Minimum Common String Partition Revisited

Haitao Jiang[1,2], Binhai Zhu[1], Daming Zhu[2], and Hong Zhu[3]

[1] Department of Computer Science, Montana State University, Bozeman, MT 59717-3880, USA. Email: `htjiang,bhz@cs.montana.edu`.
[2] School of Computer Science and Technology, Shandong University, Jinan, China. Email: `dmzhu@sdu.edu.cn`.
[3] College of Software, East China Normal University, Shanghai, China. Email: `hzhu@sei.ecnu.edu.cn`.

Abstract. Minimum Common String Partition (MCSP) has drawn much attention due to its application in genome rearrangement. In this paper, we investigate three variants of MCSP: $MCSP^c$, which requires that there are at most c elements in the alphabet; d-MCSP, which requires the occurrence of each element to be bounded by d; and x-balanced MCSP, which requires the length of blocks being in range $(n/k - x, n/k + x)$, where n is the length of the input strings, k is the number of blocks in the optimal common partition and x is a constant integer. We show that $MCSP^c$ is NP-hard when $c \geq 2$. As for d-MCSP, we present an FPT algorithm which runs in $O^*((d!)^{2k})$ time. As it is still unknown whether an FPT algorithm only parameterized on k exists for the general case of MCSP, we also devise an FPT algorithm for the special case x-balanced MCSP parameterized on both k and x.

1 Introduction

String comparison has drawn a lot of attention due to its applications in computational biology, text processing and compression. In this paper, we revisit the Minimum Common String Partition (MCSP) problem which has a close relation to the genome rearrangement problems such as Edit Distance, Sorting by Reversals and Sorting by Transpositions, etc. In fact, MCSP was initially studied as 'edit distance with moves' [4, 15].

A partition P of a string X is a sequence $P = \langle P_1, P_2, \ldots, P_m \rangle$ of strings whose concatenation is equal to X, that is $P_1 P_2 \ldots P_m = X$. The string P_i's are called the blocks of P. Given a partition P of a string X and a partition Q of a string Y, we say that the pair $\pi = (P, Q)$ is a common partition of X and Y if Q is a permutation of P, that is, there exists a permutation σ on $[m]$ such that $P_i = Q_{\sigma_i}, 1 \leq i \leq m$. The *minimum common string partition problem* is to compute a common partition of X and Y with the minimum number of blocks.

In the Minimum Common String Partition (MCSP) problem, we are given two strings X and Y of length n over an alphabet Σ. Let each element appear the same number of times in X and Y. Throughout this paper, we assume that X and Y satisfy this condition. Clearly, this is a sufficient and necessary condition for X

and Y to have a common string partition. For example, two strings $X = beabcdb$ and $Y = abdbebc$ have a common partition $(\langle b, e, ab, c, db\rangle, \langle ab, db, e, b, c\rangle)$. There are several variants of MCSP. The restricted version where each letter occurs at most d times in each input string is denoted as d-MCSP. Another important version where the input strings are over an alphabet with size bounded by c, is abbreviated as $MCSP^c$. We also introduce a new version where the resulting blocks in the partition are designated to have nearly the equal length, we call it x-balanced MCSP. For the x-balanced MCSP problem, if the optimum solution has k blocks, the length of each block ranges from $n/k-x$ to $n/k+x$. The signed minimum common string partition problem (SMCSP) is also a variant of MCSP in which each letter of the two input strings is given a "+" or "-" sign. For a string S with signs, let $-S$ denote the reverse of S, with each letter sign flipped. The definition of common partition on signed strings is a bit different from that of unsigned strings, where $P_i = Q_{\sigma_i}$ or $P_i = -Q_{\sigma_i}$.

Related Work

The problem d-MCSP is well studied. 2-MCSP (and therefore MCSP) is NP-hard; moreover, APX-hard [9]. Several approximation algorithms are known for the problem [9, 12]. Chen et al. [1] studied the problem of computing signed reversal distance with duplicates (SRDD). They introduced the signed minimum common partition problem as a tool for dealing with SRDD and observed that for any two signed strings X and Y, the size of a minimum common partition and the minimum number of reversal operations needed to transform X and Y, are within a multiplicative factor 2 of each other. Kolman and Walen [14] devised an $O(d^2)$-approximation algorithm running in $O(n)$ time for SRDD.

Chrobak et al. [3] analyzed the greedy algorithm for MCSP. They showed 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 $O(n^{0.67})$. In fact, the same bounds hold for SMCSP. Kaplan and Shafrir [11] improved the lower bound to $\Omega(n^{0.46})$ when the input strings are over an alphabet of size $O(\log n)$. Kolman [13] described a simple modification of the greedy algorithm, and the approximation ratio of the modified algorithm is $O(p^2)$ for p-MCSP. Christie and Irving [2] proved that the problem of computing (unsigned) reversal distance is NP-hard for binary strings, it turns out that this problem has some connection to $MCSP^c$.

In the framework of parameterized complexity, Damaschke first solved MCSP by an FPT algorithm with respect to a combination of parameters k (size of the optimum solution), r (the repetition number) and t (the distance ratio) [5]. In that paper, the repetition number r of a string X is the maximum integer i so that $X = uv^iw$ holds for some strings u, v, w and v is nonempty, where v^i is the concatenation of i v's; the distance ratio is defined as $t = n/m$, where m is the shortest blocks in the optimum solution.

Our Contribution

We prove that the problem $MCSP^c$ is NP-complete when $c \geq 2$. For d-MCSP, we present an FPT algorithm for it with running time $O^*((d!)^{2k})$. When the MCSP solution is supposed to be x-balanced, we devise an FPT algorithm with parameter k and x which runs in $O((2x)^k k! n))$ time.

2 Preliminaries

As aforementioned, we recall the formal definition of parameterized MCSP.

Minimum Common String Partition:
Input: two strings X, Y over an alphabet Σ, an positive integer k.
Question: Can X have a partition P and Y have a partition Q such that Q is a permutation of P, where both P and Q contain k blocks ?

We say that the elements from Σ *occur* or *appear* in X and Y. A specific occurrence of some element is called a *letter*. The strings in the partition are called blocks and there is a *break* or *cut* between two consecutive blocks in X or Y.

An FPT (Fixed-Parameter Tractable) algorithm for a decision problem Π with input parameters $k_1, \cdots, k_i$ is an algorithm which solves the problem in $O(f(k_1, \cdots, k_i)n^c)$=$O^*(f(k_1, \cdots, k_i))$ time, where f is any function only on k_1, $\cdots$, k_i, n is the input size and c is some fixed constant not related to $k_1, \cdots, k_i$. For convenience we also say that Π is in FPT. More details on FPT algorithms can be found in [6, 7].

It is open whether an FPT algorithm exists for the general *MCSP* problem with k, the size of the optimal solution, being the unique parameter. In fact this problem is also connected to computing the breakpoint distance between genomes with gene duplications, which is also NP-complete and the existence of an FPT algorithm is also unknown [10]. In this paper, we try to handle variants of *MCSP* by using additional parameters.

This paper is organized as follows. In Section 2, we present the NP-completeness of $MCSP^c$. In Section 3, we present FPT algorithms for the two variants of *MCSP*. In Section 4, we conclude the paper with several open questions.

3 Hardness for $MCSP^c$

In this section, we prove that $MCSP^c$ is NP-complete when $c \geq 2$ by a reduction from 3-PARTITION [8]. Firstly, we recall the formal definition of 3-PARTITION.

3-PARTITION :
Input: Positive integers n and B, and a set of positive integers $A = \{a_1, a_2, \ldots, a_{3n}\}$, with $B/4 < a_i < B/2$ and $\sum_{a_i \in A} a_i = nB$.
Question: Can A be partitioned into n disjoint sets $S_1, S_2, \ldots, S_n$ such that, for $1 \leq i \leq n$, $\sum_{a_j \in S_i} a_j = B$?

The problem 3-PARTITION is strongly NP-hard: that is, there is a polynomial $p(n)$ such that it is still NP-hard when all the a_i's are at most $p(n)$. Our reduction is polynomially bounded for instances of this type.

Given an instance of 3-PARTITION with integers (weights) $a_1, a_2, \ldots, a_{3n}$, we construct two strings X, Y for an instance of $MCSP^2$ as follows

$$X = 11110^{a_1}11110^{a_2}\cdots 11110^{a_{3n}}$$

$$Y = (0^B 1)^n 1^{11n}$$

For the sake of clarity, we first make the following claims.

Claim. Any block in the optimal common string partition has the form 0^p1^q, 0^p1, 10^p, 0^p or 1^q where $p, q \geq 1$; moreover, there is exactly one block of the form 0^p1^q, for $p \geq 1, 2 \leq q \leq 4$.

Proof. Suppose to the contrary that there is a common block H with the form $0^p1^q0^r\cdots$, then from the format of X and Y, q is either equal to one or equal to four. But H cannot be a substring of X when $q = 1$ and cannot be a substring of Y when $q = 4$. Similarly, if there exists a common block in the optimal common partition $H' = 1^q0^p1^s\cdots$, then H', as a substring of X, must satisfy that $p < B$; and as a substring of Y, must satisfy that $p = B$. which is a contradiction. Consequently, all the common blocks must be of the form 0^p1^q, 10^p, 0^p and 1^q, as there is only one 1 between two 0's in Y. To see why there is one block of the form 0^p1^q for $p \geq 1, 2 \leq q \leq 4$, notice that such block can only occur once as a substring at the end of Y. So the claim holds. □

Claim. Except for the common block 0^p1^q for $p \geq 1, 2 \leq q \leq 4$, the interior '11' from other '1111' in X is matched to '11' from 1^{11n+1} in Y.

Proof. From the first claim we conclude that, except for the common block 0^p1^q for $p \geq 1, 2 \leq q \leq 4$, if a common block contains some 0's, then it contains at most one 1. So the interior two 1's from other '1111' must be in the common blocks containing only 1's. Since there are no consecutive 1's in Y besides the substring 1^{11n+1}, the claim holds. □

In fact, except for the common block 0^p1^q for $p \geq 1, 2 \leq q \leq 4$, the interior '11' from other '1111' in X is matched to '11' from the last $11n + 1 - q$ 1's in Y.

Theorem 1. *$MCSP^2$ is NP-complete.*

Proof. As $MCSP^2$ is obviously in NP, it suffices to prove that the 3-PARTITION instance has a precise partition if and only if X and Y have a common partition with $6n - 1$ blocks.

($\Rightarrow$) Assume that $S_1, S_2, \ldots, S_n$ satisfy $\sum_{a_j \in S_i} a_j = B$, for all i. For exactly one $S_i = \{a_p, a_q, a_r\}$, we obtain three blocks from X: $0^{a_p}, 0^{a_q}, 0^{a_r}1111$ and one block from Y: 0^B1111. For any other $S_i = \{a_p, a_q, a_r\}$, we obtain three blocks from X: $0^{a_p}, 0^{a_q}, 0^{a_r}1$ and one block from Y: 0^B1. Obviously, the block from Y can be split into three blocks corresponding to the blocks from X. Then there are $2n$ remaining blocks of the form '1111' and $n - 1$ remaining blocks of the form '111' in X and all these blocks are separated (not adjacent). The unique block left in Y is 1^{11n-3}. Consequently, we obtain a common partition of X, Y with $6n-1$ blocks.

($\Leftarrow$) From the above two claims, we can see that, except for one '1111' substring, all the 0^{a_i}'s and the interior substrings '11' of '1111' are in mutually distinct blocks in the common partition. So there are at least $6n-1$ blocks in any optimal common partition. If the number of blocks is exactly $6n-1$, which means that each 0^{a_i} substring of X falls into one block in the optimal common partition and those blocks are mutually distinct. Then each substring of the form 0^B in Y is split into exactly three blocks of the form 0^{a_i}, the reason is that $B/4 < a_i < B/2$. Thus, the 3-PARTITION instance has a valid partition. $\square$

Since we can construct an instance of $MCSP^c$ when $c > 2$ from an instance of $MCSP^2$ by adding the same substring composed of letters other than those in $MCSP^2$ at the end of the two input strings, we can easily obtain the following corollary.

Corollary 1. *$MCSP^c$ is NP-complete for $c \geq 2$.*

4 The FPT Algorithms

In this section, we design FPT algorithms for two variants of MCSP. As it is still unknown whether MCSP has an FPT algorithm parameterized only on k, we hope these algorithms could help shed light on answering this open question. On the other hand, these two variants are closely related to genome rearrangement problems, hence are meaningful practically.

4.1 On *d-MCSP*

Firstly, we introduce the following definitions and notations. A *duo* is a substring of length two. A *specific duo* is an occurrence of a duo in X or Y. Note that two specific duos may share a letter when they form a substring of length three. Two specific duos are continuous if they form a substring of length three. A match is a pair $(a_i a_{i+1}, b_j b_{j+1})$ of specific duos, one from X and the other one from Y, such that $a_i = b_j$ and $a_{i+1} = b_{j+1}$. We also say that the pair of specific duos are matched to each other if they form a match. Two matches $(a_i a_{i+1}, b_j b_{j+1})$ and $(a_k a_{k+1}, b_l b_{l+1})$ can *co-exist* if (1) $\{i, i+1\} \cap \{k, k+1\} = \emptyset$ and $\{j, j+1\} \cap \{l, l+1\} = \emptyset$; or (2) $k = i+1$ and $l = j+1$; or (3) $i = k+1$ and $j = l+1$. Otherwise, if (1) $i = k, j \neq l$; or (2) $j = l, i \neq k$; or (3) $k = i+1, l \neq j+1$; or (4) $i = k+1, j \neq l+1$, then the matches form a *conflict*.

We take two strings $X = a^{-1}a^{-2}a^{-3}ba^{-4}$ and $Y = a^{-1}a^{-2}a^{-3}a^{-4}b$, where $a^{-1} = a^{-2} = a^{-3} = a^{-4} = a$, for an example. There exists a duo $\{aa\}$ in both X and Y, but there are two specific duos $\{a^{-1}a^{-2}, a^{-2}a^{-3}\}$ in X and three specific duos $\{a^{-1}a^{-2}, a^{-2}a^{-3}, a^{-3}a^{-4}\}$ in Y; moreover, these specific duos are continuous in X and Y respectively. Matches $(a^{-1}a^{-2}, a^{-2}a^{-3})$ and $(a^{-2}a^{-3}, a^{-3}a^{-4})$ can co-exist, but $(a^{-1}a^{-2}, a^{-1}a^{-2})$ and $(a^{-2}a^{-3}, a^{-3}a^{-4})$ form a conflict.

Note that two matches with continuous specific duos in one string are not in conflict if and only if these duos are matched to two continuous specific duos in the other string.

Lemma 1. *Each duo appears the same number of times in a common partition of X and Y.*

Proof. Otherwise, there must be such a specific duo that is unmatched in the partition. □

Lemma 2. *If each duo appears the same number of times in a partition of X and a partition of Y, then two matches are in conflict if and only if the specific duos are continuous in one string and not continuous in the other.*

Proof. Two matches are in conflict means that there exists a letter which is matched to two occurrences of that element in the two matches. Clearly in this case the two specific duos cannot be continuous in both strings. □

Lemma 3. *Given a pair of partition $\pi = (P, Q)$ for X and Y, if all specific duos are matched and all matches are not in conflict then π is a common partition.*

Proof. For any block $G = g_1g_2 \cdots g_m$ in P, since all the specific duos g_ig_{i+1} are matched, there must be matches (g_ig_{i+1}, h_jh_{j+1}) and $(g_{i+1}g_{i+2}, f_kf_{k+1})$ that can be realized at the same time. Since g_ig_{i+1} and $g_{i+1}g_{i+2}$ are continuous duos, h_jh_{j+1} and f_kf_{k+1} should be continuous as well; that is, h_{j+1} and f_k are the same letter. As the above analysis holds for all i, we can see that there is a block $H = h_1h_2 \cdots h_m$ in Q such that $G = H$. The lemma holds. □

Our FPT algorithm *Cut-Duos* is based on the above three lemmas. The rough idea is to cut redundant specific duos such that all duos appear the same number of times in X and Y, then cut one of the continuous duos that corresponds to two matches in conflict. In the algorithm, we use arrays C and D indexed by duos to store the number of occurrence of each duo in X and Y respectively. That is, $C_{ab} = r$ means ab appears r times in X.

```
Algorithm Cut-Duos
Input: two strings $X, Y$ such that each element appears at most $d$ times in each string
Output: A common partition $(P, Q)$
 1   Compute the number of occurrence of each duo in $X$ and $Y$, and store them
in arrays $C$ and $D$.
 2   For every duo $ab$ in $X$, if $r = C_{ab} > D_{ab} = s$
   2.1   Choose $r - s$ $ab$'s in $X$, cut them.
   2.2   Compute matches between the remaining $ab$'s in $X$ and $Y$.
 3   For every duo $ab$ in $Y$, if $r = C_{ab} < D_{ab} = s$
   3.1   Choose $s - r$ $ab$'s in $Y$, cut them.
   3.2   Compute matches between the remaining $ab$'s in $X$ and $Y$.
 4   For every two matches $(ab, ab)$ and $(bc, bc)$ computed in step 2 and step 3
with the common element $b$, if they are in conflict
   4.1   Choose one duo from $ab$ and $bc$, cut it in both $X$ and $Y$.
 5   Return the remaining blocks in $X$ and $Y$ respectively as the common
partition $(P, Q)$.
```

Theorem 2. *Algorithm Cut-Duos computes a minimum common partition for the two input strings X and Y in $O^*((d!)^{2k})$ time.*

Proof. Correctness: firstly all specific duos are matched after executing Step 2 and Step 3, then all the remaining matches can co-exist after executing Step 4. Consequently, Lemma 3 guarantees that algorithm *Cut-Duos* computes a common partition for the two input strings.

Time Analysis: In the algorithm *Cut-Duos*, Step 2 has a recurrence relation

$$f(k) = \binom{r}{s}(s!)f(k-(r-s)).$$

Step 3 has a similar recurrence relation. Step 4 has a recurrence relation

$$f(k) = 2f(k-1).$$

Since $r, s \leq d$ and $f(k)$ achieves its maximum value when $r - s = 1$, we have $f(k) \leq O^*((d!)^{2k})$. □

4.2 On x-balanced MCSP

In this subsection, we deal with the x-balanced MCSP problem. For the specific applications in genome arrangement, due to the relation between MCSP and genome rearrangement, each block corresponds to a gene and each gene should have roughly the same gene content. So the blocks are supposed to have nearly the equal length, possibly with some small deviations. Assume that the optimal solution size is k, then we can see that every block is of some length between $(n/k) + x$ and $(n/k) - x$.

The idea of the FPT algorithm is to cut the input string into k blocks of length ranging from $(n/k) - x$ to $(n/k) + x$, then compute all possible matches between the resulting k blocks. In the algorithm, assume that the length of the ith block X_i in X is $L(i)$. When we refer to *positions* from integer u to v $(u < v)$ in X, we mean the set of duos $\{x_u x_{u+1}, x_{u+1}x_{u+2}, \ldots, x_{v-1}x_v\}$. The algorithm *Cut-Depending-on-Length* is presented as follows.

Theorem 3. *Algorithm Cut-Depending-on-Length computes a minimum common partition for the two input strings X and Y in $O((2x)^k k! n)$ time.*

Proof. In the algorithm we branch on Step 2 and Step 3. Step 2 runs in $(2x)^{k-1}$ times. Step 3 has $k!$ possibilities. Step 4 and Step 5 run in $O(n)$ time. So the total running time of the algorithm is $O((2x)^k k! n))$. □

We comment that Algorithm Cut-Depending-on-Length works for other variants of MCSP whenever they are x-balanced.

5 Concluding Remarks

An earlier version of this paper appears in FAW'10, LNCS 6213, pp.45-52. In this paper, we deal with some variants of MCSP. The first one, $MCSP^c$, is proved to be NP-complete when $c \geq 2$. For d-MCSP and x-balanced MCSP, we devise FPT algorithms for them. The original MCSP problem seems difficult to be solved with an FPT algorithm only on the parameter k, so some extra practical parameters which make the problem fixed-parameter tractable are meaningful. As d-MCSP has close relation to some specific genome rearrangement problem, it would be interesting to design good approximations for the d-MCSP problem.

```
Algorithm *Cut-Depending-on-Length*
Input: *an x-balanced instance of MCSP*
Output: *A common partition (P,Q)*
 1   Compute the range of the length of blocks.
 2   For input string $X$, for $1 \leq i \leq k-1$
   2.1  Let $S_0$ be set of position from $n-(n/k+x)(k-i)$ to $n-(n/k-x)(k-i)$
   2.2  Let $S_1$ be set of position from
$\sum_{1\leq j<i} L(j)+(n/k-x)$ to $\sum_{1\leq j<i} L(j)+(n/k+x)$
   2.3  Choose a duo from $S_0 \cap S_1$ and cut it.
 3   Generate a random permutation $T = t_1, t_2, \cdots, t_k$ on $\{1, 2, \ldots, k\}$.
 4   For $i = 1$ to $k$, cut on $Y$ such that the length of the $i$th block $Y_i$ in $Y$
is equal to $L(t_i)$.
 5   For $i = 1$ to $k$, check whether $Y_i = X_{t_i}$.
 6   Return a common partition $(P,Q)$ with $k$ common blocks, if exists.
```

Acknowledgments

This research is partially supported by NSF grant DMS-0918034, and by NSF of China under grant 60928006 and 61070019. We also thank Minghui Jiang and anonymous reviewers for some constructive comments.

References

1. X. Chen and J. Zheng and Z. Fu and P. Nan and Y. Zhong and S. Lonardi and T. Jiang. Computing the assignment of orthologous genes via genome rearrangement. In *Proc. of the 3rd Asia-Pacific Bioinformatics Conf (APBC'05)*, pages 363-378, 2005.
2. D. A. Christie and R. W. Irving. Sorting strings by reversals and by transpositions. *SIAM Journal on Discrete Mathematics*, 14(2):193-206 2001.
3. M. Chrobak and P. Kolman and J. Sgall. The greedy algorithm for the minimum common string partition problem. In *Proc. of the 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'04)*, LNCS 3122, pages 84-95, 2004.
4. G. Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. In *Proc. of the 13th ACM-SIAM Symposium on Discrete Algorithms (SODA'02)*, pages 667-676, 2002.

5. P. Damaschke. Minimum Common String Partition Parameterized. In *Proc. of the 8th Workshop on Algorithms in Bioinformatics (WABI'08)*, LNCS 5251, pages 87-98, 2008.
6. R. Downey and M. Fellows. *Parameterized Complexity*, Springer-Verlag. 1999.
7. J. Flum and M. Grohe. *Parameterized Complexity Theory*, Springer-Verlag. 2006.
8. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman. 1979.
9. A. Goldstein and P. Kolman and J. Zheng. Minimum common string partitioning problem: Hardness and approximations. In *Proc. of the 15th International Symp. on Algorithms and Computation (ISAAC'04)*, LNCS 3341, pages 473-484, 2004. also in: The Electronic Journal of Combinatorics 12 (2005), paper R50.
10. H. Jiang and C. Zheng and D. Sankoff and B. Zhu. Scaffold Filling under the Breakpoint Distance. In *Proc. of the 8th Annual RECOMB Satellite Workshop on Comparative Genomics (RECOMB-CG'10)*, LNBI 6398, pages 83-92, 2010.
11. H. Kaplan, N. Shafrir. The greedy algorithm for edit distance with moves. *Inf. Process. Lett*, 97(1):23-27 2006.
12. P. Kolman and T. Walen. Reversal Distance for Strings with Duplicates: Linear Time Approximation Using Hitting Set. In *Proc. 4th Workshop on Approximation and Online Algorithms (WAOA'06)*, LNCS 4368, pages 279-289, 2006.
13. P. Kolman. Approximating reversal distance for strings with bounded number of duplicates. In *Proc. of the 30th International Symposium on Mathematical Foundations of Computer Science (MFCS'05)*, LNCS 3618, pages 580-590, 2005.
14. P. Kolman and T. Walen. Approximating reversal distance for strings with bounded number of duplicates. *Discrete Applied Mathematics*, 155(3):327-336, 2007.
15. D. Shapira and J. Storer. Edit distance with move operations. In *Proc. of the 13th Annual. Symposium on Combinatorial Pattern Matching (CPM'02)*, LNCS 2373, pages 85-98, 2002.