim is: $s_i + l, s_i + l + 1$ is an initial break of $A$ if and only if the pair $t'_i - l - 1, t'_i - l$ is an initial break of $B$); the former case is as before. For signed $k$-MCSP, the number of duos cut in $\mathcal{A}$ in one iteration is at most $2k$, for $k$-RMCSP it is at most $2(2k-1)$. □

We note that the same approximation ratio holds even with respect to the number of breaks in common partitions (not only with respect to the number of blocks). Considering the relation between signed MCSP and signed SBR, and between RMCSP and unsigned SBR, we get the following theorem.

**Theorem 2.6** *There exists a polynomial time $4k^2$-approximation algorithm for signed $k$-SBR, and $8(2k-1)^2$-approximation algorithm for unsigned $k$-SBR.*

Concerning the running time of Refined Greedy, observe that just finding the longest common substring of $\mathcal{A}$ and $\mathcal{B}$ in linear time requires an involved algorithm [13, 16], and Refined Greedy looks for the longest common substring in every iteration.

## 2.2 Educated Greedy: $O(k^2)$-approximation in time $O(k \cdot n)$

In the previous analysis we never used the fact that $S$ was the *longest* common substring; we only used that it was not possible to extend $S^A$ and still have a matching substring in $B$ (proof of Observation 2.5). Based on this observation, here we present more efficient implementation of the algorithm. As in the case of Refined Greedy, we describe Educated Greedy in detail for

unsigned $k$-MCSP; the necessary modifications for signed $k$-MCSP and $k$-RMCSP are the same as before.

**Algorithm** EDUCATED GREEDY
**Input:** two related strings $A = a_1 \dots a_n$ and $B = b_1 \dots b_n$
$\mathcal{A} \leftarrow (A)$, $\mathcal{B} \leftarrow (B)$
$i = 1$
**while** $i \leq n$ **do**
$S \leftarrow$ longest common substring of $\mathcal{A}$, $\mathcal{B}$ that starts in $A$ on position $i$ and does not overlap previously marked blocks
cut the boundary duos of $S^A$ in $\mathcal{A}$ and the boundary duos of $S^B$ in $\mathcal{B}$
mark $S^A$ in $\mathcal{A}$ and $S^B$ in $\mathcal{B}$
cut in $\mathcal{A}$ and $\mathcal{B}$ *all* unmarked occurrences of duos $\delta \in \Phi$, where $\Phi$ is the set of boundary duos of $S^A$ and $S^B$
$i \leftarrow i + |S|$
**Output:** $(\mathcal{A}, \mathcal{B})$

**Theorem 2.7** *There exist an $O(k^2)$-approximation algorithms for unsigned and signed $k$-MCSP, $k$-RMCSP and $k$-SBR running in time $O(k \cdot n)$.*

*Proof:* The proof of Lemma 2.5 is the only place in the proof of Theorem 2.1 that refers to the choice of the common substring $S$ used by REFINED GREEDY. However, as mentioned above, the proof only needs the fact that $S$ cannot be extended on either side. Thus, Lemma 2.5 holds also for the choices of EDUCATED GREEDY and the $O(k^2)$ approximation ratio follows by the same reasoning as for REFINED GREEDY.

Concerning the running time, EDUCATED GREEDY goes once through $A$ from left to right, and in every iteration, there are at most $k$ possibilities (resp., $2k$ for $k$-RMCSP) where to look for the common substring $S_j$. EDUCATED GREEDY spends at most $k \cdot |S_j|$ (resp., $2k \cdot |S_j|$) steps in iteration $j$ and advances by $|S_j|$ positions to the right in $A$. Thus, the common partition is computed in time $O(k \cdot n)$ and the proof is completed. □

# 3 Conclusion

We presented a simple, $O(k^2)$-approximation algorithms for $k$-MCSP and $k$-SBR, running in time $O(k \cdot n)$. For instances with $3 < k \leq O(\sqrt{\log n \log^* n})$, this is the best approximation ratio and, moreover, EDUCATED GREEDY is faster than the previous best approximation algorithm.

We conclude with a few challenging open problems. Is it possible to implement REFINED GREEDY in linear time? Is there a simple $O(k)$-approximation algorithm for $k$-SBR? What is the best possible approximation ratio for the general SBR? Is it possible to get below the $O(\log n \log^* n)$ upper bound? Is it NP-hard to approximate better than within $\Omega(\log n)$?


## Acknowledgment

We would like to thank Jiří Sgall for suggestion to implement REFINED GREEDY more efficiently.

## References


[1] A. Bergeron, J. Mixtacki, and J. Stoye. Reversal distance without hurdles and fortresses. In *Proceedings of 15th Annual Combinatorial Pattern Matching Symposium (CPM)*, volume 3109 of Lecture Notes in Computer Science, pages 388–399. Springer-Verlag, 2004.

[2] P. Berman, S. Hannenhalli, and M. Karpinski. 1.375-approximation algorithm for sorting by reversals. In *Proceedings of the 10th Annual European Symposium on Algorithms (ESA)*, volume 2461 of *Lecture Notes in Computer Science*, pages 200–210, 2002.

[3] P. Berman and M. Karpinski. On some tighter inapproximability results. In *Proceedings of the 26th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 1644 of *Lecture Notes in Computer Science*, pages 200–209, 1999.

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

[5] X. Chen, J. Zheng, Z. Fu, P. Nan, Y. Zhong, S. Lonardi, and T. Jiang. Assignment of orthologous genes via genome rearrangement. Submitted, 2004.

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

[7] M. Chrobak, P. Kolman, and J. Sgall. The greedy algorithm for the minimum common string partition problem. In *Proceedings of the 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, volume 3122 of *Lecture Notes in Computer Science*, pages 84–95, 2004.

[8] 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.

[9] N. El-Mabrouk. Reconstructing an ancestral genome using minimum segments duplications and reversals. *Journal of Computer and System Sciences*, 65(3):442–464, 2002.

[10] F. Ergun, S. Muthukrishnan, and S. C. Sahinalp. Comparing sequences with segment rearrangements. In *Proceedings of the 23rd Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 2914 of *Lecture Notes in Computer Science*, pages 183–194, 2003.

[11] A. Goldstein, P. Kolman, and J. Zheng. Minimum Common String Partition Problem: Hardness and Approximations. In *Proceedings of the 15th International Symposium on Algorithms and Computation (ISAAC)*, Lecture Notes in Computer Science, 2004.

[12] 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, Jan. 1999.

[13] J. H. Morris, Jr and V. R. Pratt. A linear pattern-matching algorithm. Report 40, University of California, Berkeley, 1970.

[14] D. Sankoff and N. El-Mabrouk. Genome rearrangement. In T. Jiang, Y. Xu, and M. Q. Zhang, editors, *Current Topics in Computational Molecular Biology*. The MIT Press, 2002.

[15] D. Shapira and J. A. Storer. Edit distance with move operations. In *Proceedings of the 13th Symposium on Combinatorial Pattern Matching (CPM)*, volume 2373 of *Lecture Notes in Computer Science*, pages 85–98, 2002.

[16] P. Weiner. Linear pattern matching algorithm. In *Proceedings of the 14th Annual IEEE Symposium on Switching and Automata Theory*, pages 1–11, 1973.