Fast Matching-based Approximations for Maximum Duo-Preservation String Mapping and its Weighted Variant

Brian Brubach[1]
Department of Computer Science, University of Maryland, College Park, MD 20742, USA
bbrubach@cs.umd.edu
https://orcid.org/0000-0003-1520-2812

Abstract

We present a new approach to approximating the Maximum Duo-Preservation String Mapping Problem (MPSM) based on massaging the constraints into a tractable matching problem. MPSM was introduced in Chen, Chen, Samatova, Peng, Wang, and Tang [10] as the complement to the well-studied Minimum Common String Partition problem (MCSP). Prior work also considers the k-MPSM and k-MCSP variants in which each letter occurs at most k times in each string. The authors of [10] showed a k^2-appoximation for $k \geq 3$ and 2-approximation for $k = 2$. Boria, Kurpisz, Leppänen, and Mastrolilli [6] gave a 4-approximation independent of k and showed that even 2-MPSM is APX-Hard. A series of improvements led to the current best bounds of a $(2+\epsilon)$-approximation for any $\epsilon > 0$ in $n^{O(1/\epsilon)}$ time for strings of length n and a 2.67-approximation running in $O(n^2)$ time, both by Dudek, Gawrychowski, and Ostropolski-Nalewaja [16]. Here, we show that a 2.67-approximation can surprisingly be achieved in $O(n)$ time for alphabets of constant size and $O(n + \alpha^7)$ for alphabets of size α.

Recently, Mehrabi [28] introduced the more general weighted variant, Maximum Weight Duo-Preservation String Mapping (MWPSM) and provided a 6-approximation. Our approach gives a 2.67-approximation to this problem running in $O(n^3)$ time. This approach can also find an $8/(3(1-\epsilon))$-approximation to MWPSM for any $\epsilon > 0$ in $O(n^2\epsilon^{-1} \lg \epsilon^{-1})$ time using the approximate weighted matching algorithm of Duan and Pettie [15].

Finally, we introduce the first streaming algorithm for MPSM. We show that a single pass suffices to find a 4-approximation on the size of an optimal solution using only $O(\alpha^2 \lg n)$ space.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases approximation algorithm, maximum duo-preservation string mapping, minimum common string partition, string comparison, streaming algorithm, comparative genomics

Digital Object Identifier 10.4230/LIPIcs.CPM.2018.5

Acknowledgements The author wishes to thank advisors Mihai Pop and Aravind Srinivasan for their support, guidance, and encouragement.

1 Introduction

String comparison is a fundamental problem in many fields such as bioinformatics and data compression. The difference between two strings is often measured by some notion of edit distance, the number of edit operations required to transform one string into another.

[1] Supported in part by NSF awards CCF-1422569 and CCF-1749864 as well as the NIH, grant R01-AI-100947 to Mihai Pop.

The classic Levenshtein distance definition includes insertion, deletion, and/or substitution operations on single characters. However, the more general edit distance with moves problem studied in [13] allows an additional operation wherein an entire block of text is shifted within a string.

Variations of these shift operations, also known as rearrangements, are commonly studied in genomics [31, 11] with several biologically motivated twists on the above definition. String comparison of DNA or protein sequences can provide an estimate of how closely related different species are. In data compression, we may want to store many similar strings as a single string along with the edits required to recover all strings. These two applications even overlap naturally in the field of bioinformatics where extremely large datasets of biological sequences are common. For example, the challenge of pan-genome storage is to store many highly similar sequences from the same clade such as a bacterial species.

One way to capture just the “moves” operation on two strings which are permutations of each other is the Minimum Common String Partition problem (MCSP). In that problem, we cut (partition) each string into a multi-set of substrings such that the two multi-sets are identical and the number of cuts is minimized. This paper studies the complementary problem to MCSP, the Maximum Duo-Preservation String Mapping Problem (MPSM) and its weighted variant (MWPSM). Our goal is to find a one-to-one mapping from the letters of one string to the other. The objective is to maximize the pairs of consecutive letters (duos) which map to pairs of consecutive letters in the other string (i.e. pairs that are not cut in MCSP).

While MCSP has been well-studied for some time, a recent flurry of work on MPSM has given us a deeper understanding of that problem. Mehrabi [28] introduced the Maximum Weight Duo-Preservation String Mapping Problem (MWPSM) to better capture applications in comparitive genomics. Beyond identifying the number of block moves, the weighted variant allows us to address questions like, "How far did these blocks move?" This better captures the concept of “synteny” in genetics [22, 29]. Also addressing practical considerations, Dudek, et al [16] included a quadratic time version of their approximation algorithm whereas much of the prior work has focused on improving the approximation in polynomial time.

1.1 Problem Description

The Maximum Duo-Preservation String Mapping Problem (MPSM) is defined as follows. We are given two strings $A = a_1a_2 \dots a_n$ and $B = b_1b_2 \dots b_n$ of length n such that B is a permutation of A. Let a_i and b_j be the i^{th} and j^{th} characters of their respective strings. A *proper* mapping π from A to B is a one-to-one mapping with $a_i = b_{\pi(i)}$ for all $i = 1, \dots, n$. A *duo* is simply two consecutive characters from the same string. We say that a duo (a_i, a_{i+1}) is *preserved* if a_i is mapped some b_j and a_{i+1} is mapped to b_{j+1}. The objective is to return a proper mapping from the letters of A to the letters of B which preserves the maximum number of duos. Note that the number of duos preserved in each string is identical and by convention we count the number of duos preserved in a single string rather than the sum over both strings. Let OPT_{MPSM} denote the number of duos preserved from a single string in an optimal solution to the MPSM problem. Figure 1 shows an example of an optimal mapping which preserves the maximum possible number of duos.

The complementary Minimum Common String Partition problem (MCSP) seeks to find partitions of the strings A and B where a partition P_A of A is defined as a set of substrings whose concatenation is A. The objective is to find minimum cardinality partitions P_A of A and P_B of B such that P_B is a permutation of P_A. Let OPT_{MCSP} denote the cardinality of a partition in an optimal solution. We can see that $OPT_{MCSP} = |P_A| = |P_B| = n - OPT_{MPSM}$.

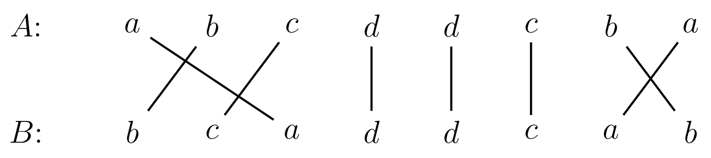

Figure 1 Illustration of a mapping π from A to B that preserves 3 duos: bc, dd, and dc. A solution to the complementary MCSP problem on the same strings would be partitions $P_A = a, bc, ddc, b, a$ and $P_B = bc, a, ddc, a, b$ with $|P_A| = |P_B| = 5$.

The variants, k-MPSM and k-MCSP, add the restriction that each letter occurs at most k times in each string. For a given algorithm, let ALG_{MPSM} be number of duos preserved by the algorithm. The approximation ratio for that algorithm is defined as OPT_{MPSM}/ALG_{MPSM}.

In MWPSM, a weight is assigned to every pair of preservable duos and we seek to maximize the weight of the solution. While [28], discusses using weights to capture the positions of preserved duos within their respective strings, the weights in MWPSM can be arbitrary and are not required to be a function of position.

1.2 Related Work

The Maximum Duo-Preservation String Mapping Problem (MPSM) was introduced in [10] along with the related Constrained Maximum Induced Subgraph (CMIS) and Constrained Minimum Induced Subgraph (CNIS) problems. They used a linear programming and randomized rounding approach to approximate the k-CMIS problem which they show is a generalization of k-MPSM. This led to a k^2-approximation for $k \geq 3$ and a 2-approximation for $k = 2$. This was improved by [6] to a 4-approximation independent of k and running in $O(n^{3/2})$ time as well as approximation ratios of 3 for $k = 3$ and 8/5 for $k = 2$. [6] also showed that k-MPSM is APX-hard even for $k = 2$, meaning no polynomial-time approximation scheme (PTAS) exists assuming $P \neq NP$. The approximation was subsequently improved to 3.5 using local search [5], 3.25 using a combinatorial triplet matching approach [7], and finally $2 + \epsilon$ for any $\epsilon > 0$ in $n^{O(1/\epsilon)}$ time, again using local search [16]. The work of [16] also presented a 2.67-approximation running in $O(n^2)$ time.

The recent interest in MPSM led to the study of several variants including Maximum Weight Duo-preservation String Mapping (MWPSM), k-MPSM, and fixed-parameter tractability (FPT). The weighted variant of MPSM was introduced in [28] along with an algorithm achieving a 6-approximation. That work was the first to apply the local ratio technique developed by Bar-Yehuda and Even [2] to an MPSM problem. Recent work on k-MPSM led to a $(1.4 + \epsilon)$-approximation for 2-MPSM [32]. On the FPT side, [3] showed that MPSM is fixed-parameter tractable when parameterized by the number of preserved duos and [27] achieved a faster running time with a randomized algorithm.

The Minimum Common String Partition problem (MCSP) has been extensively studied from many angles including polynomial-time approximation [10, 12, 13, 20, 26, 25], fixed-parameter tractability [8, 14, 23, 9], and heuristics [17, 4, 18]. FPT algorithms have been parameterized by maximum number of times any character occurs, minimum block size, and the size of the optimal minimum partition. Heuristic approaches range from an ant colony optimization algorithm [17] to integer linear programming (ILP) based strategies [4, 18] which in some cases solve the problem optimally for strings up to 2,000 characters in length.

The problem was shown to be NP-hard (thus implying MPSM is also NP-hard) and APX-hard even for 2-MCSP [20]. The current best approximations are an $O(\log n \log^* n)$-

approximation due to [13] for general MCSP bases on the related edit distance with moves problem and an $O(k)$-approximation for k-MCSP due to [26]. Applications to evolutionary distance and genome rearrangement can be found in [31, 11].

Unclaimed results in prior work: An analysis of prior work shows that 4-approximations to both problems studied here can be achieved using slight modifications to existing work. For MWPSM, the algorithm in [6] can be extended by choosing a maximum weight matching and partition rather than maximum cardinality. For the unweighted problem, Goldstein and Lewenstein [21] showed an $O(n)$ time greedy algorithm for MCSP. Although not discussed in their paper which pre-dated MPSM, we note that the greedy algorithm for MCSP achieves a 4-approximation for MPSM by a fairly straightforward charging argument. Formal proofs of these claims are outside the scope of this paper and we leave them to the interested reader. Additionally, we will not refer to these approximations when comparing our work to previous best known results. We simply mention them here for completeness and to give a nod to two nice papers in the area.

1.3 Our Contributions

We show a transformation of the Maximum Duo-Preservation String Mapping (MPSM) problem into a related tractable problem. This transformation leads to new algorithms for both weighted and unweighted MPSM. For the weighted case, we present an 8/3-approximation running in $O(n^3)$ time. This improves upon the previous best 6-approximation in polynomial time [28] (a tighter bound on the running time is not given in the paper). It also matches the best quadratic time approximation for the unweighted problem of 2.67 and approaches the best unweighted approximation of $2 + \epsilon$ for any $\epsilon > 0$ in $n^{O(1/\epsilon)}$ time, both due to [16]. We further show in Corollary 2 that we can improve the running time at the cost of a weaker approximation. For the unweighted case, we present the first linear time algorithm with an 8/3-approximation again matching the previous best quadratic time algorithm and coming fairly close to the best known $(2 + \epsilon)$-approximation achieved by a significantly larger running time. In particular, the move from quadratic to linear time in length of the strings is significant for practical settings wherein the string length may be long enough that quadratic time is prohibitive. Finally, we introduce the first streaming algorithm for MPSM in the streaming model where each string is read one character at a time. We show that a single pass suffices to find a 4-approximation on the size of an optimal solution using only $O(\alpha^2 \lg n)$ space.

In addition, the techniques here are novel to this problem and may inspire future improvements. While [7] also used a form of triplet matching, the structure of the triplet matching is different as is the approach to achieving a feasible solution to MPSM. Our main results are summarized in the theorems below.

▶ **Theorem 1.** *There exists an algorithm which finds an $8/3$-approximation to MWPSM on strings of length n in $O(n^3)$ time.*

▶ **Corollary 2.** *Using the approximate weighted matching algorithm of [15], we can find an $8/(3(1-\epsilon))$-approximation to MWPSM on strings of length n for any $\epsilon > 0$ in $O(n^2\epsilon^{-1} \lg \epsilon^{-1})$ time.*

▶ **Theorem 3.** *There exists an algorithm which finds an $8/3$-approximation to MPSM on strings of length n over alphabets of size α in $O(n + \alpha^7)$ time.*

▶ **Corollary 4.** *There exists an algorithm which finds an* $8/3$*-approximation to MPSM on strings of length* n *over constant-sized alphabets in* $O(n)$ *time.*

▶ **Theorem 5.** *There exists a single-pass streaming algorithm which finds a* 4*-approximation to the size of an MPSM on strings of length* n *over alphabets of size* α *using only* $O(\alpha^2 \lg n)$ *space.*

1.4 Preliminaries

Let $A = a_1 a_2 \dots a_n$ and $B = b_1 b_2 \dots b_n$ be two strings of length n with a_i and b_j being the i^{th} and j^{th} characters of their respective strings. A *duo* $D_i^A = (a_i, a_{i+1})$ contains a pair of consecutive characters a_i and a_{i+1}. We use $D^A = (D_1^A, \dots, D_{n-1}^A)$ and $D^B = (D_1^B, \dots, D_{n-1}^B)$ to denote the sets of *duos* for A and B, respectively. We similarly define a *triplet* $T_i^A = (a_i, a_{i+1}, a_{i+2})$ as a set of three consecutive characters a_i, a_{i+1}, and a_{i+2} in the string and sets of *triplets* $T^A = (T_1^A, \dots, T_{n-2}^A)$ and $T^B = (T_1^B, \dots, T_{n-2}^B)$ for strings A and B, respectively. Observe that the duos D_i^A and D_{i+1}^A correspond to the first two and last two characters, respectively, of the triplet T_i^A. We refer to duos D_i^A and D_{i+1}^A as *subsets* of the triplet T_i^A.

A proper mapping π from A to B is a one-to-one mapping from the letters of A to the letters of B with $a_i = b_{\pi(i)}$ for all $i = 1, \dots, n$. Recall that a duo (a_i, a_{i+1}) is preserved if and only if a_i is mapped to some b_j and a_{i+1} is mapped to b_{j+1}. We call a pair of duos (D_i^A, D_j^B) *preservable* if and only if $a_i = b_j$ and $a_{i+1} = b_{j+1}$. For MWPSM, let $w(D_i^A, D_j^B)$ be the weight gained by mapping D_i^A to D_j^B.

For consistency, we define the concept of conflicting pairs of duos using the terminology of [6]. Two preservable pairs of duos (D_i^A, D_j^B) and (D_h^A, D_ℓ^B) are said to be *conflicting* if no proper mapping can preserve both of them. These conflicts can be of two types type 1 and type 2. In *type 1 conflicts*, either $i = h \wedge j \neq \ell$ or $i \neq h \wedge j = \ell$. In *type 2 conflicts*, either $i = h + 1 \wedge j \neq \ell + 1$ or $i \neq h + 1 \wedge j = \ell + 1$.

The algorithms here only show how to map the characters of the preserved duos. In all cases, note that any unmapped characters can be mapped arbitrarily to identical characters in the other string in linear time.

2 Main techniques and algorithm for MWPSM

For both algorithms, we first solve a weighted bipartite matching problem we call Alternating Triplet Matching (ATM). In this section, we define ATM, show that a solution to ATM has weight at least 3/4 of an optimal solution to MWPSM, and finally show that we can convert a solution to ATM to a feasible duo mapping while preserving 1/2 of its weight. Combining these facts leads to an 8/3-approximation to MWPSM.

2.1 The Alternating Triplet Matching (ATM) problem

Here, we define this problem in terms of MWPSM. Modifications for the unweighted variant (to admit a faster solution) will be defined in Section 3. Let $T^{A'} = \{T_i^A \mid i \text{ is odd}\}$, $T^{B'} = \{T_i^B \mid i \text{ is odd}\}$ and $T^{B''} = \{T_i^B \mid i \text{ is even}\}$. Throughout the paper, we refer to triplets starting at odd indices in their respective strings as *odd triplets* and similarly use the term *even triplets*. Note, we do not use the even triplets from A.

Using these subsets, we formulate bipartite matching problems on two separate graphs $G' = \{T^{A'}, T^{B'}, E'\}$ and $G'' = \{T^{A'}, T^{B''}, E''\}$. The edges of G' depend on the letters in the triplets. Consider triplets $T_i^{A'} = (D_i^A, D_{i+1}^A)$ and $T_j^{B'} = (D_j^B, D_{j+1}^B)$. For each pair of duos

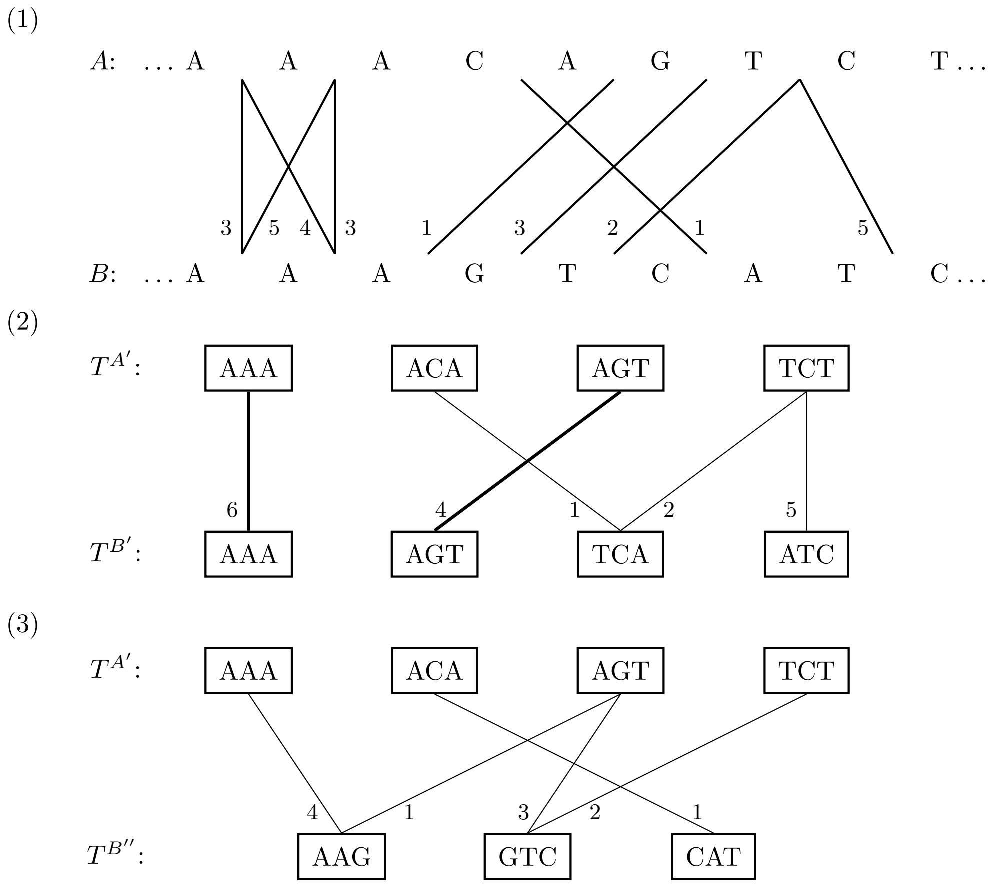

Figure 2 Illustration of how to generate an ATM instance from an MWPSM instance. (1) Substrings of the original two strings, A and B, starting at some odd index and featuring weighted edges representing the weight of preserving a pair of duos. (2) The graph G' with thicker edges representing an exact match between two triplets. In the case of multiple edges between a pair of triplets (e.g. the five edges between the "AAA" triplets), we only show the heaviest weight edge. (3) The graph G''. Note that that the weight of a mapping which maps the two "AGTC" strings to each other is 6, which can be achieved by a matching in G', but not in G''.

D_h^A and D_ℓ^B with $h \in \{i, i+1\}$, $\ell \in \{j, j+1\}$, and $D_h^A = D_\ell^B$, we add an edge $e = (T_i^{A'}, T_j^{B'})$ with weight $w(e) = w(D_h^A, D_\ell^B)$. Additionally, if $T_i^{A'} = T_j^{B'}$, we add an edge $e = (T_i^{A'}, T_j^{B'})$ between them with weight $w(e) = w(D_i^A, D_j^B) + w(D_{i+1}^A, D_{j+1}^B)$. In other words, the edge gets the combined weight of the duo pairs preserved by mapping the substring $T_i^{A'}$ to the substring $T_j^{B'}$. The graph G'' is defined similarly. There could be up to five edges total if the triplets contain one letter repeated (e.g. "AAA"). In the case of multiple edges between a pair of triplets, we only need to consider the heaviest edge among them since each triplet can be matched at most once. However, we keep all edges for the sake of simplifying some of the proofs. Figure 2 illustrates the procedure of generating an ATM instance.

2.2 MWPSM algorithm and analysis

Let $OPT_{G'}$ and $OPT_{G''}$ be the weights of maximum weight matchings in G' and G'', respectively. Note that we can find these matchings in the time it takes to compute maximum weight bipartite matching. Since our graphs have $O(n)$ vertices and could have $O(n^2)$ edges, this takes $O(n^2 \lg n + n \cdot n^2) = O(n^3)$ time [19]. Lemma 6 states that either $OPT_{G'}$ or $OPT_{G''}$ will be a (3/4)-approximation to the weight of an optimal solution to MWPSM, OPT_{MWPSM}. Let $OPT_{ATM} = \max(OPT_{G'}, OPT_{G''})$.

▶ **Lemma 6.** $OPT_{ATM} \geq (3/4)OPT_{MWPSM}$.

Proof. We divide the edges of OPT_{MWPSM} into two partitions. The first partition, P^{same}, includes mappings, in which both letters occur at odd indices or both letters occur at even indices. The second partition, P^{diff}, includes the remaining mappings wherein one letter is at an odd index and the other is at an even index (this could be odd from A, even from B or even from A, odd from B).

Note that the mapping of each preserved pair of duos (D_i^A, D_j^B) will be contained in one of these two partitions. Without loss of generality, let the weight of P^{same} be at least the weight of P^{diff}. We show how to transform OPT_{MWPSM} into a feasible bipartite matching in G' while retaining the full weight of P^{same} and at least half of the weight of P^{diff}. Thus, we retain at least 3/4 of the weight of OPT_{MWPSM}.

For each triplet in the vertex set of G' that contains one or two preserved duos from P^{same}, we can add an edge to our matching with weight equal to the weight of the preserved duos. This works because consecutive pairs of preserved duos (D_i^A, D_j^B) and (D_{i+1}^A, D_{j+1}^B) with i and j both being odd will correspond to a "double" edge in the ATM instance with weight equal to $w(D_i^A, D_j^B) + w(D_{i+1}^A, D_{j+1}^B)$. On the other hand, if i and j are both even, then the duos of (D_i^A, D_j^B) and (D_{i+1}^A, D_{j+1}^B) are contained in four different triplets and will be added separately. Thus, we can maintain all of the weight of P^{same} in a matching in G'.

A slightly trickier case arises with P^{diff}. Any consecutive pairs of preserved duos (D_i^A, D_j^B) and (D_{i+1}^A, D_{j+1}^B) in P^{diff} will have i and j of different parity. This results in the duos being contained in three triplets, two from one partition and one from the other. That means the edges in the ATM instance capturing the weights of the two pairs will be conflicting. Thus we can only preserve the weight of one of the two pairs in our ATM solution. To guarantee that we add at least half of the weight of P^{diff} to our solution, we further partition it into pairs (D_i^A, D_j^B) with i being odd and those with i being even. Then we simply choose the heavier of those two partitions to add to our ATM solution.

For the case where P^{diff} is heavier than P^{same}, we can do a similar construction for G''. Thus, our ATM solution in either G' or G'' could have at least 3/4 the weight of an optimal solution to MWPSM. ◀

We can now show how to transform an optimal solution to ATM (the heavier of the two matchings) into a feasible string mapping which preserves at least half of the weight of the ATM solution. Let $G = (D^A, D^B, E)$ be a bipartite graph on the duos of A and B with edge weights equal to the weight of preserving each pair of duos. We first show how to convert an ATM solution into a matching M in G. Then, we show how to resolve conflicts of type 2 (conflicts of type 1 will not arise since M is a matching).

The transformation is simply a reversal of how we constructed the ATM graphs. For each edge between triplets in our ATM solution (the heavier of the two matchings in G' and G''), we add an edge or edges to M corresponding to the duos that "created" that triplet edge.

To resolve conflicts, we consider the conflict graph C wherein we have a node for each edge in M and an arc between nodes if their corresponding edges are in conflict. We can prove that C has maximum degree 2, meaning it will be a collection of paths and cycles. Further, we note that each cycle will have even length due to Lemma 7 and the fact that the underlying graph is bipartite. Thus, for each path or cycle, we choose the heavier of the two maximal independent sets in that path or cycle to add to our final MPSM solution. Lemma 7 establishes that C has maximum degree 2.

▶ **Lemma 7.** *Each edge in M conflicts with at most one other edge at each endpoint.*

Proof. First, we note that each duo is contained in at most one triplet edge from the ATM solution and therefore can only be matched once in M. In other words, M is a classical matching in the bipartite graph of duos. This follows from the fact that consecutive triplets in a string starting at only odd (or only even) indices will overlap at exactly one letter.

This ensures that no conflicts of type 1 can arise since that would require a duo to be matched twice. We can also show that at most one conflict of type 2 arises at each endpoint. Without loss of generality, consider the endpoint D_i^A. Consider the duos D_{i-1}^A and D_{i+1}^A where such a conflict might arise. Notice that one of these duos must have come from the same triplet as D_i^A, while the other comes from a different triplet. The duo from the same triplet will either be unmatched or matched as a non-conflicting parallel edge. Thus no conflict arises from that duo. The duo from a different triplet could contribute at most one conflicting edge by the above claim that each duo is matched at most once. Applying this argument to both endpoints of a given edge completes the proof. ◀

▶ **Lemma 8.** *M can be converted into M', a feasible solution to MWPSM, such that the weight of M' is at least $(1/2)OPT_{ATM} \geq (3/8)OPT_{MWPSM}$.*

Proof. The conflict graph on the edges of M must be a collection of paths and even length cycles since it has maximum degree 2 and G is bipartite. We can simply decompose each path or cycle into two independent sets and choose the heavier of the two. This operation discards at most half of the weight of M while removing all conflicts and leaving us with a feasible solution to MWPSM. ◀

The proofs of Theorem 1 and Corollary 2 follow from the preceding lemmas.

3 Linear time algorithm for unweighted MPSM

The basic approach follows roughly the same steps as the weighted algorithm from Section 2: construct an ATM instance, solve the matching problem, transform the solution into a duo matching on the strings, and resolve conflicts. We show that with a small modification, each step can be done in linear time for the unweighted problem. The key insight that allows for this speedup is that identical triplets can be collapsed into single vertices and we can solve a b-matching problem we call b-ATM. In the *b-matching* variant of classical matching, each vertex in the graph has a capacity and can be matched that many times. We will abuse notation a bit and refer to each vertex as having capacity b, although we actually allow the capacity of each node to be different. The following subsections illustrate how to perform the aforementioned steps and bound the running time of each step.

3.1 Constructing the b-ATM instance in $O(n + \alpha^4)$ time

We construct a triplet matching problem as in Section 2.1 with one crucial adjustment: identical triplets are collapsed into single vertices with capacity equal to the number of occurrences of that triplet in its given set ($T^{A'}$, $T^{B'}$, or $T^{B''}$). The number of times each vertex is allowed to be matched is equal to its capacity. Similarly, each edge can be matched multiple times up to the smaller capacity among its two endpoints. Algorithm 1 shows how to construct a b-ATM instance from the two input strings in linear time.

Algorithm 1: CONSTRUCT B-ATM

```
1 Traverse each string to build a set of triplets with counts for $A'$, $B'$, and $B''$.
2 For $G'$ and $G''$, create a vertex for each triplet with capacity equal to its count. Add
    edges between the triplets as in Section 2.1 with the following modification. If two
    triplets match exactly, give the edge weight 2 and if they only share a duo in
    common, give the edge weight 1.
```

Algorithm 2: SOLVE B-ATM

```
1 Add each edge with weight 2, corresponding to two identical triplets, to the matching.
2 Find a maximum b-matching in the remaining "unweighted" graph using maximum
    flow techniques.
```

As in Section 2.1, let $OPT_{G'}$ and $OPT_{G''}$ be the weights of maximum weight b-matchings in G' and G'', respectively. Lemma 9 states that either $OPT_{G'}$ or $OPT_{G''}$ will be a (3/4)-approximation to the size of an optimal solution to MPSM, OPT_{MPSM}. Let $OPT_{b\text{-}ATM} = \max(OPT_{G'}, OPT_{G''})$ as constructed by Algorithm 1.

▶ **Lemma 9.** $OPT_{b\text{-}ATM} \geq (3/4)OPT_{MPSM}$.

Proof. This proof follows from Lemma 6. Suppose we constructed an ATM instance as in Section 2.1, but for the unweighted problem. By Lemma 6, we would have $OPT_{ATM} \geq (3/4)OPT_{MPSM}$. Now note that we can collapse all identical triplet vertices in each partition of OPT_{ATM} to get a feasible solution to the b-ATM problem without reducing the weight. ◀

▶ **Lemma 10.** *Algorithm 1 constructs a graph with $O(\alpha^3)$ vertices and $O(\alpha^4)$ edges in $O(n + \alpha^4)$ time.*

Proof. Step 1 of the algorithm clearly runs in less than $O(n + \alpha^4)$ time. It simply traverses each string once, storing the triplets in some appropriate data structure with constant insert and query time.

To bound the running time of step 2, we first bound the number of edges created. Note that the bipartite graph of b-ATM has $O(\alpha^3)$ vertices in each partition since that is the maximum number of 3-mers in an alphabet of size α. To bound the edge set, notice that for any 3-mer, there exist at most 4α other 3-mers with a substring of length 2 in common. Thus, the max degree of each node is $O(\alpha)$ and the size of the edge set E is at most $O(\alpha^4)$. When adding edges, we can check for the existence of each edge in constant time, again assuming the triplet are stored in some appropriate data structure. ◀

3.2 Solving b-ATM quickly

Algorithm 2 shows how to solve b-ATM within our time constraints. Lemma 11 proves the correctness of this algorithm while Lemma 12 bounds its running time.

▶ **Lemma 11.** *Algorithm 2 finds a maximum weight b-matching in the b-ATM instance.*

Proof. Here, we need to justify Step 1 of Algorithm 2 by showing that there always exists some maximum b-matching which contains all of the edges corresponding to identical pairs of triplets. First note that it is feasible to include all such edges since they can never conflict with each other. For each triplet in one partition, there is at most one identical triplet in the other partition.

```
Algorithm 3: TRANSFORM b-ATM TO MPSM
1 Assign each copy of a 3-mer and its edge from the b-ATM solution to a triplet from
    the original strings to get an ATM solution.
2 Transform the ATM solution into a duo matching as detailed in Section 2.2
3 Resolve conflicts by traversing the paths/cycles of the conflict graph and discarding
    every other edge.
```

We apply the following claim iteratively to complete the proof. Given a maximum weight b-matching M which does not include all identical pair edges, we can always add one such edge without decreasing the weight of the solution. Consider an arbitrary identical pair edge e that is not in M. To add e to M we need to remove at most two edges from M, one for each endpoint of e. Since e has a weight of 2 while the removed edges have weights of 1 each, swapping those edges for e will not reduce the weight of the solution. ◀

▶ **Lemma 12.** *Algorithm 2 runs in $O(n)$ time plus the time to compute an unweighted maximum b-matching on a graph with $O(\alpha^3)$ vertices and $O(\alpha^4)$ edges and total capacity $O(n)$. Using current maximum flow algorithms, Algorithm 2 can run in $O(n + \alpha^7)$ time.*

Proof. If the graph were unweighted, we could find a maximum b-matching in $O(|V||E|) = O(\alpha^7)$ time using the maximum flow approach in [30]. Fortunately, by Lemma 11, we can first add all edges with weight 2 to our solution. Thus, we are left with an "unweighted" residual problem that can be solved using a maximum flow algorithm. ◀

3.3 Transforming the b-ATM solution to a duo matching and resolving conflicts

Now that we have solved our b-ATM problem we need transform it back to a duo matching. The obvious challenge here is that each b-ATM vertex represents roughly b copies of a given 3-mer that must each be assigned to a triplet in the original string in linear time while preserving the weight of the b-ATM solution. There are $b!$ such assignments and b could be on the order of n. However, the important observation here is that we can do this *arbitrarily* and still preserve the size of the b-ATM solution.

▶ **Lemma 13.** *Algorithm 3 constructs a feasible solution to MPSM with size equal to half the weight of $OPT_{b\text{-}ATM}$.*

Proof. The proof follows from Lemma 8. Notice that we assign exactly one copy of a 3-mer to each triplet and the result is a feasible solution to the ATM problem. ◀

▶ **Lemma 14.** *Algorithm 3 runs in $O(n)$ time.*

Proof. Assigning each copy of a 3-mer and its edge to a triplet can be done in constant time if we maintain lists of the indices at which each 3-mer occurs in each string, resulting in $O(n)$ time overall. Similarly, generating the duo-matching can easily be done in $O(n)$ time. Resolving conflicts in the unweighted problem involves traversing $O(n)$ edges and removing every other one which can be done in $O(n)$ time as well. ◀

The proofs of Theorem 3 and Corollary 4 follow from the preceding lemmas.

4 A streaming algorithm for MPSM

We observe that the algorithm of [6] can be adapted into a single-pass streaming algorithm in the streaming model where each string is read one character at a time. We present an algorithm using $O(\alpha^2 \lg n)$ space and giving a 4-approximation of the size of an MPSM solution without providing an explicit mapping. In [6], they upper bound MPSM by a maximum matching in the duo graph. Then they show that a feasible MPSM solution can be found while preserving at least $1/4$ of the edges in the matching.

The algorithm is simple. Maintain a counter for each 2-mer in the alphabet and a counter for the size of the matching. While processing the first string, count the number of occurrences of each 2-mer. For the second string, each time you encounter a duo with a nonzero count, decrease its count by 1 and increase the size of the matching by 1. At the end, divide the size of the matching by 4 to get a 4-approximation to the size of the optimal MPSM. The following Lemmas establish the space-efficiency and correctness of the the algorithm.

▶ **Lemma 15.** *The streaming algorithm uses only $O(\alpha^2 \lg n)$ space where α is the alphabet size and n is the length of the strings.*

Proof. The number of 2-mers from an alphabet of size α is α^2. We require only $O(\lg n)$ bits of space for each 2-mer counter since no 2-mer could appear more than $O(n)$ times where n is the length of the strings. Similarly, we keep just one counter for the size of the matching which requires only $O(\lg n)$ bits of space since the size of the matching is at most n. In addition to the counters, we must store the previously seen letter since our streaming model involves reading one character at a time, but we are counting duos. However, this only requires $O(\lg \alpha)$ space. ◀

▶ **Lemma 16.** *The streaming algorithm achieves a 4-approximation to MPSM.*

Proof. We first show that the size of a maximum matching in a bipartite duo graph G as defined in [6] is equal to the sum of the minimum number of occurrences of each duo among the two strings. Notice that G can be decomposed into a set of connected components for each 2-mer since each vertex only has edges to other vertices corresponding to the same 2-mer. Further, each of these connected components is a complete bipartite graph with maximum matching size equal to the minimum size of the two partitions.

Thus, computing the above sum gives us the size of the maximum matching. We note that the number of times the matching size counter increase due to vertices of a given 2-mer is exactly equal to the minimum number of times that 2-mer appears in either of the two strings.

Finally, as shown in [6], a maximum matching in the duo graph is an upper bound on the optimal solution to MPSM and can always be converted into a feasible MPSM solution while preserving at least $1/4$ of its size. ◀

The proof of Theorem 5 follows from Lemmas 15 and 16.

5 Conclusion and future directions

We showed a transformation of the Maximum Duo-Preservation String Mapping (MPSM) problem into a related tractable problem. This led to new algorithms for both MWPSM and MPSM. For the weighted case, we presented a tighter approximation closing in on the best unweighted result using a reasonably fast algorithm. We also showed that the running time could be improved at the expense of a slightly weaker approximation. For the unweighted case, we presented the first linear time algorithm with an approximation matching the previous best quadratic time algorithm and fairly close to the best known approximation achieved by a significantly larger running time. Finally, we presented the first streaming algorithm for MPSM showing that a constant approximation is achievable in the single-pass streaming model.

We believe the most pressing future direction is to explore the applications and utility of this problem further. The complementary relationship with Minimum Common String Partition (MCSP) has driven much of the current interest in MPSM. However, given their relationship, new approximations for MPSM do not directly lead to any improvements for MSCP. It is reasonable to ask if the study of MPSM can teach us anything about MCSP or at least inspire new heuristics. We note that some current linear-time algorithms for MCSP are greedy algorithms [21] with a proven lower bound of $\Omega(n^{0.46})$ [24] (Although this bound arises from carefully constructed strings over a $(\log n)$-sized alphabet). This is in contrast to the best known approximation for MCSP, $O(\log n \log^* n)$ [13]. Perhaps the linear time MPSM algorithm presented here could be combined with greedy approaches leading to better, more robust heuristics. Further, since MPSM currently appears to be "easier" than MCSP, it would be fruitful to explore more applications for MPSM itself in bioinformatics, data compression, and beyond.

On the theoretical side, the biggest questions revolve around the factor of 2 approximation. Is this tight for MPSM conditioned on some hardness conjecture or can we do better? It surely seems like a natural bound. Regardless, can we achieve a 2-approximation in linear time? Likewise, for MWPSM, a 2-approximation could be seen as the next major goal. All of this seems within reach, using existing ideas or different tools such as LP rounding techniques. Another direction would be to add edit operations. It seems that MWPSM could be adapted to handle the cost of substitutions. However, this is nontrivial since existing algorithms assume that letters which do not belong to preserved duos can be mapped at no penalty.

Finally, we propose variants of MWPSM that may admit a faster approximation than we see in this paper. Suppose the weights are not arbitrary, but follow some "rules". [28] suggested the weight of a duo-preservation could be a function of the "closeness" of the mapping in terms of the positions of the characters in their respective strings. However, [28] and this paper consider only arbitrary weights. One could imagine a weight function like $w(D_i^A, D_j^B) = n - |i - j|$ that does not require us to examine every edge in the duo graph. Of course, the function need not be so naive as any metric or geometric weight functions admit faster matching algorithms [1].

References

1 Pankaj K. Agarwal and R. Sharathkumar. Approximation algorithms for bipartite matching with metric and geometric costs. In *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 555–564, New York, NY, USA, 2014. ACM.

2 R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. In G. Ausiello and M. Lucertini, editors, *Analysis and Design of Algorithms for Combinatorial Problems*, volume 109 of *North-Holland Mathematics Studies*, pages 27–45. North-Holland, 1985.

3 Stefano Beretta, Mauro Castelli, and Riccardo Dondi. Parameterized tractability of the maximum-duo preservation string mapping problem. *CoRR*, abs/1512.03220, 2015. `arXiv:1512.03220`.

4 Christian Blum, José A. Lozano, and Pinacho Davidson. Mathematical programming strategies for solving the minimum common string partition problem. *European Journal of Operational Research*, 242(3):769–777, 2015.

5 Nicolas Boria, Gianpiero Cabodi, Paolo Camurati, Marco Palena, Paolo Pasini, and Stefano Quer. A 7/2-approximation algorithm for the maximum duo-preservation string mapping problem. In Roberto Grossi and Moshe Lewenstein, editors, *27th Annual Symposium on Combinatorial Pattern Matching (CPM 2016)*, volume 54 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:8, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

6 Nicolas Boria, Adam Kurpisz, Samuli Leppänen, and Monaldo Mastrolilli. Improved approximation for the maximum duo-preservation string mapping problem. In *Algorithms in Bioinformatics: 14th International Workshop, WABI 2014, Wroclaw, Poland, September 8-10, 2014. Proceedings*, pages 14–25, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

7 Brian Brubach. Further improvement in approximating the maximum duo-preservation string mapping problem. In Martin Frith and Christian Nørgaard Storm Pedersen, editors, *Algorithms in Bioinformatics*, pages 52–64, Cham, 2016. Springer International Publishing.

8 Laurent Bulteau, Guillaume Fertin, Christian Komusiewicz, and Irena Rusu. A fixed-parameter algorithm for minimum common string partition with few duplications. In Aaron Darling and Jens Stoye, editors, *Algorithms in Bioinformatics: 13th International Workshop, WABI 2013, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, pages 244–258, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

9 Laurent Bulteau and Christian Komusiewicz. Minimum common string partition parameterized by partition size is fixed-parameter tractable. In *Proceedings of the Twenty-fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 102–121, Philadelphia, PA, USA, 2014. Society for Industrial and Applied Mathematics.

10 Wenbin Chen, Zhengzhang Chen, Nagiza F. Samatova, Lingxi Peng, Jianxiong Wang, and Maobin Tang. Solving the maximum duo-preservation string mapping problem with linear programming. *Theoretical Computer Science*, 530:1–11, 2014.

11 Xin Chen, Jie Zheng, Zheng Fu, Peng Nan, Yang Zhong, S. Lonardi, and Tao Jiang. Assignment of orthologous genes via genome rearrangement. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2(4):302–315, Oct 2005.

12 Marek Chrobak, Petr Kolman, and Jiří Sgall. The greedy algorithm for the minimum common string partition problem. In Klaus Jansen, Sanjeev Khanna, José D. P. Rolim, and Dana Ron, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques: 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2004, and 8th International Workshop on Randomization and Computation, RANDOM 2004, Cambridge, MA, USA, August 22-24, 2004. Proceedings*, pages 84–95, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

13 Graham Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. *ACM Trans. Algorithms*, 3(1):2:1–2:19, 2007.

14 Peter Damaschke. Minimum common string partition parameterized. In Keith A. Crandall and Jens Lagergren, editors, *Algorithms in Bioinformatics: 8th International Workshop, WABI 2008, Karlsruhe, Germany, September 15-19, 2008. Proceedings*, pages 87–98, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

15 Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *J. ACM*, 61(1):1:1–1:23, 2014.

16 Bartlomiej Dudek, Pawel Gawrychowski, and Piotr Ostropolski-Nalewaja. A family of approximation algorithms for the maximum duo-preservation string mapping problem. In Juha Kärkkäinen, Jakub Radoszewski, and Wojciech Rytter, editors, *28th Annual Symposium on Combinatorial Pattern Matching (CPM 2017)*, volume 78 of *Leibniz Interna-*

tional Proceedings in Informatics (LIPIcs), pages 10:1–10:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

17 S. M. Ferdous and M. Sohel Rahman. Solving the minimum common string partition problem with the help of ants. In Ying Tan, Yuhui Shi, and Hongwei Mo, editors, *Advances in Swarm Intelligence: 4th International Conference, ICSI 2013, Harbin, China, June 12-15, 2013, Proceedings, Part I*, pages 306–313, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

18 S. M. Ferdous and M. Sohel Rahman. An integer programming formulation of the minimum common string partition problem. *PLoS ONE*, 10(7):1–16, 07 2015.

19 Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, jul 1987.

20 Avraham Goldstein, Petr Kolman, and Jie Zheng. Minimum common string partition problem: Hardness and approximations. In *Proceedings of the 15th International Conference on Algorithms and Computation*, ISAAC'04, pages 484–495, Berlin, Heidelberg, 2004. Springer-Verlag.

21 Isaac Goldstein and Moshe Lewenstein. Quick greedy computation for minimum common string partition. *Theor. Comput. Sci.*, 542:98–107, 2014.

22 RC Hardison. Comparative genomics. *PLoS Biol*, 1(2):e58, 2003.

23 Haitao Jiang, Binhai Zhu, Daming Zhu, and Hong Zhu. Minimum common string partition revisited. *Journal of Combinatorial Optimization*, 23(4):519–527, 2012.

24 Haim Kaplan and Nira Shafrir. The greedy algorithm for edit distance with moves. *Information Processing Letters*, 97(1):23–27, 2006.

25 Petr Kolman and Tomasz Waleń. Approximating reversal distance for strings with bounded number of duplicates. *Discrete Applied Mathematics*, 155(3):327–336, 2007.

26 Petr Kolman and Tomasz Waleń. Reversal distance for strings with duplicates: Linear time approximation using hitting set. In Thomas Erlebach and Christos Kaklamanis, editors, *Approximation and Online Algorithms: 4th International Workshop, WAOA 2006, Zurich, Switzerland, September 14-15, 2006. Revised Papers*, pages 279–289, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

27 Christian Komusiewicz, Mateus de Oliveira Oliveira, and Meirav Zehavi. Revisiting the parameterized complexity of maximum-duo preservation string mapping. In Juha Kärkkäinen, Jakub Radoszewski, and Wojciech Rytter, editors, *28th Annual Symposium on Combinatorial Pattern Matching (CPM 2017)*, volume 78 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:17, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

28 Saeed Mehrabi. Approximating weighted duo-preservation in comparative genomics. In Yixin Cao and Jianer Chen, editors, *Computing and Combinatorics*, pages 396–406, Cham, 2017. Springer International Publishing.

29 A. R. Mushegian. *Foundations of Comparative Genomics.* Elsevier, 1 2007.

30 James B. Orlin. Max flows in O(nm) time, or better. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 765–774, New York, NY, USA, 2013. ACM.

31 Krister M. Swenson, Mark Marron, Joel V. Earnest-Deyoung, and Bernard M. E. Moret. Approximating the true evolutionary distance between two genomes. *J. Exp. Algorithmics*, 12:3.5:1–3.5:17, 2008.

32 Yao Xu, Yong Chen, Guohui Lin, Tian Liu, Taibo Luo, and Peng Zhang. A (1.4 + epsilon)-approximation algorithm for the 2-max-duo problem. In Yoshio Okamoto and Takeshi Tokuyama, editors, *28th International Symposium on Algorithms and Computation (ISAAC 2017)*, volume 92 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 66:1–66:12, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.