Minimum Common String Partition Problem: Hardness and Approximations*

Avraham Goldstein
Department of Mathematics
Stony Brook University
Stony Brook, NY, U.S.A.
avi_goldstein@netzero.net

Petr Kolman†
Department of Applied Mathematics
Faculty of Mathematics and Physics
Charles University, Prague, Czech Republic
kolman@kam.mff.cuni.cz

Jie Zheng‡
Department of Computer Science
University of California
Riverside, CA, U.S.A.
zjie@cs.ucr.edu

Submitted: Aug 19, 2004; Accepted: Aug 28, 2005; Published: Sep 29, 2005
Mathematics Subject Classifications: 68W25, 68Q25

Abstract

String comparison is a fundamental problem in computer science, with applications in areas such as computational biology, text processing and compression. In this paper we address the minimum common string partition problem, a string comparison problem with tight connection to the problem of sorting by reversals with duplicates, a key problem in genome rearrangement.

A *partition* of a string A is a sequence $\mathcal{P} = (P_1, P_2, \ldots, P_m)$ of strings, called the *blocks*, whose concatenation is equal to A. Given a partition $\mathcal{P}$ of a string A and a partition $\mathcal{Q}$ of a string B, we say that the pair $\langle \mathcal{P}, \mathcal{Q} \rangle$ is a *common partition* of A and B if $\mathcal{Q}$ is a permutation of $\mathcal{P}$. The *minimum common string partition* problem (MCSP) is to find a common partition of two strings A and B with the minimum number of blocks. The restricted version of MCSP where each letter occurs at most k times in each input string, is denoted by k-MCSP.

In this paper, we show that 2-MCSP (and therefore MCSP) is NP-hard and, moreover, even APX-hard. We describe a 1.1037-approximation for 2-MCSP and a linear time 4-approximation algorithm for 3-MCSP. We are not aware of any better approximations.

*Preliminary version of this work was presented at the 15^{th} International Symposium on Algorithms and Computation [9].

†Research done while visiting University of California at Riverside. Partially supported by project 1M0021620808 of MŠMT ČR, and NSF grants CCR-0208856 and ACI-0085910.

‡Supported by NSF grant DBI-0321756.

1 Introduction

String comparison is a fundamental problem in computer science, with applications in areas such as computational biology, text processing and compression. Typically, a set of string operations is given (e.g., delete, insert and 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. Edit distance or permutation sorting by reversals are two well known examples. In this paper we address, motivated mainly by genome rearrangement applications, the minimum common string partition problem (MCSP). Though MCSP takes a static approach to string comparison, it has tight connection to the problem of sorting by reversals with duplicates, a key problem in genome rearrangement.

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 $\mathcal{P}$. 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 if $\mathcal{Q}$ is a permutation of $\mathcal{P}$. The *minimum common string partition* problem is to find a common partition of A, B with the minimum number of blocks. The restricted version of MCSP where each letter occurs at most k times in each input string, is denoted by k-MCSP. We denote by $\#blocks(\pi)$ the number of blocks in a common partition π. We say that two strings A and B are *related* if every letter appears the same number of times in A and B. Clearly, a necessary and sufficient condition for two strings to have a common partition is that they are related.

The *signed* minimum common string partition problem (SMCSP) is a variant of MCSP in which each letter of the two input strings is given a "+" or "−" sign (in genome rearrangement problems, the letters represent different genes on a chromosome and the signs represent orientation of the genes). For a string P with signs, let $-P$ denote the reverse of P, with each sign flipped. A common partition of two signed strings A and B is the pair $\pi = \langle \mathcal{P}, \mathcal{Q} \rangle$ of a partition $\mathcal{P} = (P_1, P_2, \ldots, P_m)$ of A and a partition $\mathcal{Q} = (Q_1, Q_2, \ldots, Q_m)$ of B together with a permutation σ on $[m]$ such that for each $i \in [m]$, either $P_i = Q_{\sigma(i)}$, or $P_i = -Q_{\sigma(i)}$.

New results. In this paper, we show that 2-MCSP (and therefore MCSP) is NP-hard and, moreover, even APX-hard. We also describe a 1.1037-approximation for 2-MCSP and a linear time 4-approximation algorithm for 3-MCSP. All of our results apply also to signed MCSP. We are not aware of any better approximations.

Related work. 1-MCSP coincides with the breakpoint distance problem of two permutations [14] which is to count the number of ordered pairs of symbols that are adjacent in the first string but not in the other; this problem is obviously solvable in polynomial time. Similarly as the breakpoint distance problem does, most of the rearrangement literature works with the assumption that a genome contains only one copy of each gene. Under this assumption, a lot of attention was given to the problem of sorting by reversals. *Reversal* is an operation that reverses a specified substring of a given string; in the case of signed strings, it also flips the sign of each letter in the reversed substring. In the problem of *sorting by reversals*, the task is to determine the minimum number of reversals that transform a given string A into a given string B. The problem is solvable in polynomial time for signed strings containing only one copy of each symbol [10] but is NP-hard for unsigned strings [3]. The assumption about uniqueness of each gene is unwarranted for genomes with multi-gene families such as the human genome [12]. Chen et al. [4] studied a generalization of the problem, the problem of *signed reversal distance with duplicates* (SRDD); according to them, SRDD is NP-hard even if there are at most two copies of each gene. They also introduced the signed minimum common partition problem as a tool for dealing with SRDD. Chen et al. observe that for any two related signed strings A and B, the size of a minimum common partition and the minimum number of reversal operations needed to transform A to B, are within a multiplicative factor 2 of each other. (In the case of unsigned strings, no similar relation holds: the reversal distance of $A = 1234\ldots n$ and $B = n\ldots 4321$ is 1 while the size of minimum common partition is $n-1$.) They also give a 1.5-approximation algorithm for 2-MCSP. They reduce the problem to a vertex cover problem (on 6-claw-free graphs) and show that an α-approximation for minimum vertex cover yields an α-approximation for 2-MCSP. Christie and Irving [5] consider the problem of (unsigned) reversal distance with duplicates (RDD) and prove that it is NP-hard even for strings over binary alphabet.

Chrobak et al. [6] analyze a natural heuristic for MCSP, the greedy[1] algorithm: iteratively, at each step extract a longest common substring from the input strings. They show that for 2-MCSP, the approximation ratio is exactly 3, for 4-MCSP the approximation ratio is $\Omega(\log n)$; for the general MCSP, the approximation ratio is between $\Omega(n^{0.43})$ and $O(n^{0.67})$. The same bounds apply for SMCSP. Kolman [11] described a simple modification of the greedy algorithm; the approximation ratio of the modification is $O(k^2)$ for k-MCSP and it runs in time $O(k \cdot n)$. The same bounds hold also for k-SMCSP and k-SRDD.

Closely related is the problem of edit distance with moves in which the allowed string operations are the following: insert a character, delete a character, move a substring. Cormode and Muthukrishnan [7] describe an $O(\log n \log^* n)$-approximation algorithm for this problem. Shapira and Storer [13] observed that restriction to move-a-substring operations only (instead of allowing all three operations listed above) does not affect the edit-distance of two strings by more than a constant multiplicative factor. Since the size of a minimum common partition of two strings and their distance with respect to move-a-substring operations differ only by a constant multiplicative factor, the algorithm of Cormode and Muthukrishnan yields an $O(\log n \log^* n)$-approximation for MCSP.

1.1 Preliminaries

Throughout the paper, we assume that the two strings A, B given as input to MCSP are related. This is a necessary and sufficient condition for the existence of a common

[1]Shapira and Storer [13] also analyzed the greedy algorithm and claimed an $O(\log n)$ bound on its approximation ratio; unfortunately, the analysis was flawed, it applies only to a special subclass of MCSP problems.

partition.

Given a string $A = a_1 \ldots a_n$, for the sake of simplicity we will use the symbol a_i to denote two different things. First, a_i may denote the specific occurrence of the letter a_i in the string A, namely the occurrence on position i. Alternatively, a_i may denote just the letter itself, without any relation to the string A. Which alternative we mean will be clear from context.

Common partitions as mappings. Given two strings $A = a_1 \ldots a_n$ and $B = b_1 \ldots b_n$ of length n, a common partition π of A and B can be naturally interpreted as a bijective mapping from A to B (that is, if $P_1, \ldots, P_m$ is the partition of A and $Q_1, \ldots, Q_m$ is the partition of B in π, then for each $j \in [m]$, the letters from P_j are mapped from left to right to the corresponding $Q_{j'}$), and this in turn as a permutation on $[n]$. With this understanding in mind, we say that a pair of consecutive positions $i, i+1 \in [n]$ is a *break* of π in A if $\pi(i+1) \neq \pi(i) + 1$. In other words, a break is a pair of letters that are consecutive in A but are mapped by π to letters that are not consecutive in B. The number of breaks in π will be denoted by $\#breaks(\pi)$.

Clearly, not every permutation on $[n]$ corresponds to a common partition of A and B. We say that a permutation ρ on $[n]$ *preserves letters* of A and B, if $a_i = b_{\rho(i)}$, for all $i \in [n]$. Then, every letter-preserving mapping ρ can be interpreted as a common partition ρ, and $\#blocks(\rho) = \#breaks(\rho) + 1$. On the other hand, for a common partition $\pi = \langle \mathcal{P}, \mathcal{Q} \rangle$ interpreted as a permutation, $\#blocks(\pi) \geq \#breaks(\pi) + 1$ (the inequality is due to possible unnecessary breaks in π). Thus, the MCSP problem is to find a permutation π on $[n]$ that preserves letters of A and B and has the minimum number of breaks. An alternative formulation is that the goal is to find a letter-preserving permutation that maps the maximum number of pairs of consecutive letters in A to pairs of consecutive letters in B.

Common partitions and independent sets. Let Σ denote the set of all letters that occur in A. A *duo* is an ordered pair of letters $xy \in \Sigma^2$ that occur consecutively in A or B (that is, there exists an i such that $x = a_i$ and $y = a_{i+1}$, or $x = b_i$ and $y = b_{i+1}$). A *specific* duo is an occurrence of a duo in A or B. The difference is that a duo is just a pair of letters whereas a specific duo is a pair of letters together with its position. A *match* is a pair $(a_i a_{i+1}, b_j b_{j+1})$ of specific duos, one from A and the other one from B, such that $a_i = b_j$ and $a_{i+1} = b_{j+1}$. Two matches $(a_i a_{i+1}, b_j b_{j+1})$ and $(a_k a_{k+1}, b_l b_{l+1})$, $i \leq k$, are *in conflict* if either $i = k$ and $j \neq l$, or $i + 1 = k$ and $j + 1 \neq l$, or $i + 1 < k$ and $\{j, j+1\} \cap \{l, l+1\} \neq \emptyset$. Informally, two matches are in conflict if they cannot be realized at the same time.

We construct a *conflict graph* $G = (V, E)$ of A and B as follows. The set of nodes V consists of all matches of A and B and the set of edges E consists of all pairs of matches that are in conflict. Figure 1 shows an example of a conflict graph. The number of vertices in G can be much higher than the length of the strings A and B (and is trivially bounded by n^2).

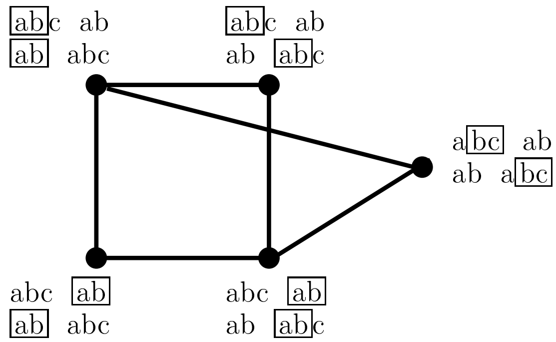


Figure 1: Conflict graph for MCSP instance $A = abcab$ and $B = ababc$.

Lemma 1.1 *For $A = a_1 \dots a_n$ and $B = b_1 \dots b_n$, let $MIS(G)$ denote the size of the maximum independent set of the conflict graph G of A and B and m denote the number of blocks in a minimum common partition of A and B. Then, $n - MIS(G) = m$.*

Proof: Given an optimal solution for MCSP, let S be the set of all matches that are used in this solution. Clearly, S is an independent set in G and $|S| = n - 1 - (m - 1)$.

Conversely, given a maximum independent set S, we cut the string A between a_i and a_{i+1} for every specific duo $a_i a_{i+1}$ that does not appear in any match in S, and similarly for B. In this way, $n - 1 - |S|$ duos are cut in A and also in B, resulting in $n - |S|$ blocks of A and $n - |S|$ blocks of B. Clearly, the blocks from A can be matched with the blocks from B, and therefore $m \leq n - |S|$. $\square$

Maximum independent set is an NP-hard problem, yet, two approximation algorithms for MCSP described in this paper make use of this reduction.

MCSP for multisets of strings. For the proofs in later sections we need a slight generalization of the MCSP. Instead of two strings A, B, the input consists of two multisets $\mathcal{A}, \mathcal{B}$ of strings. Similarly as before, a partition of the multiset $\mathcal{A} = \{A_1, \dots, A_l\}$ is a sequence of strings

$$A_{1,1}, \dots, A_{1,k_1}, A_{2,1}, \dots, A_{2,k_2}, \dots, A_{l,1}, \dots, A_{l,k_l},$$

such that $A_i = A_{i,1} \dots, A_{i,k_i}$ for $i \in [l]$. For two multisets of strings, the common partition, the minimum common partition and the related-relation are defined similarly as for pairs of strings.

Let $\mathcal{A} = \{A_1, \dots, A_l\}$ and $\mathcal{B} = \{B_1, \dots, B_h\}$ with $h \leq l$, be two related multisets of strings, and let $x_1, y_1, \dots, x_{l-1}, y_{l-1}$ be $2l-2$ different letters that do not appear in $\mathcal{A}$ and $\mathcal{B}$. Considering two strings

$$\begin{aligned} A &= A_1 x_1 y_1 A_2 x_2 y_2 A_3 \dots x_{l-1} y_{l-1} A_l \ , \\ B &= B_1 y_1 x_1 B_2 y_2 x_2 B_3 \dots y_{h-1} x_{h-1} B_h y_h x_h \dots y_{l-1} x_{l-1} \ , \end{aligned} \tag{1}$$

it is easy to see that an optimal solution for the classical MCSP instance A, B yields an optimal solution for the instance $\mathcal{A}, \mathcal{B}$ of the multiset version, and vice versa. In particular, if m' denotes the size of a MCSP of the two multisets of strings $\mathcal{A}$ and $\mathcal{B}$, and m denotes the size of a MCSP of the two strings A and B defined as above, then

$$m = m' + 2(l-1) . \tag{2}$$

Thus, if one of the variants of the problems is NP-hard, so is the other.

2 Hardness of approximation

The main result of this section is the following theorem.

Theorem 2.1 2-MCSP *and* 2-SMCSP *are APX-hard problems.*

We start by proving a weaker result.

Theorem 2.2 2-MCSP *and* 2-SMCSP *are NP-hard problems.*

Proof: Since an instance of MCSP can be interpreted as an instance of SMCSP with all signs positive, and since a solution of SMCSP with all signs positive can be interpreted as a solution of the original MCSP and vice versa, it is sufficient to prove the theorems for MCSP only.

The proof is by reduction from the maximum independent set problem on cubic graphs (3-MIS) [8]. Given a cubic graph $G = (V, E)$ as an input for 3-MIS, for each vertex $v \in V$ we create a small instance I_v of 2-MCSP. Then we process the edges of G one after another, and, for each edge $(u, v) \in E$, we locally modify the two small instances I_u, I_v. The final instance of 2-MCSP, denoted by I_G, is the union of all the small (modified) instances I_v. We will show that a minimum common partition of I_G yields easily a maximum independent set in G.

The small instance $I_u = (X_u, Y_u)$ for a vertex $u \in V$ is defined as follows (cf. Figure 2):

$$\begin{aligned} X_u &= \{d_u,\ a_u b_u,\ c_u d_u e_u,\ b_u e_u f_u g_u,\ f_u h_u k_u,\ g_u l_u,\ h_u\} \\ Y_u &= \{b_u,\ c_u d_u,\ a_u b_u e_u,\ d_u e_u f_u h_u,\ f_u g_u l_u,\ h_u k_u,\ g_u\} \end{aligned} \tag{3}$$

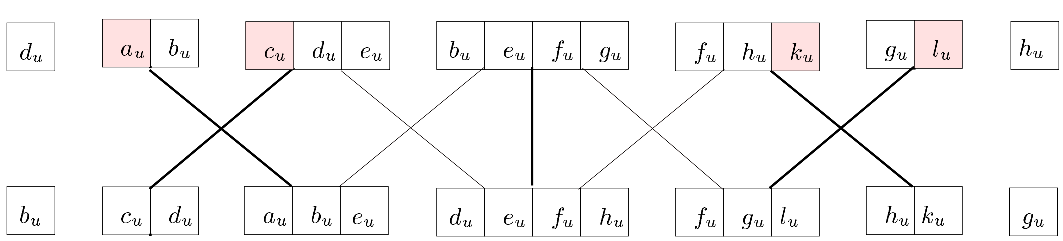


Figure 2: An instance I_u: the lines represent all matches, with the bold lines corresponding to the matches in the minimum common partition O_u.

where all the letters in the set $\cup_{u\in V}\{a_u, b_u, \ldots, l_v\}$ are distinct. It is easy to check that I_u has a unique minimum common partition, denoted by O_u, namely:

$$\begin{aligned} O_u \;=\; & \langle (d_u,\ a_ub_u,\ c_ud_u,\ e_u,\ b_u,\ e_uf_u,\ g_u,\ f_u,\ h_uk_u,\ g_ul_u,\ h_u) \\ & \;(b_u,\ c_ud_u,\ a_ub_u,\ e_u,\ d_u,\ e_uf_u,\ h_u,\ f_u,\ g_ul_u,\ h_uk_u,\ g_u)\rangle \end{aligned}$$

We observe that for $X_G = \bigcup_{u\in V} X_u$ and $Y_G = \bigcup_{u\in V} Y_u$, $I_G = (X_G, Y_G)$ is an instance of 2-MCSP, and the superposition of all O_u's is a minimum common partition of I_G. For the sake of simplicity, we will sometimes abuse the notation by writing $I_G = \bigcup_{u\in V} I_u$.

The main idea of the construction is to modify the instances I_u, such that for every edge $(u, v) \in E$, a minimum common partition of $I_G = \bigcup_{u\in V} I_u$ coincides with at most one of the minimum common partitions of I_u and I_v. This property will make it possible to obtain a close correspondence between maximum independent sets in G and minimum common partitions of I_G: if O_v denotes a minimum common partition of (the modified) I_v and O'_v denotes the common partition of (the modified) I_v derived from a given minimum common partition of I_G, then $U = \{u \in V \mid O'_u = O_u\}$ will be a maximum independent set of G. To avoid the need to use different indices, we use I_G to denote $\bigcup_{u\in V} I_u$ after any number of the local modifications; it will always be clear from context to which one are we referring.

For description of the modifications, a few terms will be needed. The letters a_u and c_u in X_u are called *left sockets of* I_u and the letters k_u and l_u in X_u are *right sockets*. We observe that all the four letters a_u, c_u, k_u, l_u appears only once in X_G (and once in Y_G). Given two small instances I_u and I_v and a socket s_u of I_u and a socket s_v of I_v, we say that the two sockets s_u and s_v are *compatible*, if one of them is a left socket and the other one is a right socket. Initially, all sockets are *free*.

For technical reasons, we orient the edges of G in such a way that each vertex has at most two incoming edges and at most two outgoing edges. This can be done as follows: find a maximal set (with respect to inclusion) of edge-disjoint cycles in G, and in each cycle, orient the edges to form a directed cycle. The remaining edges form a forest. For each tree in the forest, choose one of its nodes of degree one to be the root, and orient all edges in the tree away from the root. This orientation will clearly satisfy the desired properties.

We are ready to describe the local modifications. Consider an edge $\overrightarrow{(u,v)} \in E$ and a free right socket s_u of I_u and a free left socket s_v of I_v. That is, $Rs_u \in X_u$ and $s_vS \in X_v$, for some strings R and S. We modify the instances $I_u = (X_u, Y_u)$ and $I_v = (X_v, Y_v)$ as follows

$$\begin{array}{ll} X_u \leftarrow X_u \cup \{Rs_uS\} - \{Rs_u\}\ , & X_v \leftarrow X_v \cup \{s_u\} - \{s_vS\}\ , \\ Y_u \leftarrow Y_u\ , & Y_v \leftarrow Y_v \text{ with } s_v \text{ renamed by } s_u \end{array} \tag{4}$$

(the symbols $\cup$ and $-$ denote multiset operations).

After this operation, we say that the right socket s_u of I_u and the left socket s_v of I_v are *used* (not free). Note that in Y_v, the letter s_v is renamed to s_u. All other sockets of I_u and all other sockets of I_v that were free before the operation remain free. We also note that I_u and I_v are not 2-MCSP instances. However, for every letter, the number of its occurrences is the same in X_G and in Y_G, namely at most two. Thus, I_G is still a 2-MCSP instance.

The complete reduction from a cubic graph $G = (V, E)$ to a 2-MCSP instance is done by performing the local modifications (4) for all edges in G.

Reduction of 3-MIS to 2-MCSP

1. $\forall u \in V$, define I_u by the description (3),
2. $\forall \overrightarrow{(u, v)} \in E$, find a free right socket s_u of I_u and a free left socket s_v of I_v, modify I_u and I_v by the description (4),
3. set $I_G = \bigcup_{u \in V} I_u$.

Since the in-degree and the out-degree of every node is bounded by two, and since every instance I_u has initially two right and two left sockets, there will always be the required free sockets.

It remains to prove that a minimum common partition for the final I_G (that is, when modifications for all edges are done) can be used to find a maximum independent set in G.

Lemma 2.3 *Let G be a cubic graph on N vertices. Then, there exists an independent set I of size h in G if and only if there exists a common partition of I_G of size $12N - h$.*

Proof: Let G_C be the conflict graph of I_G; G_C has $9N$ vertices. Let $O'_u = \{(d_u c_u, d_u c_u), (b_u e_u, b_u e_u), (f_u g_u, f_u g_u), (f_u h_u, f_u h_u)\}$, that is, O'_u is a set consisting of four out of the nine possible matches in the small instance I_u (in Figure 2, these four matches are represented by the thin lines). The crucial observation is that $\bigcup_{u \in V} O'_u$ is an independent set of size $4N$ in the conflict graph G_C.

Given an independent set I of G, construct a common partition of I_G as follows. For $u \in I$, use the five matches from O_u, and for $u \notin I$, use the four matches from O'_u. The resulting solution will use $5h + 4(N - h)$ matches which corresponds to $9N - (5h + 4(N - h)) = 5N - h$ new breaks and $7N + 5N - h = 12N - h$ blocks.

Conversely, given a common partition of I_G of size m, let I consist of all vertices u such that I_u contributes 5 matches (i.e., 11 blocks) to the common partition. Then, $h \geq 12N - m$, and the proof is completed. □

Since the reduction can clearly be done in polynomial time (even in linear), with respect to $|V|$ and $|E|$, the proof of NP-hardness of 2-MCSP is completed. □

Proof: (Theorem 2.1) We use the same construction and only complement calculations of the inapproximability ratio. Given a cubic graph G on N vertices, let m' denote the size of a minimum common partition of the instance $I_G = (X_G, Y_G)$ and let m denote the size of a minimum common partition of the instance (A, B), derived from the multiset instance (X_G, Y_G) by relation (1). We note that each of X_G and Y_G consists of $7N$ strings. By Lemma 2.3 the size of a maximum independent set in G is $12N - m'$ which equals to $26N - 2 - m$ by relation (2) and the above observation about size of X_G and Y_G; thus, an α-approximation algorithm for MCSP on the instance (A, B) can be used to derive an independent set in G of size at least $26N - 2 - \alpha \cdot m$.

Berman and Karpinski [2] proved that it is NP-hard to approximate 3-MIS within $\frac{140}{139} - \epsilon$, for every $\epsilon > 0$. Thus, unless P=NP, for every $\epsilon > 0$, the approximation ratio α of any algorithm for MCSP must satisfy

$$\frac{26N - 2 - m}{26N - 2 - \alpha \cdot m} \geq \frac{140}{139} - \epsilon \ .$$

Solving for α yields, for every $\epsilon' > 0$,

$$\alpha \geq \frac{26N - 2 + 139m}{140m} - \epsilon' = 1 + \frac{26N - 2 - m}{140m} - \epsilon' \ .$$

Using the fact that a maximum independent set in any cubic graph on N vertices has always size at least $N/4$, we have $m \leq 26N - 2 - N/4$ and we conclude that it is NP-hard to approximate MCSP within $1 + \frac{1}{103 \cdot 140} - \epsilon$, for every $\epsilon > 0$. $\square$

Remark: To prove that only SMCSP is APX-hard, it is possible to start with smaller instances I_u and thus get the constant larger.

3 Algorithms

3.1 2-MCSP reduces to MIN 2-SAT

In this section we will see how to solve 2-MCSP using algorithms for MIN 2-SAT. We start by recalling the definition of MIN 2-SAT problem. In MIN 2-SAT we are given a boolean formula in conjunctive normal form such that each clause consists of at most two literals, and we seek seek an assignment of boolean values to the variables that minimizes the number of satisfied clauses. Avidor and Zwick [1] proved that unless P=NP, the problem cannot be approximated within $15/14 - \epsilon$, for any $\epsilon > 0$, and they also gave a 1.1037-approximation algorithm which is the best approximation algorithm for the problem we are aware of. The main result of this section is stated in the following theorem.

Theorem 3.1 *An α-approximation algorithm for* MIN 2-SAT *yields α-approximations for both* 2-MCSP *and* 2-SMCSP.

Corollary 3.2 *There exist polynomial* 1.1037*-approximation algorithms for* 2-MCSP *and* 2-SMCSP *problems.*

Proof:(Theorem 3.1) There are only minor differences between the reductions for signed and unsigned versions of the problem. We describe in detail the reduction for 2-MCSP and then briefly point out the differences for 2-SMCSP.

Let A and B be two related strings. We start the proof with two assumptions that will simplify the presentation:

(1) no duo appears at the same time twice in A and twice in B, and that

(2) every letter appears exactly twice in both strings.

Concerning the first assumption, the point is that in 2-MCSP, the minimum common partition never has to break such a duo. Thus, if there exists in A and B such a duo, it is possible to replace it by a new letter, solve the modified instance and then replace the new letter back by the original duo. Concerning the other, a letter that appears only once can be replaced by two copies of itself. A minimum common partition never has to use a break between these two copies, so they can be easily replaced back to a single letter, when the solution for the modified instance is found.

The main idea of the reduction is to represent a common partition of A and B as a truth assignment of a (properly chosen) set of binary variables. With each letter $a \in \Sigma$ we associate a binary variable X_a. For each letter $a \in \Sigma$, there are exactly two ways to map the two occurrences of a in A onto the two occurrences of a in B: either the first a from A is mapped on the first a in B and the second a from A on the second a in B, or the other way round. In the first case, we say that a is mapped *straight*, and in the other case that a is mapped *across*. Given a common partition π of A and B, if a letter $a \in \Sigma$ is mapped straight we set $X_a = 1$, and if a is mapped across we set $X_a = 0$. In this way, every common partition can be turned into truth assignment of the variables X_a, $a \in \Sigma$, and vice versa. Thus, there is one-to-one correspondence between truth-assignments for the variables X_a, $a \in \Sigma$, and common partitions (viewed as mappings) of A and B.

With this correspondence between truth assignments and common partitions, our next goal is to transform the two input strings A and B into a boolean formula φ such that

- φ is a conjunction of disjunctions (OR) and exclusive disjunctions (XOR),
- each clause contains at most two literals, and
- the minimum number of satisfied clauses in φ is equal to the number of breaks in a minimum common partition of A and B.

The formula φ consists of $n-1$ clauses, with a clause C_i for each specific duo $a_i a_{i+1}, i \in [n-1]$. For $i \in [n-1]$, let $s_i = 1$ if a_i is the first occurrence of the letter a_i in A (that is, the other copy of the same letter occurs on a position $i' > i$), and let $s_i = 2$ otherwise (that is, if a_i is the second occurrence of the letter a_i in A). Similarly, let $t_i = 1$ if b_i is the first occurrence of the letter b_i in B and let $t_i = 2$ otherwise. We are ready to define φ. There will be three types of clauses in φ.

If the duo $a_i a_{i+1}$ does not appear in B at all, we define $C_i = 1$. The meaning is that in this case, $i, i+1$ is a break in A in any common partition of A and B. We call such a position an *inherent break*. Let b be the number of clauses of this type.

If the duo $a_i a_{i+1}$ appears once in B, say as $b_j b_{j+1}$, let $Y = X_{a_i}$ if $s_i \neq t_j$, and let $Y = \neg X_{a_i}$ otherwise; similarly, let $Z = X_{a_{i+1}}$ if $s_{i+1} \neq t_{j+1}$ and let $Z = \neg X_{a_{i+1}}$ otherwise. We define $C_i = Y \vee Z$. In this way, the clause C_i is satisfied if and only if $i, i+1$ is a break in a common partition consistent with the truth assignment of X_{a_i} and $X_{a_{i+1}}$.

Similarly, if the duo $a_i a_{i+1}$ appears twice in B, we set $C_i = X_{a_i} \oplus X_{a_{i+1}}$ if $s_i = s_{i+1}$, and we set $C_i = \neg X_{a_i} \oplus X_{a_{i+1}}$ otherwise, where $\oplus$ denotes the exclusive disjunction. Again, the clause C_i is satisfied if and only if $i, i+1$ is a break in a common partition consistent with the truth assignment of X_{a_i} and $X_{a_{i+1}}$. Let k denote the number of these clauses.

By the construction, a truth assignment that satisfies the minimum number of clauses in $\varphi = C_1 \wedge \ldots \wedge C_{n-1}$ corresponds to a minimum common partition of A and B. In particular, the number of satisfied clauses is equal to the number of breaks in the common partition which is by one smaller than the number of blocks in the partition.

The formula φ resembles an instance of 2-SAT. However, 2-SAT formulas do not allow XOR clauses. One way to get around this is to replace every XOR clause by two OR clauses. This increases the length of the formula which in turn increases the resulting approximation ratio for 2-MCSP. In the rest of the section, we describe how to avoid this drawback.

Consider a duo $a_i a_{i+1}$ in A for which C_i is a XOR-clause. Then the duo $a_i a_{i+1}$ appears twice in B, and, by our assumption (1), the other occurrence of the letter a_i in A is followed by a letter different from a_{i+1} or the other occurrence of the letter a_i is the last letter in A. This implies that $k \leq b+1$.

Let $\bar{\varphi}$ be the boolean formula derived from φ by omitting clauses of the first type, that is, $\bar{\varphi} = \bigwedge_{i:C_i \neq 1} C_i$. Let φ' be the formula that we get from $\bar{\varphi}$ by replacing each XOR clause $(X \oplus Y)$ by $(X \vee Y) \wedge (\bar{X} \vee \bar{Y})$ and keeping all other clauses. Since for any values of boolean variables X and Y, $(X \oplus Y) + 1 = (X \vee Y) + (\bar{X} \vee \bar{Y})$ (when using the boolean values of the parentheses as integers), the minimum number of satisfied clauses in $\bar{\varphi}$ is exactly by k smaller than the minimum number of satisfied clauses in φ'.

Let s be the minimum number of satisfied clauses in the formula $\bar{\varphi}$. Then, $s+b+1$ is the size of a minimum common partition of A and B and the minimum number of satisfied clauses in the 2-SAT formula φ' is $s+k$. An α-approximation for MIN 2-SAT instance φ' satisfies at most $\alpha \cdot (s+k)$ clauses and the same truth assignment satisfies at most $\alpha \cdot (s+k) - k$ clauses in $\bar{\varphi}$. Considering the additional b breaks for clauses of the first type, this truth assignment corresponds to a common partition with at most $\alpha \cdot (s+k) - k + b \leq \alpha \cdot (s+b+1) - 1$ breaks. Since the size of the minimum common partition is $s+b+1$, this is an α-approximation. For unsigned MCSP, the proof is completed.

For signed MCSP, we use the same correspondence between truth assignments and common partitions; the only difference is in definition of the clauses C_i. We leave the details to the reader. □

3.2 Linear time 4-approximation for 3-MCSP

Similarly as in Section 3.1 we assume, without loss of generality, that no duo has three occurrences in A and in B at the same time. Again, the point is that if there exists a duo ab with three occurrences in A and three occurrences in B, for a 3-MCSP instance A, B, the duo ab never has to be broken in a minimum common partition of A and B, and therefore can be replaced by a new letter a' without altering the size of the optimal solution. For technical reasons we augment both strings at the end by a new character $a_{n+1} = b_{n+1} = \$$.

The main idea of the algorithm is to look for duos that must be broken in every common partition (e.g., a duo ab appearing twice in A but only once in B has this property) and to cut in both strings *every* occurrence of such duos; the hard part is to find sufficiently many duos broken in every common partition. Concerning the approximation bound, we will use a simple charging schema to prove it (since we are dealing with 3-MCSP, the algorithm makes only $O(1)$ cuts per single cut in the minimum partition). To show the correctness of the algorithm (i.e., the algorithm finds a common partition) we exploit again the relation of MCSP and the maximum independent set in the conflict graph G. More specifically, if the size of MIS in the conflict graph G, after several steps of the algorithm, equals the number of specific duos in the remaining unbroken substrings of A, then the algorithm has found a common partition of A and B; in our case the conflict graph in question has a simple structure that makes it possible to easily calculate the size of the MIS. The algorithm consists of three main phases, described in the rest of the section.

Phase 1. Before describing this phase, we need one more definition. A duo ab is *good* if the number of its occurrences in A equals the number of its occurrences in B, and is *bad* otherwise. As before, let m denote the size of a minimum common partition of a given pair of strings A and B.

Observation 3.3 *In every common partition of A and B, for every bad duo ab there must be at least one break immediately after some occurrence of a in A and at least one break immediately after some occurrence of a in B.*

In the first phase of the algorithm, for every bad duo ab, we cut both strings A and B after *every* occurrence of a. We charge the cuts of ab to the breaks that appear by Observation 3.3 in the optimal partition, that is, to the breaks after letter a. At most three cuts are charged to a single break. Let $\mathcal{A}$ and $\mathcal{B}$ denote the two multisets of strings we obtain from A and B after performing all these cuts.

Phase 2. At this point, every specific duo of $\mathcal{A}$ and $\mathcal{B}$ has either one or two matches. In the first case, we talk about a *unique duo* and a *unique match*, in the later case about an *ambiguous duo* and an *ambiguous match*. There are four vertices in the conflict graph corresponding to matches of an ambiguous duo and they are connected in a 4-cycle. The 4-cycle will be called a *square ab*; a vertex corresponding to a match of a unique duo will be called a *single*. We say that two duos *interfere* if a match of the first duo is in a conflict with a match of the other duo.

Let G_1 be the conflict graph of $\mathcal{A}$ and $\mathcal{B}$. The edges of G_1 can be classified into three groups: edges between two ambiguous matches, between an ambiguous match and a unique match, and, between two unique matches. We are going to have a closer look on these three cases.

Consider two ambiguous duos that interfere, say ab and bc. There are only two ways how the two occurrences of ab and the two occurrences of bc can appear in $\mathcal{A}$: either there are two occurrences of a substring abc, or a single occurrence of each of bc, abc and ab, in any order. There are the same possibilities for $\mathcal{B}$. Thus, there are only three basic ways how the duos ab and bc can appear in the strings $\mathcal{A}$ and $\mathcal{B}$ (up to symmetry of $\mathcal{A}$ and $\mathcal{B}$ and up to permutation of the depicted substrings):

$$\mathcal{A} = \ldots\ abc\ \ldots\ abc\ \ldots, \qquad \mathcal{B} = \ldots\ abc\ \ldots\ abc\ \ldots \tag{1.1}$$
$$\mathcal{A} = \ldots\ bc\ \ldots\ abc\ \ldots\ ab\ \ldots, \quad \mathcal{B} = \ldots\ bc\ \ldots\ abc\ \ldots\ ab\ \ldots \tag{1.2}$$
$$\mathcal{A} = \ldots\ abc\ \ldots\ abc\ \ldots, \qquad \mathcal{B} = \ldots\ bc\ \ldots\ abc\ \ldots\ ab\ \ldots \tag{1.3}$$

The corresponding subgraphs of G_1 are depicted in Figure 3. We observe several things. If we want to keep all occurrences of ab and bc in case (1.1) and it is already given how to match the ab duos (bc, resp.), then it is uniquely given how to match the bc duos (ab, resp.). If we want to keep all occurrences of ab and bc in case (1.2), then there is only one way how to match them all; we say that the duos have a *preference* and we call these matches the *preferred matches* of ab and bc. Concerning case (1.3), in any common partition of $\mathcal{A}$ and $\mathcal{B}$ at least one occurrence of the duos ab and bc must be broken.

Let ab be an ambiguous duo and bc a unique duo. There are two possibilities how they may interfere (cf. Figure 3):

$$\mathcal{A} = \ldots\ abc\ \ldots\ ab\ \ldots, \mathcal{B} = \ldots\ abc\ \ldots\ ab\ \ldots \tag{2.1}$$
$$\mathcal{A} = \ldots\ ab\ \ldots\ abc\ \ldots, \mathcal{B} = \ldots\ ab\ \ldots\ ab\ \ldots\ bc\ \ldots \tag{2.2}$$

Similarly as before, there is only one way how to match all duos in case (2.1). We say that the duos have a *preference* and we call these matches the *preferred matches* of ab and bc. Concerning (2.2), in any common partition of $\mathcal{A}$ and $\mathcal{B}$ at least one of ab and bc must be broken.

Finally, let ab and bc be two unique duos. There is only one way how they may interfere.

$$\mathcal{A} = \ldots\ abc\ \ldots, \mathcal{B} = \ldots\ ab\ \ldots\ bc\ \ldots \tag{3.1}$$

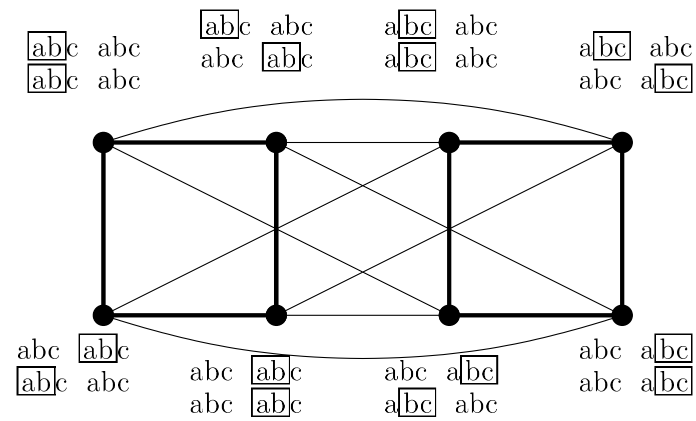
Again, in any common partition of $\mathcal{A}$ and $\mathcal{B}$, ab or bc must be a break.

In the second phase of the algorithm, we cut all occurrences of duos ab and bc that have an interference of type (1.3), (2.2) or (3.1). We charge these cuts to the breaks that appear, by the above observations, in the optimal solution. At most four cuts are charged to a single break.

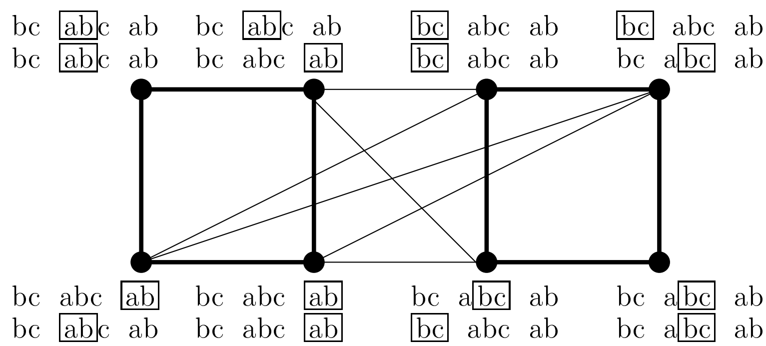
Phase 3. Let $\mathcal{A}_2$ and $\mathcal{B}_2$ denote the two sets of strings after performing all cuts of phase two, and let G_2 be the corresponding conflict graph. Note that a duo might have more than one preferred match. The problem we are facing now is to decide which preferences to obey and which not since the more preferences we obey the less breaks we need. A graph H of *inconsistent preferences* will help us.

By definition, a duo ab *without* preference has interferences of type (1.1) only, and therefore interferes with at most two other duos. We already observed that if a duo ab has an interference of type (1.1), say with a duo bc, *and* two matches of the duo bc are

1.1



1.2



1.3

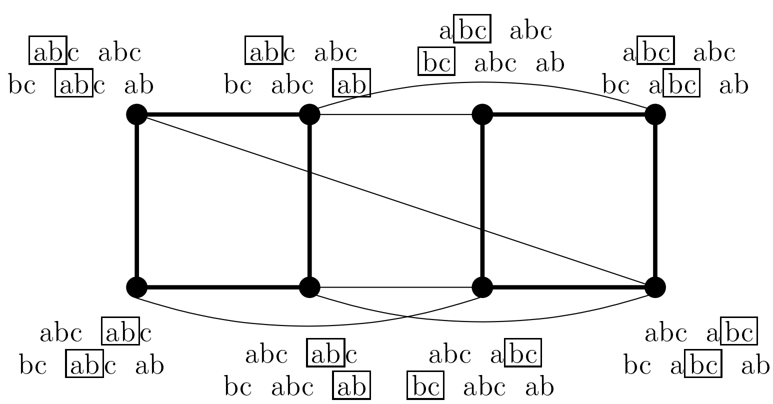


Figure 3: Subgraphs of G_1 corresponding to matches of two interfering duos of types (1.1), (1.2) and (1.3). Each node represents a match, e.g., the leftmost upper node in all three subgraphs represents the match between the left occurrence of ab in $\mathcal{A}$ with the left occurrence of ab in $\mathcal{B}$. Bold edges only highlight which conflicts appear between matches of the same duo. Observe that the order of the substrings ab, bc and abc does not affect the structure of these conflict subgraphs.

already fixed, *and* no new breaks are allowed, then the matches of the duo ab are uniquely given. In this way, a duo without preference transmits a fixed preference of a neighboring duo on one side to a neighboring duo on its other side, etc. A difficulty arises when a preference of one duo, say bc, is transmitted by a sequence of duos without preferences to another duo with a preference, say xy, and the preference transmitted to xy is different from the preference of xy. Then, in every common partition, at least one specific duo bc, or xy, or one of the transmitting duos, must be a break. We say that the preferences of bc and xy are *inconsistent*. Similarly, if bc has two different preferences, we also say that bc is inconsistent with bc.

We define the graph $H = (V_H, E_H)$ of *inconsistent preferences*. The vertex set V_H consists of all duos with a preference and the set of edges E_H consists of all pairs of duos with inconsistent preferences (which includes loops for duos inconsistent by themselves). By the above discussion, the graph H can be constructed in time linear in the number of duos.

Lemma 3.4 *The size of a minimum vertex cover for H is a lower bound for the number of breaks in a minimum common partition of $\mathcal{A}_2$ and $\mathcal{B}_2$.*

Proof: Consider a minimum common partition π of $\mathcal{A}_2$ and $\mathcal{B}_2$. By Lemma 1.1, π corresponds to a maximum independent set of G_2, and thus, also to a minimum vertex cover of G_2. Let $C_2 \subseteq V_2$ denote nodes in this vertex cover. We observe that for every square ab in G_2, at least two of its vertices must be in C_2 and for every square ab with three or four vertices in C_2, there must be a break between some a and b in π. Similarly, for every single cd in C_2, there must be a break between c and d in π.

We are going to derive a vertex cover C for H from the vertex cover C_2 for G_2. The nodes in C will come from three different sources:

- For every square ab with preference in G_2, if the square ab has three or four vertices in C_2, we put the vertex $ab \in V_H$ to C.
- For every single $ab \in C_2$, we put $ab \in V_H$ to C.
- For every square ab without preference in G_2, if the square ab has three or four vertices in C_2, we add its closest duo with preference to C (ties broken arbitrarily).

In this way, for every pair of duos with inconsistent preferences, at least one of them will be in C, and each vertex in C can be charged to a different break in π. Thus, C is indeed a vertex cover of size at most equal to the number of breaks of the common partition π. □

In phase three, the algorithm cuts all duos corresponding to a minimum vertex cover of H (resp., to 2-approximation of a minimum vertex cover). And we charge these cuts to the breaks in the minimum vertex cover, by the above lemma. Let $\mathcal{A}_3$ and $\mathcal{B}_3$ denote the two sets of substrings we are left with; the algorithm outputs $(\mathcal{A}_3, \mathcal{B}_3)$.

The complete algorithm can be schematically summarized as follows:

Algorithm

Phase 1: Cut all bad duos.
Phase 2: Cut all duos involved in conflicts of type (1.3), (2.1) and (3.1).
Phase 3: Cut all duos corresponding to (approximated) minimum vertex cover in the graph H of inconsistent preferences.

Theorem 3.5 *The described algorithm has approximation ratio* 4 *and runs in linear time, for both unsigned and signed* 3-MCSP.

Proof: First we have to prove that the algorithm really computes a common partition; this is done in the next lemma.

Lemma 3.6 *The pair* $(\mathcal{A}_3, \mathcal{B}_3)$ *is a common partition of* A *and* B*.*

Proof: Let G_3 be the conflict graph of $\mathcal{A}_3$ and $\mathcal{B}_3$. We are going to construct a maximum independent set in G_3, corresponding to a common partition of $\mathcal{A}_3$ and $\mathcal{B}_3$ with no additional breaks. The procedure is based on the observations in proof of Lemma 3.4. All we have to do is to note that there are no inconsistent duos in G_3. Thus, by following the preferences for matches, we will never run into a conflict. If there is a component in G_3 without any preference, we choose arbitrarily a match for some duo and consider it as a preference.

More specifically, we take to our independent set

- all singles, and,
- from every duo with preference, the two vertices corresponding to the preferred matches, and,
- from every duo without preference the two vertices corresponding to the matches that are forced by a neighboring duo.

In this way, every single from G_3 appears in the independent set, and for every duo with four possible matches, two of them are in the independent set. This is a maximum independent set corresponding to a common partition with no additional breaks. □

To prove the approximation ratio, we observe that the cuts of the algorithm are charged in every phase to different breaks in the optimal solution. In every step we charged at most four cuts of the algorithm to a break in the optimal solution which results into a 4-approximation.

Concerning the running time, using a hash table, Phases 1 and 2 can be implemented in time $O(n)$. We already noted that the graph H of inconsistent preferences can be constructed in linear time. Since the number of edges in H is $O(n)$, a 2-approximation of the vertex cover can be computed in time $O(n)$, yielding a total time $O(n)$.

For signed MCSP, the algorithm goes along the same lines, it only has to consider $+a+b$ and $-b-a$ as the occurrences of the same duo. □

Acknowledgment

We thank Xin Chen for introducing us the MCSP; this motivated our work. We wish to thank Tao Jiang, Marek Chrobak, Neal Young and Stefano Lonardi for many useful discussions. We also gratefully acknowledge comments given to us by Jiří Sgall on early version of the paper.

References

[1] A. Avidor and U. Zwick. Approximating MIN k-SAT. In *Proceedings of 13th International Symposium on Algorithms and Computation (ISAAC)*, volume 2518 of *Lecture Notes in Computer Science*, pages 465–475, 2002.

[2] P. Berman and M. Karpinski. On some tighter inapproximability results. In *Proceedings of the of 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. ACM Press, 1997.

[4] X. Chen, J. Zheng, Z. Fu, P. Nan, Y. Zhong, S. Lonardi, and T. Jiang. Computing the assignment of orthologous genes via genome rearrangement. In *Proceedings of 3rd Asia-Pacific Bioinformatics Conference (APBC)*, pages 363–378, 2005. To appear in IEEE/ACM Transactions on Computational Biology and Bioinformatics.

[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)*, pages 84–95, 2004. To appear in ACM Transactions on Algorithms.

[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. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman & Company, San Francisco, 1978.

[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)*, volume 3341 of *Lecture Notes in Computer Science*, pages 484–495, 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, 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] D. Sankoff and N. El-Mabrouk. Genome rearrangement. In T. Jiang, Y. Xu, and M. Q. Zhang, editors, *Current Topics in Computational Molecular Biology*, pages 135–155. The MIT Press, 2002.

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

[14] G. A. Watterson, W. J. Ewens, T. E. Hall, and A. Morgan. The chromosome inversion problem. *Journal of Theoretical Biology*, 99:1–7, 1982.