Linear Time Approximation Algorithm for Sorting by Reversals with Bounded Number of Duplicates

Petr Kolman [*]

November 1, 2004

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.

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

Key words. Approximation algorithms, String comparison, Sorting by reversal, Computational biology.

1 Introduction

For a string $A = a_1 \ldots a_n$, a *reversal* $\rho(i, j)$, $1 \leq i < j \leq n$ transforms the string A into a string $A' = a_1 \ldots a_{i-1} a_j a_{j-1} \ldots a_i a_{j+1} \ldots a_n$, that is, the reversal $\rho(i, j)$ reverses the order of symbols in the substring $a_i \ldots a_j$ 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 $\mathsf{SBR}(A, B) \leq 2$. 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, $\mathsf{SBR}(A', B') \leq 2$.

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 $P = a_1 \ldots a_n$, 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 $A = a_1 a_2 \ldots a_n$ and $B = b_1 b_2 \ldots b_n$, 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 $\mathcal{P} = (P_1, P_2, \ldots, P_m)$ 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 $\mathcal{P} = (P_1, P_2, \ldots, P_m)$, 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 $\mathcal{A} = (A_1, \ldots, A_m)$ and $\mathcal{B} = (B_1, \ldots, B_{m'})$, 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 [3, 10, 5]. 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 [12]. On the other hand, a weaker assumption that a genome contains at most $k = O(1)$ copies of each gene is often warranted (cf. [8]). That is why k-SBR is of practical 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 [10] but is NP-hard [3] and even MAX-SNP hard [2] for strings without signs; the best known approximation ratio for the unsigned 1-SBR is 1.375 by an algorithm of Berman et al. [1]. A recent result of Chen et al. [4] 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 [4, 6, 9]. The best approximation ratio for 2-SBR is 2.2074 and the algorithm relies on semidefinite programming [9]; the algorithm for 3-SBR runs in linear time and has approximation ratio 8 [9]. 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 [7].

Instead of bounding the number of duplicates, there is another way to restrict the general problem of 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 [5].

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 σ 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 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 relation $=$ is used; in *signed* MCSP, the input consists of two signed strings and 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 the relation $\cong$.

The signed MCSP problem was introduced by Chen et al. [4] 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 [9]. 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 [6, 9], $O(\log n \log^* n)$ approximation ratio for the general MCSP [7].

Chrobak et al. [6] analyzed the behavior of a natural greedy heuristic for

MCSP (iteratively, at each step extract a longest common substring from the input strings). 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, MOVE a substring or REVERSE a substring) 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 REVERSE and the input strings are related. For any two related strings A and B, MCSP(A, B) differs by a constant multiplicative factor from the edit distance of A and B with only MOVE operations, and the edit distance using only the MOVE operations differs also by a constant multiplicative factor from the edit distance with operations {INSERT, DELETE, MOVE} [13]. For the later problem, Cormode and Muthukrishnan [7] 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 in earlier in this subsection.

2 Algorithms

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

In the previous section, we briefly described the 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.

To describe REFINED GREEDY, a few more terms are needed. A *duo* is 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.

In k-MCSP instances, each letter in the string A must be matched with exactly one of the (at most) k occurrences of the same letter in B. 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 (cf. [6]). REFINED GREEDY avoids 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 prevent the algorithm from propagating mistakes.

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: $(\mathcal{A}, \mathcal{B})$
```

For example, if $A = cdabcdabceab$, $B = abceabcdabcd$, then algorithm REFINED GREEDY first marks substring $S_1 = abcdabc$ and cuts all unmarked occurrences of duos from $\Phi = \{da, ce, ea, cd\}$. In the second iteration, REFINED GREEDY looks for the longest unmarked substring in partitions $\mathcal{A} = (c, d, \overline{abcdabc}, e, ab)$ and $\mathcal{B} = (abc, e, \overline{abcdabc}, d)$ and marks substring $S_2 = ab$. In the next iterations, the unmarked substrings are only the single-letter characters and the REFINED GREEDY marks subsequently c,d and e. The resulting common partition that REFINED GREEDY outputs is

$$\langle (c, d, abcdabc, e, ab), (ab, c, e, abcdabc, d) \rangle \ .$$

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 $-\delta$.

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.

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

Proof: Since $\mathcal{Q}$ is a permutation of $\mathcal{R}$, every block P from $\mathcal{Q}$ that contains δ appears also in $\mathcal{R}$, and vice versa. Thus, if we cut all occurrences of δ 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 Δ be the set of all boundary duos of blocks in $\mathcal{P}$ and in $\mathcal{Q}$. We are going to iteratively construct common partitions π_i of A and B that will help us to estimate the size of the common partition found by Refined Greedy. We define π_0 as the common partition derived from π by cutting *all* occurrences of all duos in Δ (the fact that π_0 is a partition follows from Observation 2.2). For k-MCSP instances, the number of blocks increases at most k times. The breaks in π_0 are called *initial* breaks. Let S_i denote the substring that Refined Greedy used in iteration i and let Φ_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 π_i as the common partition derived from π_{i-1} by cutting all occurrences of all duos in Φ_i.

We are going to compare the blocks used by Refined Greedy with the blocks in π_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 π_0: if one occurrence of a duo is cut in π_0, then all occurrences of this duo are cut. □

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 A_{i+1} and B_{i+1} denote the strings A and B after removing (that is, after shrinking to length zero) from them all substrings that are marked in $\mathcal{A}_i$ and $\mathcal{B}_i$, resp., and let $\pi_i' = (\mathcal{Q}, \mathcal{R})$ denote the restriction of π_i to only unmarked symbols of A and B. Observation 2.3 implies the following important claim.

Observation 2.4 *π_i' is a common partition of A_{i+1} and B_{i+1}.*

Proof: By Observation 2.2, π_i is a common partition of $\mathcal{A}_{i+1}$ and $\mathcal{B}_{i+1}$. By Observation 2.3, $\mathcal{Q}$ is a permutation of $\mathcal{R}$, therefore $\pi_i' = (\mathcal{Q}, \mathcal{R})$ is a common partition of A_{i+1} and B_{i+1}. □

Given a break $l, l+1$ of a partition of A, and a substring $S = a_i \ldots a_j$ of A, we say that the substring S *goes over the break* $l, l+1$ if $i \leq l < j$.

Lemma 2.5 *If the block $S_i = a_{s_i} \ldots a_{t_i}$ is not an entire block in $\mathcal{Q}$ (that is, in π_i'), then there exists a position h, $s_i - 1 \leq h \leq t_i$, such that $h, h+1$ is an initial break in A.*

Proof: The lemma follows from the fact that π_i' is a common partition of A_{i+1} and B_{i+1} (by Observation 2.4) and from the greedy nature of the Refined Greedy: every common substring S of $\mathcal{A}_i$ and $\mathcal{B}_i$ that is not an entire block of $\mathcal{Q}$ with the property that it does not start and end at an initial break and it does not go over an initial break, can be extended in a longer common substring. □

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 from $\mathcal{Q}$ of π_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 in $\mathcal{Q}$, then, by Observation 2.5, S either goes over an initial break and it gets marked, or S starts and ends at an initial break and they both get partially marked. We charge all cuts done by Refined Greedy in this iteration to the initial break that gets marked, resp. to the two breaks that get partially marked, half to each. Clearly, 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 $2 \cdot k^2 \cdot (m-1)+1 = O(k^2 \cdot m)$.

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 δ but also all occurrences of duo $-\delta$. To get the common partition π_0 from π, for each $\delta \in \Delta$ we cut all occurrences of δ as well as all occurrences of $-\delta$; for unsigned k-MCSP the number of breaks in π_0 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 $\mathcal{A}$ 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 [11, 14], and the 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 S was always a *maximal* common substring. 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 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 algorithm 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. $\square$

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 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 bellow 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 his suggestion for more efficient implementation of Refined Greedy.

References

[1] 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.

[2] 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.

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

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

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

[6] 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.

[7] 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.

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

[9] 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.

[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, Jan. 1999.

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

[12] 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.

[13] 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.

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