Efficient assembly consensus algorithms for divergent contig sets

Annie Chateau, Tom Davot, Manuel Lafond

Efficient Assembly Consensus Algorithms for Divergent Contig Sets

Annie Chateau[1], Tom Davot[1], Manuel Lafond[2]

[1] LIRMM - CNRS UMR 5506 - Montpellier, France

[2] Université de Sherbrooke, Canada, {chateau,davot}@lirmm.fr,manuel.lafond@USherbrooke.ca

Abstract. Assembly is a fundamental task in genome sequencing, and many assemblers have been made available in the last decade. Because of the wide range of possible choices, it can be hard to determine which tool or parameter to use for a specific genome sequencing project. In this paper, we propose a consensus approach that takes the best parts of several contigs datasets produced by different methods, and combines them into a better assembly. This amounts to orienting and ordering sets of contigs, which can be viewed as an optimization problem where the aim is to find an alignment of two fragmented strings that maximizes an arbitrary scoring function between matched characters. In this work, we investigate the computational complexity of this problem. We first show that it is NP-hard, even in an alphabet with only two symbols and with all scores being either 0 or 1. On the positive side, we propose an efficient, quadratic time algorithm that achieves approximation factor 3.

Keywords: B.2.4.a Algorithms, F.2 Analysis of Algorithms and Problem Complexity, L.3.0.j. Molecular biology [3]

1 Introduction

Assembling genomes is a notorious problem in reconstructing the genetic content of species, a fundamental task in understanding the biology of present and past organisms. Whole genome assembly has already been difficult to address in the last decades, but recent trends in metagenome assembly offer even deeper challenges. There exist many generic software to transform high throughput sequencing data into genomic sequences, and several algorithmic

[3] Preprint version.
The final authenticated version is available online at https://doi.org/10.1016/j.compbiolchem.2021.107516.

and heuristic approaches have been implemented, each intended to perform best on different kinds of sequencing data. This includes, for instance, the proprietary CLC suite, very popular among biologists, or the De Bruijn Graphs based tools MetaSPADES [1], MegaHit [2], Velvet [3], ABySS [4] or IDBA-UD [5]. Other tools are based on the overlap-layout-consensus, which use all-against-all alignments and overlap graphs, and include SGA [6] and Edena [7]. A survey on recent methods can be found in [8].

When faced with the plethora of available options, the following question is inevitably raised: which tool is the best to assemble a particular dataset? Performing assembly using a selection of multiple software can be informative, but also raises the issue of choosing the best sets of contigs inferred by each approach. An interesting option is to take the best parts of all assemblies and combine them into a better one. This can be especially useful when assembling metagenomic datasets, since different tools are more likely to produce different contigs (see e.g. [9, 10]). In this paper, we address the problem of creating a consensus from multiple assemblies obtained from different methods. The main idea of our approach is that if two contigs partially overlap, they might belong to the same genomic segment and we should merge them. Of course, several conflicting overlaps may arise, and so we must align contigs according to some scoring function. We are interested in the algorithmic complexity aspects of this problem.

Genome assembly in a nutshell. We focus here on second and third generation of sequencing data, respectively called short and long reads. Short reads contain hundreds of bases, offer generally high coverage depth, and are usually paired-end reads (issued from amplified fragments that each generating two reads). Long reads contain thousands of bases and come from single-end technologies [11]. Assembly tools are usually tailored for one specific type of read, although hybrid approaches have recently been developed (e.g. [12]). From an algorithmic point of view, the k-mer approaches use the De Bruijn Graph of the reads, in which consecutive overlaps lead to long paths that represent contigs. Because the value of k is typically small, the generated contigs tend to be shorter but more accurate, and are useful

for short reads [13]. Methods based on the assembly graph also work with overlaps, but result in longer contigs since they keep entire reads instead of splitting them into k-mers [14]. One downside of these approaches is that they are prone to chimeric reads. In a metagenomic context, several subspecies from a common environment are sequenced and the origin of each read is difficult to determine [15], which leads to even more discrepancies between the contigs inferred by distinct methods. The fact that various approaches have different strengths and weaknesses motivates the need for consensus methods.

State of the art. Comparison of sequences at the genome level usually involves multiple alignment, which have been shown useful for aligning whole genomes of scaffolds [16]. At the contig level, the quantity of small sequences and the lack of order among contigs makes the multiple alignment approach inappropriate. One of the first standard tools suite to compare sets of sequences at the contig level is CD-HIT, which performs a pairwise comparison followed by a clustering step [17]. Clusters are useful when grouping similar sequences together, but miss partially overlapping contigs and ignore the fact that several contigs of one dataset may be required to "cover" each other (see Figure 1). Assembly consensus has also been called *assembly reconciliation* in the literature [18]. Several methods have been published in the last decade, although many of them are designed for specific groups of species [19–21] or assume knowledge of a reference assembly [22, 23]. Other general-purpose consensus software exist [19, 24] but to our knowledge, they are based on heuristic approaches, have not formulated clear optimization criteria for assembly consensus and have not focused on the complexity aspects. The closest to our work is [25], where the authors propose a precise formulation of the problem of fragmented sequence alignment. This is analogous to our contig consensus problem, although this work was motivated by human and mouse genome comparisons, whereas our input contigs are from the same genome. The authors do provide a 3-approximation that may be adapted to produce a consensus. However, their algorithm is based on local search and, although it is shown to terminate in polynomial time, it is difficult

to determine its exact complexity. We estimate that it runs in at least $\Omega(|\mathcal{M}|^6)$ time, where $\mathcal{M}$ is the set of pairs of substrings that can possibly be matched.

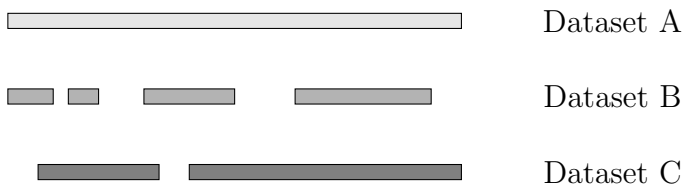
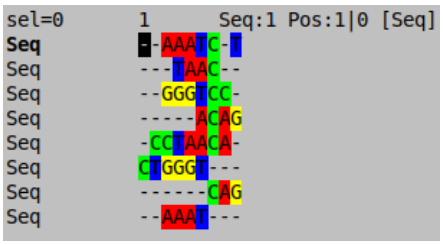


Fig. 1. Three datasets with various sizes of contigs. Dataset A has a long contig, Dataset B gives smaller contigs and Dataset C is intermediary. A clustering using sequence similarity does not catch the fact that each of these datasets represents the same information.

Contributions. We formalize the contig consensus problem : given several contigs datasets, all issued from a same sequencing dataset but assembled by different tools, we want to orient and align the contigs while maximizing a given scoring function. We want to know which contigs and sets of contigs are "the same". For instance, a long contig produced by the tool A may be "the same as" a set of fragmented contigs produced by the tool B. Chimeric or erroneous contigs complicate this process, so instead of finding substrings that match exactly, we aim to maximize a scoring function. We focus on two datasets in this paper. Though we denote by "aligning two assemblies" the proposal of this consensus, this is not reducible to a multiple alignment problem. Indeed, multiple alignment aims, given a set of contigs, to align them together, our goal here is to offer the possibility to use several contigs to cover another one or having several sets of contigs to produce the same sequence, and find the best combination of contigs in each dataset, in this purpose. See Figure 2 for an example.

We show that aligning two assemblies optimally is NP-complete, even in the most restricted settings where the alphabet is binary and the scores between characters are 0 or 1. We also describe a 3-approximation that runs in time $O(|\mathcal{M}|^2)$ on any scoring function, where again $\mathcal{M}$ is the set of matchable substrings. This is a significant improvement on the local search complexity of [25], making our approach is usable in practice on large assemblies. The algorithm introduces novel ideas by taking the best of three sub-problems and

$\mathcal{A} = \{AAATCT, TAAC, GGGTCC, ACAG\}, \mathcal{B} = \{CCTAACA, CTGGGT, CAG, AAAT\}$



$\mathcal{A}$ AAATCT GGGTCC TAAC ACAG

$\mathcal{B}$ AAAT CTGGGT CCTAACA CAG

Consensus AAATCTGGGTCCTAACACAG

Fig. 2. Two sets of contigs representing the same sequence. Set $\mathcal{A}$ and $\mathcal{B}$ represent contigs issued from two different aligners. On the left, the output of a multiple aligner. On the right, the desired assemblies consensus.

representing matches in a graph admitting a so-called bisimplicial elimination orderings. For better readability, proofs has been moved to appendix.

2 Notation and Problem Description

We write $[i] = \{1, \ldots, i\}$ and $[i, j] = \{i, i+1, \ldots, j\}$. For a string w, we write $w[p]$ for the p-th character of w. We let w^r denote the reverse of w.

Let $W = \{w_1, \ldots, w_n\}$ be a set of strings. We write $W^r = \{w^r : w \in W\}$ for the set of reversed strings. A *W-assignment* is a pair of function (ρ, ϕ) where $\rho : W \to \mathbb{N}$ and $\phi : W \to \{f, r\}$. Here, ρ assigns a position to each string, and ϕ an orientation (f stands for forward, r for reverse). The interval occupied by $w \in W$ is denoted $I_\rho(w) := [\rho(w), \rho(w) + |w| - 1]$. We require that no two distinct $w_i, w_j \in W$ overlap, i.e. $I_\rho(w_i) \cap I_\rho(w_j) = \emptyset$. Let $\mathcal{A}(W)$ be the set of all possible W-assignments.

For $w \in W$ and $k \in I_\rho(w)$, we say that character $w[k - \rho(w) + 1]$ *occupies* position k if $\phi(w) = f$, and that $w^r[k - \rho(w) + 1]$ occupies position k if instead $\phi(w) = r$. If there is no character that occupies position k, then we say that λ occupies position k, where λ is a null symbol. Let $C_{\rho,\phi}(k)$ denote the character that occupies position k with respect to ρ and ϕ (note that this character is unique).

In the contig alignment problem, we are given two sets of strings $S = \{s_1, \ldots, s_n\}$ and $T = \{t_1, \ldots, t_m\}$, along with a scoring function $\sigma : \Sigma \times \Sigma \to \mathbb{N}$, where Σ is the alphabet. We

will assume that σ is symmetric, and that $\sigma(\lambda, x) = \sigma(x, \lambda) = 0$ for any $x \in \Sigma$. Our goal is to find an S-assignment and a T-assignment that maximizes the score of matched positions.

Maximum String Matching	
Input:	two sets of strings S and T, a scoring function $\sigma : \Sigma \times \Sigma \to \mathbb{N}$
Find:	an S-assignment (ρ, ϕ) and a T-assignment (ρ', ϕ') that maximizes $$\sum_{k=1}^{\infty} \sigma(C_{\rho,\phi}(k), C_{\rho',\phi'}(k)).$$

More concretely, it is not necessary to assign positions up to infinity. We may assume that no position matches two λ characters, which bounds the maximum position occupied to the sum of lengths of S and T strings.

Note that Veeramachaneni et al. showed in [25] that a version of our problem that allows gaps in the alignment is MAX-SNP-hard, if allowed an arbitrary alphabet. The hardness results on binary alphabets for Maximum String Matching, presented in the following, can be adapted for this problem.

3 Computational Hardness

In this section, we show that Maximum String Matching is NP-hard even in the most restricted setting, which is when Σ is binary and the σ costs are 0 or 1. Moreover, our hardness result holds even if, in addition, each string is symmetric, i.e. $w = w^r$ for each $w \in S \cup T$, thereby removing the problem of orienting contigs. We build a reduction from 3-Partition defined as follows.

3-Partition Problem	
Input:	A multiset N of $3m$ positive integers and an integer n.
Find:	A partition of N into m triples such that all triples have the same sum n.

In [26], it is shown that 3-Partition is NP-complete even if, for every integer $i \in N$, we have $n/4 < i < n/2$.

Construction 1. *Let $N = \{x_1, \ldots, x_{3m}\}$ be a multiset of $3m$ positive integers such that the sum of its integer is equal to $m \times n$. We construct an instance of* Maximum String Matching *over the alphabet $\{\alpha, \beta\}$ as follows.*

- *For each integer x_i, we add a string s_{x_i} into S composed by exactly x_i characters α.*
- *We construct m strings $t_1, \ldots, t_m$ in T such that for each $i \leq m$, $|t_i| = n + 2$ and $t[1] = t[n+2] = \beta$, whereas $t[j] = \alpha$ for each $j \in \{2, \ldots, n+1\}$.*

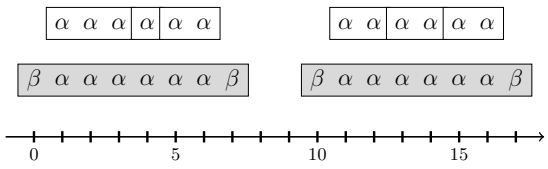


Fig. 3. Example of an instance produced by Construction 1 from the multiset $N = \{3, 2, 2, 2, 2, 1\}$

An example of an instance produced by the previous construction is depicted in Figure 3. For each $t_i \in T$, we call the interval $t_i[2, n+1]$ an α-interval. The idea of the reduction is the following: if we manage to construct an assignment with score $m \times n$, then its means that every α-interval is entirely covered with three strings $(s_{x_i}, s_{x_j}, s_{x_k})$. Since the sum of x_i, x_j and x_k is equal to n, we can construct the triple (x_i, x_j, x_k) into the solution. It leads us to the following result.

Theorem 1. Maximum String Matching *is NP-hard even on symmetric strings on binary alphabets and σ costs in $\{0, 1\}$.*

Proof. Let $N = \{x_1, \ldots, x_{3m}\}$ be a multiset of $3m$ positive integers such that each integer is strictly between $\frac{n}{4}$ and $\frac{n}{2}$. Let S, T be the sets of strings produced by Construction 1.

First note that 3-Partition is strongly NP-complete, meaning that even if the maximum value in N is bounded by a polynomial function in the input size, the problem remains NP-complete. In that case, string lengths in S and T are also bounded by a polynomial function in the input size of 3-Partition. Thus S and T can be built in polynomial time.

It remains to show that N is a yes-instance of 3-Partition Problem if and only if the instance resulting from Construction 1 admits an assignment with score $m \times n$.

"$\Rightarrow$" Let $\{N_1, \ldots, N_m\}$ be a partition of N that is a solution for the 3-Partition Problem. We construct an S-assignment (ρ, ϕ) and a T-assignment (ρ', ϕ') as follows. First note that since each string of S and T are symmetric (*i.e.* $w = w^r$), the orientation is irrelevant and it only suffices to assign a position for each string. For each $i \leq m$, we set $\rho'(t_i) = i(n+2)+1$. For the S-assignment, we place the strings corresponding to the three integers of N_i alongside the α-intervals of t_i without space between them. Formally, let $N_i = (x_j, x_k, x_\ell)$. We set $\rho(s_{x_j}) = \rho'(t_i) + 1, \rho(s_{x_k}) = \rho(s_{x_j}) + x_j$ and $\rho(s_{x_\ell}) = \rho(s_{x_k}) + x_k$. Since $x_j + x_k + x_\ell = n$, the α-interval of t_i is entirely covered and t_i has score n. Thus, we construct an assignment with score $m \times n$.

"$\Leftarrow$" Consider an assignment with score $m \times n$ for S and T. We construct a partition $\{N_1, \ldots, N_m\}$ as follows. First, note that since there are exactly $m \times n$ α-characters in both S and T, each S-string must be positioned over a unique α-interval since otherwise a α-character of S would not match with another α-character of T. And in that case the score $m \times n$ can not be reached. Further, since each integer is strictly between $n/4$ and $n/2$, then the length of any string in S is also strictly between $n/4$ and $n/2$. Thus for each $t_i \in T$, the α-interval of t_i intersects with exactly three strings s_{x_j}, s_{x_k} and s_{x_ℓ} such that $|s_{x_j}| + |s_{x_k}| + |s_{x_\ell}| = n$. We can then add the triple $N_i = (x_j, x_k, x_\ell)$ in the partition.

4 Quadratic-time Approximation Algorithm

In the rest of this paper, we devise an approximation algorithm for Maximum String Matching. Our first goal is to reduce the problem to that of matching substrings of S and T. A similar technique was used in [25]. Before presenting the algorithm, we need to introduce some additional notions.

4.1 Reduction to string matches

We extend the scoring function σ to strings as follows: for strings s and t of the same length l, we define $\sigma(s,t) = \sum_{i=1}^{l} \sigma(s[i], t[i])$.

Let $s \in S \cup S^r$ and $t \in T \cup T^r$. Consider two substrings $s[i..j]$ and $t[p..q]$ of s and t, respectively. If these two substrings were matched in a solution, their contribution to the score would be $\sigma(s[i..j], t[p..q])$. We reformulate the problem as one of finding a set of maximum score matches that can be part of a solution. This reformulation is somewhat more technical, but more convenient to work with.

A *string-interval* of w is a triple $I = [w, i, j]$ in which w is a string, $1 \leq i < j \leq |w|$. The triple $[w, i, j]$ can be thought of as the $w[i..j]$ substring, but with the additional information that it originates from w (so two equal substrings are considered different if they originate from different strings). We say that two string-intervals $[w, i, j]$ and $[w', p, q]$ *intersect* if and only if $w = w'$ and $[i, j] \cap [p, q] \neq \emptyset$. Given two string-intervals $I = [w, i, i + k]$ and $J = [w', j, j + k]$ of the same length, we define $\sigma(I, J) = \sigma(w[i..i + k], w'[j..j + k])$.

Let $s \in S \cup S^r$ and $t \in T \cup T^r$. We say that a pair of string-intervals

$$(I, J) = ([s, i, i + k], [t, j, j + k])$$

of the same length is a *match*. The *length* of such a match is $k + 1$.

We distinguish four types of matches that can occur in a solution:

- we match a prefix of s with a suffix of t: $i = 1$ and $j + k = |t|$;

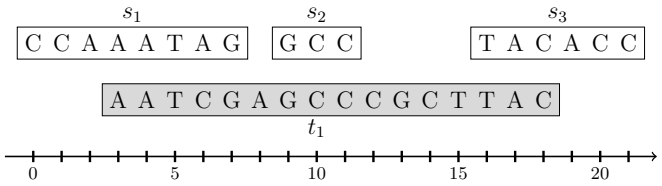


Fig. 4. Assume unit scores for matching identical characters. Example of matches: $([s_1, 4, 8], [t_1, 1, 5])$ is a suffix match of score 4, $([s_2, 1, 3], [t_1, 7, 9])$ is a full match of score 3 and $([s_3, 1, 3], [t_1, 14, 16])$ is a prefix match of score 3.

- we match a suffix of s with a prefix of t: $i + k = |s|$ and $j = 1$;
- we match all of s into t: $i = 1$ and $i + k = |s|$;
- we match all of t into s: $j = 1$ and $j + k = |t|$.

The first type of match is called a *prefix match*, the second type is called a *suffix match*, and the last two are called *full matches* (see Figure 4). If (I, J) is of one of the first three types, we say that (I, J) matches s into t, and otherwise it matches t into s. Note that a full match could also be a prefix and/or suffix match, since a string is a prefix and suffix of itself.

Conflicting matches. Two matches $(I, J) = ([s_1, i, i + k], [t_1, j, j + k])$ and $(I', J') = ([s_2, p, p + r], [t_2, q, q + r])$ are *in conflict* if at least one of the following conditions holds:

- I intersects with I' or J intersects with J';
- $s_1 = s_2^r$, or $t_1 = t_2^r$.

That is, we forbid matches that overlap, and we forbid matching a substring of s and a substring of s^r, forcing us to choose only one orientation.

We need to define an auxiliary graph to define what we consider as a compatible set of matches. Let $\mathcal{M}$ be a set of matches. it would be intuitive to construct a graph with vertex set $\mathcal{M}$, adding an edge between conflicting matches, and finding a maximum weight independent set. This does not quite work, as there are examples in which this approach ends up assigning both a contig and its reverse. The auxiliary graph we need is actually used to determine when such an independent set is valid or not.

More precisely, let $\mathcal{M}' \subseteq \mathcal{M}$ be a subset of matches in which no two matches are in conflict. Construct a directed graph $D(\mathcal{M}') = (S \cup T, E)$ in which the vertices are the input strings. For $s \in S$ and $t \in T$, we add the arc (s, t) to E if $\mathcal{M}'$ contains a suffix match from either s or s^r into t or t^r. We also add an arc (t, s) if $\mathcal{M}'$ contains a prefix match from either s or s^r into t or t^r.

In what follows, the score of a set of matches $\mathcal{M}'$ is the sum of scores of the matches in $\mathcal{M}'$.

Theorem 2. *Let S, T and σ be an instance of* Maximum String Matching*, and let $\mathcal{M}$ be the set of all possible prefix, suffix and full matches between S and T. Then there exists an S- and T-assignment of score ℓ if and only if there exists $\mathcal{M}' \subseteq \mathcal{M}$ of score ℓ such that no two matches of $\mathcal{M}'$ are in conflict, and such that $D(\mathcal{M}')$ is acyclic.*

We have established that one strategy to solve Maximum String Matching is to list all the possible matches, and then to find a subset $\mathcal{M}'$ satisfying the above. Note that instead of listing all the matches, another idea would be to run BLAST to retain only significant hits between the S and T strings, and maximize those matches only. This might not always yield the absolute best solution, but would certainly accelerate the process. The next lemma describes the total number of matches if they are all listed, and may be helpful in determining when it is necessary to rely on BLAST-type heuristics.

Lemma 1. *Let S, T and σ be an instance of* Maximum String Matching*. Then the number of possible prefix, suffix and full matches is $O(|S| \cdot \ell(T) + |T| \cdot \ell(S))$, where $\ell(S)$ (resp. $\ell(T)$) is the sum of lengths of the strings in S (resp. T).*

4.2 Algorithm Description

We now describe an $O(|\mathcal{M}|^2)$ time 3-approximation for the problem of finding a maximum weight conflict-free subset $\mathcal{M}' \subseteq \mathcal{M}$ such that $D(\mathcal{M}')$ is acyclic. One of the main challenges is to avoid cycles in $D(\mathcal{M}')$ while forming matches. Our idea is to restrict the problem to subsets of matches that are guaranteed to avoid cycles. Algorithm 1 describes the high-level

approach. In the subsequent sections, we will detail how each portion can be approximated efficiently.

function *main($\mathcal{M}$)*
  Let $\mathcal{M}_1 \subseteq \mathcal{M}$ be the set of matches that are either:
    - a prefix match involving some $s \in S$
    - a suffix match involving some $s^r \in S^r$
    - a full match
  Compute a solution $\mathcal{M}'_1$ to the instance $\mathcal{M}_1$

  Let $\mathcal{M}_2 \subseteq \mathcal{M}$ be the set of matches that are either:
    - a suffix match involving some $s \in S$,
    - a prefix match involving some $s^r \in S^r$, or
    - a full match.
  Compute a solution $\mathcal{M}'_2$ to the instance $\mathcal{M}_2$

  Let $\mathcal{M}_3 \subseteq \mathcal{M}$ be the set of prefix and suffix matches
  Compute a solution $\mathcal{M}'_3$ to the instance $\mathcal{M}_3$

  Return the maximum score solution among $\mathcal{M}'_1, \mathcal{M}'_2$ and $\mathcal{M}'_3$

Algorithm 1: Main approximation

We show that if a reasonable approximation can be obtained for each subproblem, then one of them also yield a reasonable global approximation.

Theorem 3. *Let $\mathcal{M}_1, \mathcal{M}_2$ and $\mathcal{M}_3$ be defined as in Algorithm 1. Assume that $\mathcal{M}'_1$ and $\mathcal{M}'_2$ are an α-approximation to instances $\mathcal{M}_1$ and $\mathcal{M}_2$, respectively. Also assume that $\mathcal{M}'_3$ is a β-approximation to instance $\mathcal{M}'_3$. Then Algorithm 1 is a $\frac{2\alpha+\beta}{2}$-approximation.*

We will show that ratios $\alpha = 2$ and $\beta = 2$ can be achieved, implying a 3-approximation. It should be noted that improving either of these ratios might be possible, and would immediately improve the approximability of our problem.

The quadratic time will follow from the fact that each of $\mathcal{M}_1, \mathcal{M}_2$ and $\mathcal{M}_3$ can be computed in time $|\mathcal{M}|^2$.

4.3 Restricted prefix, suffix and full matches

In this section, we allow only the subset of matches $\mathcal{M}_1$, which we will treat as an input for the restricted problem. To simplify notation, we will call this restricted input $\mathcal{M}$ in this section. We are thus given a set $\mathcal{M}$ of matches, and assume that it contains only prefix matches of S strings, suffix matches of S^r strings, and full matches. We will also assume that a match $M = ([s, i, i+k], [t, j, j+k])$ is in $\mathcal{M}$ if and only if the corresponding reverse match is also present, i.e. $M' = ([s^r, |s| - i - k + 1, |s| - i + 1], [t^r, |t| - j - k + 1, |t| - j + 1])$ is also in $\mathcal{M}$. The score of M and M' is also the same. Observe that if $\mathcal{M}$ is generated from all possible matches as in the previous section, then this assumption holds, even after restricting our instance.

We are looking for a maximum score subset $\mathcal{M}' \subseteq \mathcal{M}$ such that $D(\mathcal{M}')$ is acyclic and no two matches of $\mathcal{M}'$ are in conflict. In fact, we may already observe that under our restrictions, for any solution $\mathcal{M}'$, $D(\mathcal{M}')$ must be acyclic.

Lemma 2. *For any $\mathcal{M}' \subseteq \mathcal{M}$, $D(\mathcal{M}')$ is acyclic.*

Proof. Let $s \in S$ and recall that s cannot be in a prefix match because of our restrictions. Thus s has no in-neighbor in $D(\mathcal{M}')$ and thus cannot be part of a cycle. Similarly, any $s^r \in S^r$ cannot be in a suffix match, has no out-neighbor in $D(\mathcal{M}')$ and cannot be in a cycle. It follows that $D(\mathcal{M}')$ has no cycle. □

It therefore suffices to look for a conflict-free set of matches. By modeling matches as a graph, with matches as vertices and conflicts as edges, this reduces to finding a maximum-weight independent set. In order to get our desired approximation though, we need to simplify this graph and get rid of reverse strings. We describe how to achieve this by *projecting* each reversed substring into its forward counterpart.

For a string-interval $[w, i, j]$ with $w \in S \cup S^r \cup T \cup T^r$, the *projection* of w is the interval

$$P([w,i,j]) = \begin{cases} [w,i,j] & \text{if } w \in S \cup T \\ [w^r, |w|-j+1, |w|-i+1] & \text{if } w \in S^r \cup T^r \end{cases}$$

Roughly speaking, $P[w,i,j]$ occupies the same characters as $[w,i,j]$, but relative to the forward string only.

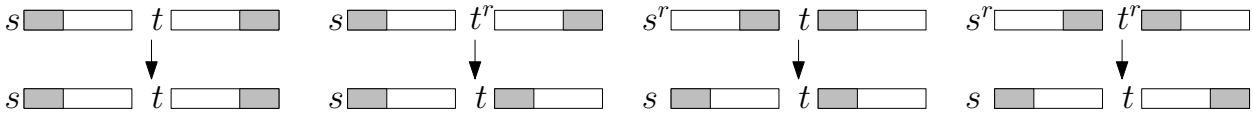


Fig. 5. An illustration of matches after projecting. The first two cases show a suffix match of a $s \in S$ string, and the last two a prefix match of some $s^r \in S^r$. Not shown : full matches.

Let $P(\mathcal{M}) = \{(P(I), P(J)) : (I,J) \in \mathcal{M}\}$ be the set of projected matches. For each $(P(I), P(J)) \in P(\mathcal{M})$, assign the score $\sigma(P(I), P(I)) = \sigma(I,J)$. An illustration of projections is shown in Figure 5. Note that projected matches may **not** satisfy the restrictions of prefix and suffix matches. This is not a problem, as projected matches are only conceptual. As we show, if $\mathcal{M}$ is restricted, we may safely project every match.

Lemma 3. *Assume that $\mathcal{M}$ is a set of matches under our restrictions. Then there is a conflict-free subset $\mathcal{M}' \subseteq \mathcal{M}$ of score k if and only if there exists a conflict-free subset $\mathcal{P} \subseteq P(\mathcal{M})$ of score k.*

4.4 Projected matches and bisimplicial elimination orderings

For the remainder, we assume that we have converted our set of matches $\mathcal{M}$ into their projections $\mathcal{P}$. Recall that in $\mathcal{P}$, all matches involve only forward strings. Thus, the only possible conflicts are intersections of string-intervals. Consider the graph $G(\mathcal{P}) = (V,E)$ in which $V = \mathcal{P}$, and $M_1, M_2 \in \mathcal{P}$ share an edge if they intersect. We need to find an independent set of maximum weight in $G(\mathcal{P})$.

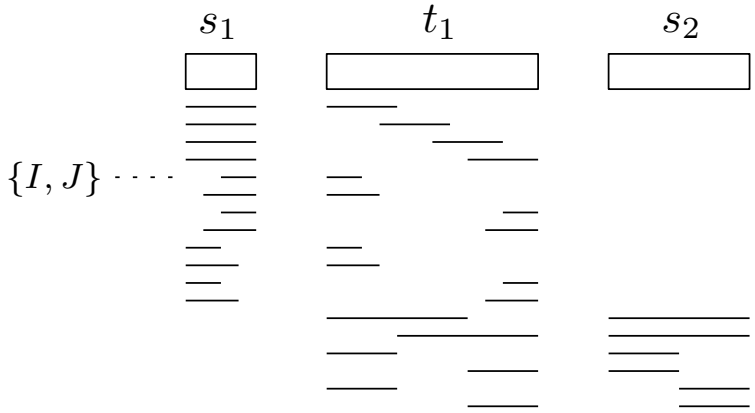


Fig. 6. An illustration of matches between strings s_1, t_1 and s_2. Two lines aligned horizontally represent the two substrings of a match. The match $\{I, J\}$ pointed by the dashed line has a late finisher in s_1 and an early ender in t_1. One can see that all matches that intersect with I must intersect with its last position, and all matches that intersect with J must intersect with its first position. Therefore, $\{I, J\}$ is bisimplicial.

It turns out that $G(\mathcal{P})$ admits a so-called bisimplicial elimination ordering, and that a 2-approximation for independent set exists in graphs that have this property. In a graph G, a vertex v is called *bisimplicial* if the neighbors of v can be partitioned into two cliques (with possible edges between the cliques). An ordering $(v_1, \ldots, v_n)$ of $V(G)$ is called a *bisimplifical elimination ordering* (BEO) if, for each $i \in [n]$, v_i is bisimplicial in $G[\{v_i, \ldots, v_n\}]$, the subgraph induced by $\{v_i, \ldots, v_n\}$. Finding a bisimplicial vertex can be done in time $O(|V(G)|^3)$ by checking the neighbors of each vertex, and thus finding a BEO, if any, can be done in time $O(|V(G)|^4)$. To our knowledge, no better algorithm is known, but in the particular case of $G(\mathcal{P})$ graphs, it can be done faster.

Before proceeding, we need some additional notions. To simplify the presentation, assume that the matches in $\mathcal{P}$ are unordered pairs of string-intervals (membership in S or T will not matter at this point). Therefore, $\{I, J\} \in \mathcal{P}$ and $\{J, I\} \in \mathcal{P}$ refer to the same match.

Let $\mathcal{I} = \{H : \{I, J\} \in \mathcal{P} \text{ and } H \in \{I, J\}\}$ be the set of all string-intervals involved in $\mathcal{P}$. We say that $[w, i, j] \in \mathcal{I}$ is a w-interval. A w-interval $I = [w, i, j]$ is *prefix* if $i = 1$, *suffix* if $j = |w|$ and *full* if it is both prefix and suffix. Moreover, a string-interval $I = [w, i, j] \in \mathcal{I}$ is called an *early-ender* if, for any other w-interval $[w, i', j']$, we have $j' \geq j$. Similarly, I is

called a *late-starter* if, for any other w-interval $[w, i', j']$, we have $i' \leq i$. Figure 6 provides an illustration.

We can now describe a procedure that constructs a BEO.

```
1  function beo(P)
2      B = ()                                  // B is the BEO we are constructing.
3      while P is not empty do
4          if all string-intervals I ∈ 𝓘 are full then
5              Append all elements of P to B in any order
6              Return B
7          Let I be any early-ender or late-starter that is not full
8          F = null                            // F is the next bisimplicial vertex
9          while F = null do
10             Let {I, J} ∈ P be any match containing I
11             Let w ∈ S ∪ T such that J is a w-interval
12             if J is an early-ender or a late-starter then
13                 F = {I, J}
14             else if J is suffix or full then
15                 I = any late-starter w-interval
16             else
17                 I = any early-ender w-interval
18         end
19         Append F = {I, J} to B
20         Remove F from P
21     end
22     return B
```

Algorithm 2: BEO construction

Lemma 4. *Algorithm 2 constructs a bisimplicial elimination ordering of $G(\mathcal{P})$ in time $O(|\mathcal{P}|^2)$.*

In [27], the authors provide a 2-approximation for finding a maximum weight independent set in a graph, given a BEO. The exact complexity was not analyzed, and since running times are relevant to our motivations, we provide Algorithm 3 our version of the algorithm. This can be seen as the local ratio version of the problem (the local ratio is a well-known approximation technique, see [28]).

Algorithm 3 can easily be implemented in time $O(|\mathcal{P}|^2)$, since each recursion takes time $O(|\mathcal{P}|)$ to update the weights, and there are $O(|\mathcal{P}|)$ recursive calls made. This can also be shown to be a 2-approximation.

```
function maxMatches(P, σ, B)
    //σ is the scoring function, B is a BEO of G(P)
    if B is empty then
        Return ∅
    Let F be the first element of B
    Remove F from B
    Let σ_1 be the weight function defined as
    σ_1(F') = { σ(F')  if F' = F or F' intersects with F
              { 0      otherwise
    Let σ_2 = σ − σ_1
    M' = maxMatches(P, σ_2, B)
    if no element of M' intersects with F then
        Add F to M'
    return M'
```

Algorithm 3: Algorithm to compute a maximum weight subset of conflict-free matches.

Theorem 4. *Algorithm 3 is a 2-approximation to the problem of computing a maximum weight subset of conflict-free matches and runs in time* $O(|\mathcal{P}|^2) = O(|\mathcal{M}|^2)$.

4.5 Other restricted sets of matches

Recall that our high-level algorithm needs an approximation for $\mathcal{M}_2$, the subset of matches that contain only prefix matches of S^r strings, suffix matches of S strings, and full matches. There is no fundamental difference with the previous case: by swapping the roles of S and S^r, we can use the same algorithm as before (that is, we treat S^r as the forward strings, and S as the reverse strings). There is therefore a factor 2-approximation for this case.

If only prefix and suffix matches are allowed, it is straightforward to obtain a 2-approximation. For each $s \in S$ and $t \in T$, let $M_{s,t} \in \mathcal{M}$ be the prefix or suffix match of one of s, s^r into one of t, t^t that has maximum score. Create a bipartite graph $H = (S \cup T, E)$ in which, between each $s \in S, t \in T$, there is an edge between s and t of weight $\sigma(M_{s,t})$.

We claim that a maximum weight matching in H yields a 2-approximation. To see this, let OPT be a maximum subset of matches, restricted to only prefix and suffix matches. Let OPT_1 be the set of prefix matches, and OPT_2 the set of suffix matches. One of OPT_1 or

OPT_2 has score at least $\sigma(OPT)/2$. Observe that each string is part of at most one match in OPT_1, and the same holds for OPT_2. Therefore, a maximum matching in H corresponds to a score at least as high as that of either OPT_1 or OPT_2. The 2-approximation follows. As we show, the complexity is quadratic in $|S|+|T|$, times a logarithmic factor. If $\mathcal{M}$ is assumed to contain every possible match as we proposed earlier, the running time is quadratic in $|\mathcal{M}|$.

Theorem 5. *Denote $n = |S|+|T|$. If $\mathcal{M}$ contains only prefix or suffix matches, then* Maximum String Matching *admits a 2-approximation that runs in time $O(n^2 \log n + n \cdot |\mathcal{M}|)$. Moreover, assuming that $\mathcal{M}$ contains a match between each pair of strings in S and T, the running time is $O(|\mathcal{M}|^2)$.*

5 Conclusion

This paper is devoted to a new problem whose objective is to reach a consensus between several sets of assembly data. We prove the NP-hardness of Maximum String Matching, even in a restricted case. Finally, we also propose a quadratic 3-approximation algorithm to solve this problem. A natural perspective of our work would be to explore the practical aspects of this algorithm. Following the same idea, a longer term perspective would be to enhance the exploitation of the results, by pipelining them into a visualization tool, and propose a way to merge datasets using consensus clusters and alignments. As a second perspective, further studies could be led towards algorithmic improvements. Notably, improving the approximation factor of the suffix/prefix only case would immediately improve our global approximation. A third perspective is to consider the possibility to extend our problem and analysis to other purposes, for instance help the binning step in metagenomics, discover repeated regions, and help comparing taxonomic assignation tools' robustness with respect to the meta-assembly tool.

References

1. S. Nurk, D. Meleshko, A. Korobeynikov, and P. A. Pevzner, "metaSPAdes: a new versatile metagenomic assembler," *Genome Res*, vol. 27, pp. 824–834, 05 2017.
2. D. Li, R. Luo, C. M. Liu, C. M. Leung, H. F. Ting, K. Sadakane, H. Yamashita, and T. W. Lam, "MEGAHIT v1.0: A fast and scalable metagenome assembler driven by advanced methodologies and community practices," *Methods*, vol. 102, pp. 3–11, 06 2016.
3. D. R. Zerbino and E. Birney, "Velvet: algorithms for de novo short read assembly using de bruijn graphs," *Genome research*, vol. 18, no. 5, pp. 821–829, 2008.
4. J. T. Simpson, K. Wong, S. D. Jackman, J. E. Schein, S. J. Jones, and I. Birol, "Abyss: a parallel assembler for short read sequence data," *Genome research*, vol. 19, no. 6, pp. 1117–1123, 2009.
5. Y. Peng, H. C. Leung, S. M. Yiu, and F. Y. Chin, "IDBA-UD: a de novo assembler for single-cell and metagenomic sequencing data with highly uneven depth," *Bioinformatics*, vol. 28, pp. 1420–1428, Jun 2012.
6. J. T. Simpson and R. Durbin, "Efficient de novo assembly of large genomes using compressed data structures," *Genome research*, vol. 22, no. 3, pp. 549–556, 2012.
7. D. Hernandez, P. François, L. Farinelli, M. Østerås, and J. Schrenzel, "De novo bacterial genome sequencing: millions of very short reads assembled on a desktop computer," *Genome research*, vol. 18, no. 5, pp. 802–809, 2008.
8. E. Forouzan, P. Shariati, M. S. Mousavi Maleki, A. A. Karkhane, and B. Yakhchali, "Practical evaluation of 11 de novo assemblers in metagenome assembly," *J Microbiol Methods*, vol. 151, pp. 99–105, 08 2018.
9. S. Boisvert, F. Raymond, É. Godzaridis, F. Laviolette, and J. Corbeil, "Ray meta: scalable de novo metagenome assembly and profiling," *Genome biology*, vol. 13, no. 12, pp. 1–13, 2012.
10. T. Namiki, T. Hachiya, H. Tanaka, and Y. Sakakibara, "Metavelvet: an extension of velvet assembler to de novo metagenome assembly from short sequence reads," *Nucleic acids research*, vol. 40, no. 20, pp. e155–e155, 2012.
11. B. E. Slatko, A. F. Gardner, and F. M. Ausubel, "Overview of Next-Generation Sequencing Technologies," *Curr Protoc Mol Biol*, vol. 122, p. e59, 04 2018.
12. D. Antipov, A. Korobeynikov, J. S. McLean, and P. A. Pevzner, "hybridspades: an algorithm for hybrid assembly of short and long reads," *Bioinformatics*, vol. 32, no. 7, pp. 1009–1015, 2016.
13. A. M. Phillippy, "New advances in sequence assembly," *Genome Res*, vol. 27, pp. xi–xiii, 05 2017.
14. M. Kolmogorov, J. Yuan, Y. Lin, and P. A. Pevzner, "Assembly of long, error-prone reads using repeat graphs," *Nat Biotechnol*, vol. 37, pp. 540–546, 05 2019.
15. D. D. Roumpeka, R. J. Wallace, F. Escalettes, I. Fotheringham, and M. Watson, "A review of bioinformatics tools for bio-prospecting from metagenomic sequence data," *Frontiers in Genetics*, vol. 8, p. 23, 2017.
16. A. E. Darling, B. Mau, and N. T. Perna, "progressiveMauve: multiple genome alignment with gene gain, loss and rearrangement," *PLoS One*, vol. 5, p. e11147, Jun 2010.
17. W. Li and A. Godzik, "Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences," *Bioinformatics*, vol. 22, pp. 1658–1659, Jul 2006.

18. A. V. Zimin, D. R. Smith, G. Sutton, and J. A. Yorke, "Assembly reconciliation," *Bioinformatics*, vol. 24, no. 1, pp. 42–45, 2008.

19. H. Soueidan, F. Maurier, A. Groppi, P. Sirand-Pugnet, F. Tardy, C. Citti, V. Dupuy, and M. Nikolski, "Finishing bacterial genome assemblies with mix," in *BMC bioinformatics*, vol. 14, pp. 1–11, BioMed Central, 2013.

20. S.-H. Lin and Y.-C. Liao, "Cisa: contig integrator for sequence assembly of bacterial genomes," *PloS one*, vol. 8, no. 3, p. e60843, 2013.

21. J. L. Argueso, M. F. Carazzolle, P. A. Mieczkowski, F. M. Duarte, O. V. Netto, S. K. Missawa, F. Galzerani, G. G. Costa, R. O. Vidal, M. F. Noronha, *et al.*, "Genome structure of a saccharomyces cerevisiae strain widely used in bioethanol production," *Genome research*, vol. 19, no. 12, pp. 2258–2270, 2009.

22. G. Yao, L. Ye, H. Gao, P. Minx, W. C. Warren, and G. M. Weinstock, "Graph accordance of next-generation sequence assemblies," *Bioinformatics*, vol. 28, no. 1, pp. 13–16, 2012.

23. L. M. Soto-Jimenez, K. Estrada, and A. Sanchez-Flores, "Garm: genome assembly, reconciliation and merging pipeline.," *Current topics in medicinal chemistry*, vol. 14, no. 3, p. 418, 2014.

24. R. Vicedomini, F. Vezzi, S. Scalabrin, L. Arvestad, and A. Policriti, "Gam-ngs: genomic assemblies merger for next generation sequencing," *BMC bioinformatics*, vol. 14, no. S7, p. S6, 2013.

25. V. Veeramachaneni, P. Berman, and W. Miller, "Aligning two fragmented sequences," *Discrete Applied Mathematics*, vol. 127, no. 1, pp. 119–143, 2003.

26. M. R. Garey and D. S. Johnson, "Complexity results for multiprocessor scheduling under resource constraints," *SIAM J. Comput.*, vol. 4, no. 4, pp. 397–411, 1975.

27. Y. Ye and A. Borodin, "Elimination graphs," *ACM Transactions on Algorithms (TALG)*, vol. 8, no. 2, pp. 1–23, 2012.

28. R. Bar-Yehuda, K. Bendel, A. Freund, and D. Rawitz, "Local ratio: A unified framework for approximation algorithms. in memoriam: Shimon even 1935-2004," *ACM Computing Surveys (CSUR)*, vol. 36, no. 4, pp. 422–463, 2004.

Appendix

A Proof for Section 4

Theorem 2. *Let S, T and σ be an instance of* Maximum String Matching*, and let $\mathcal{M}$ be the set of all possible prefix, suffix and full matches between S and T. Then there exists an S- and T-assignment of score ℓ if and only if there exists $\mathcal{M}' \subseteq \mathcal{M}$ of score ℓ such that no two matches of $\mathcal{M}'$ are in conflict, and such that $D(\mathcal{M}')$ is acyclic.*

Proof. Assume that there exists an S-assignment (ϕ, ρ) and a T-assignment (ϕ', ρ') of score ℓ. We can easily convert the assignments into their set of matching substrings as follows. Let $s \in S$ and $t \in T$. Furthermore, let $s' = s$ if $\rho(s) = f$, and $s' = s^r$ if $\rho(s) = r$. Similarly, let $t' = t$ if $\rho(t) = f$, and $t' = t^r$ if $\rho(t) = r$. We say that $s'[i..i+k]$ is matched with $t'[j..j+k]$ if the following holds:

- $i, j \geq 1, i + k \leq |s|, j + k \leq |t|$; and
- $\phi(s) + i = \phi'(t) + j$.

Furthermore, $s'[i..i+k], t'[j..j+k]$ are maximally matched if $s'[i-1..i+k], t'[j-1..j+k]$ are not matched, and $s'[i, i+k+1], t'[j..j+k+1]$ are not matched.

Construct a set of matches $\mathcal{M}'$ by adding, for each maximally matched substrings $s'[i..i+k], t'[j..j+k]$, a match $([s', i, i+k], [t', j, j+k])$. We must prove the following facts on $\mathcal{M}'$:

1. $\mathcal{M}'$ only contains prefix, suffix and full matches;
2. $\mathcal{M}'$ has total score ℓ;
3. $\mathcal{M}'$ has no conflict;
4. $D(\mathcal{M}')$ has no cycle.

We prove each fact separately.

(1) Let $(I, J) = ([s', i, i+k], [t', j, j+k]) \in \mathcal{M}'$. If $i + k = |s'|$, then (I, J) is a suffix match. If $i = 1$, then (I, J) is a prefix match. So assume that $i > 1$ and $i + k < |s'|$. If $j > 1$, then

we could extend (I, J) by one character to the left, contradicting the maximality of I and J. If $j + k < |t'|$, we could extend (I, J) by one character to the right, again a contradiction. Thus $j = 1$ and $j + k = |t'|$, implying that (I, J) is a full match.

(2) Let P be the set of maximally matched pairs of subtrings. It is not hard to see that $\sum_{k=1}^{\infty} \sigma(C_{\phi,\rho}(k), C_{\phi',\rho'}(k)) = \ell$ is equal to

$$\sum_{(s'[i..i+k],t'[j..j+k])\in P} \sigma(s'[i..i + k], t'[j..j + k])$$

since summing the scores of aligned positions is equivalent to summing the scores of maximally matched substrings. Since $\mathcal{M}'$ contains one match for each element of P, it follows that $\sum_{(I,J)\in\mathcal{M}'} \sigma(I, j) = \ell$ as well.

(3) A character of a string of $S \cup S^r$ can only be part of one maximally matched pair, so no two matches $(I, J), (I', J')$ can intersect at I. The same holds for J and J'. Moreover, since we only match $s \in S$ if $\rho(s) = f$ and only match $s^r \in S^r$ if $\rho(s) = r$, we cannot match both a string s and its reverse.

(4) Observe that in $D(\mathcal{M}')$, any vertex $s \in S$ has at most one incoming neighbor (for a prefix match into some t) and at most one outgoing neighbor (for a suffix match into some t'). Similarly, any vertex $t \in T \cap V(D(\mathcal{M}'))$ has at most one incoming neighbor (for a suffix match from some s) and at most one outgoing neighbor (for a prefix match from some s'). Therefore, if there is a cycle in $D(\mathcal{M})$, it has the form $(s_1, t_1, s_2, t_2, \ldots, s_k, t_k, s_1)$. In particular, $\phi(t_1) + |t_1| < \phi(t_k)$. Moreover, s_1 or its reverse forms a suffix match with t_1 and a prefix match with t_k, which is not possible. We deduce that $D(\mathcal{M}')$ has no cycle.

This proves the first direction of the lemma.

Conversely, assume that there is $\mathcal{M}' \subseteq \mathcal{M}$ of score ℓ such that $\mathcal{M}'$ has no conflict and $D(\mathcal{M}')$ is acyclic. As argued before, vertices of $D(\mathcal{M}')$ can have at most one in-neighbor and one out-neighbor. Since it has no cycles, $D(\mathcal{M}')$ is therefore a collection of paths $P_1, \ldots, P_h$. For each $i \in [h]$, we can obtain an assignment of score $\sum_{M\in P_i} \sigma(M)$ so that the matched

substrings in the matches of P_i also match in this assignment (see Figure 7). The full matches into strings that are in P_i can also be incorporated, and this will always be possible since $\mathcal{M}'$ contains no conflict. That is, matched substrings do not intersect at any position, and we do not match both a string and its reverse since $\mathcal{M}'$ has no conflict.

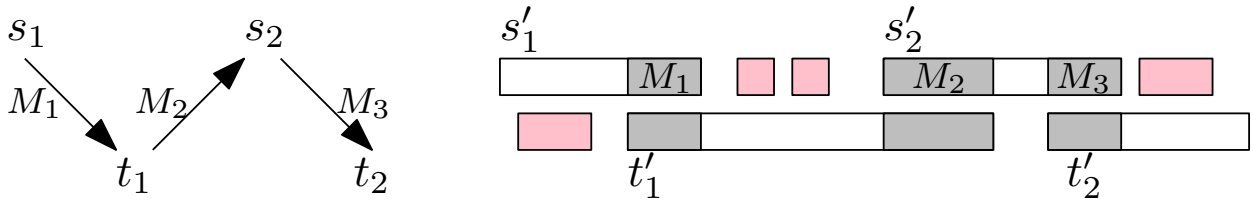


Fig. 7. Transforming a path of $D(\mathcal{M}')$ into an assignment. Each directed edge corresponds to a prefix (M_2) or suffix match (M_1, M_3). Here, the s'_i and the t'_i are the ordered strings that are matched in $\mathcal{M}'$. The pink boxes represent full matches that can be added after the prefix/suffix matches have been taken care of.

Since the P_i paths do not contain a vertex in common, the corresponding assignments can be merged into a global solution simply by concatenation. Since all prefix, suffix and full match present in $\mathcal{M}'$ is in the assignment, its score is the same as $\mathcal{M}'$. □

Lemma 1. *Let S, T and σ be an instance of* Maximum String Matching. *Then the number of possible prefix, suffix and full matches is $O(|S| \cdot \ell(T) + |T| \cdot \ell(S))$, where $\ell(S)$ (resp. $\ell(T)$) is the sum of lengths of the strings in S (resp. T).*

Proof. Consider the number of prefix matches. There are $2\ell(S)$ prefixes of strings in $S \cup S^r$, since for each $s \in S$, we may choose to match one of its $|s|$ prefixes, or one of the $|s|$ prefixes of s^r. For a fixed prefix s' and $t \in T \cup T^r$, there are two possible matches: either with the suffix of t of length $|s'|$, of the suffix of t^r of length $|s'|$. Thus, there are $2|T|$ prefix matches that involve s', and thus a total of $2\ell(S) \cdot 2|T| = 4\ell(S)|T|$ prefix matches. By the same reasoning, we can deduce that there is a total of $4\ell(S)|T|$ possible suffix matches.

Now consider a full match of a string s of $S \cup S^r$ into a string t of $T \cup T^r$. There are $O(|t|)$ positions on which we can match s. It follows that the number of full matches involving s is $O(\ell(T))$. Therefore, there are $O(|S|\ell(T))$ possible matches of an S-string into a T-string.

Similarly, there are $O(|T|\ell(S))$ matches of a T-string into an S-string. Summing up all types of matches gives $O(\ell(S)|T| + |S|\ell(T))$ possible matches. □

Theorem 3. *Let $\mathcal{M}_1, \mathcal{M}_2$ and $\mathcal{M}_3$ be defined as in Algorithm 1. Assume that $\mathcal{M}'_1$ and $\mathcal{M}'_2$ are an α-approximation to instances $\mathcal{M}_1$ and $\mathcal{M}_2$, respectively. Also assume that $\mathcal{M}'_3$ is a β-approximation to instance $\mathcal{M}'_3$. Then Algorithm 1 is a $\frac{2\alpha+\beta}{2}$-approximation.*

Proof. Let OPT be an optimal solution to instance $\mathcal{M}$. Let f be the score of the full matches in OPT, and let b be the score of prefix and suffix matches in OPT. Let OPT_1 (resp. OPT_2) be the solution obtained by removing from OPT all suffix matches involving some $s \in S$ (resp. some $s^r \in S^r$) and all prefix matches involving some $s^r \in S^r$ (resp. some $s \in S$). Note that OPT_1 and OPT_2 both contain all full matches of OPT, but that they partition the prefix and suffix matches. It follows that one of OPT_1 or OPT_2 has a score of $f + \frac{b}{2}$, say OPT_1 without loss of generality. Since OPT_1 is a possible solution to instance $\mathcal{M}_1$, the optimal score for $\mathcal{M}_1$ is at least $f + \frac{b}{2}$. Therefore, an α-approximation to $\mathcal{M}_1$ returns a solution of score at least $\frac{1}{\alpha}(f + \frac{b}{2})$.

Similarly, since $\mathcal{M}_3$ contains all prefix and suffix matches, a β-approximation to instance $\mathcal{M}_3$ returns a solution of score at least $\frac{b}{\beta}$.

It follows that the algorithm returns a solution of score $\max(\frac{1}{\alpha}(f + b/2), \frac{b}{\beta})$. Let x be such that $f = xb$. Then the score of OPT is $f + b = f(1 + 1/x) = b(1 + x)$. The best of $\mathcal{M}'_1$ and $\mathcal{M}'_2$ yields an approximation ratio of

$$\frac{f+b}{\frac{1}{\alpha}(f+b/2)} = \frac{f(1+1/x)}{f(\frac{1}{\alpha}(1+\frac{1}{2x}))} = 2\alpha\frac{x+1}{2x+1}$$

whereas $\mathcal{M}'_3$ yields a ratio of

$$\frac{f+b}{b/\beta} = \frac{b(1+x)}{b/\beta} = \beta(1+x)$$

Observe that the first ratio is decreasing with x, while the second increases with x. The worst ratio therefore occurs when both ratios are equal. Given that $2\alpha\frac{x+1}{2x+1} = \beta(1+x)$ and solving for x, we get $x = \frac{\alpha}{\beta} - \frac{1}{2}$. In that case, the approximation ratio is $\beta(1+x) = \frac{1}{2}(2\alpha+\beta)$. □

Lemma 3. *Assume that $\mathcal{M}$ is a set of matches under our restrictions. Then there is a conflict-free subset $\mathcal{M}' \subseteq \mathcal{M}$ of score k if and only if there exists a conflict-free subset $\mathcal{P} \subseteq P(\mathcal{M})$ of score k.*

Proof. Let $\mathcal{M}'$ be a conflict-free subset of $\mathcal{M}$. We claim that $\mathcal{P} = \{(P(I), P(J)) : (I, J) \in \mathcal{M}'\}$ is also conflict-free. Let $(I, J), (I', J') \in \mathcal{M}'$ be two distinct matches. note that $P(I)$ and $P(I')$ cannot be intervals from $s \in S$ and the other from $s^r \in S^r$, since all projections involve forward strings only. The same holds for $P(J)$ and $P(J')$. Since $\mathcal{M}'$ has no conflict, I and I' cannot intersect and cannot be from string-intervals of both some $s \in S$ and of $s^r \in S^r$. Thus, $P(I)$ and $P(I')$ cannot intersect either : if I and I' are intervals of the same string, then the corresponding intervals after projection still do not intersect, and if I and I' are from different strings, they cannot intersect at all. Similarly, $P(J)$ and $P(J')$ are not in conflict. It follows that $(P(I), P(J))$ and $(P(I'), P(J'))$ are not in conflict. Since this is true for any pair of matches in $\mathcal{M}'$, this proves our claim. Moreover, the score of $\mathcal{P}$ is the same as the score of $\mathcal{M}$, proving the first direction of the lemma.

For the more difficult direction of the proof, let $\mathcal{P} \subseteq \mathcal{P}(\mathcal{M})$ be a conflict-free subset of $\mathcal{P}(\mathcal{M})$. We construct a solution $\mathcal{M}' \subseteq \mathcal{M}$ with the same score. Intuitively, the idea is to "unproject" each element of $\mathcal{P}$ so that all prefix and suffix matches are into forward T strings. This forces us to orient matches $S \cup S^r$ substrings accordingly, so during the construction, we will mark either $s \in S$ or its reverse s^r as "chosen", depending on the matches of $\mathcal{P}$. Once either s or s^r is marked as chosen, we adapt all subsequent matches to make sure we don't match both s and s^r.

Now to make this precise, let $(P(I), P(J)) \in \mathcal{P}$, where (I, J) is the corresponding match. Here, I is a string-interval from some $s \in S$ or its reverse s^r, and J from some $t \in T$ or its

reverse t^r. Assume that (I, J) is a prefix match from $s \in S$, or a suffix match from $s^r \in S^r$. In either case, $P(I) = [s, 1, k]$ for some $s \in S$ and $k \leq |s|$. There are four possibilities, which we handle separately.

1. if I is on string $s \in S$ and J is on string $t \in T$, then mark s as *chosen* and add (I, J) to $\mathcal{M}'$;
2. if I is on string $s^r \in S^r$ and J is on string $t \in T$, then mark s^r as *chosen* and add (I, J) to $\mathcal{M}'$;
3. if I is on string $s \in S$ and J is on string $t^r \in T^r$, then mark s^r as *chosen* (even though I is on s). In this case, $(I, J) = ([s, 1, k], [t^r, |t^r| - k + 1, |t^r|])$ must be a prefix match, but instead we add the suffix match

   $$(I', J') = ([s^r, |s^r| - k + 1, |s^r|], [t, 1, k])$$

   to $\mathcal{M}'$, which we assume is present.

   Observe that $P(I) = P(I')$ and $P(J) = P(J')$. Also note that $\sigma(I', J') = \sigma(I, J) = \sigma(P(I), P(J))$, since it only reverses the order of the matched characters;
4. if I is on string $s^r \in S^r$ and J is on string $t^r \in T^r$, then mark s as *chosen*. In this case, $(I, J) = ([s^r, |s^r| - k + 1, |s^r|], [t^r, 1, k])$ must be a suffix match, but instead we add the prefix match

   $$(I', J') = ([s, 1, k], [t, |t| - k + 1, |t|])$$

   to $\mathcal{M}'$, which we assume is present.

   Observe that $P(I) = P(I')$ and $P(J) = P(J')$. Also note that $\sigma(I', J') = \sigma(I, J) = \sigma(P(I), P(J))$, since it only reverses the order of the matched characters.

Suppose that we have handled every $(P(I), P(J)) \in \mathcal{P}$ corresponding to prefix or suffix matches. Note that for $s \in S$, it is possible that neither s nor s^r is chosen yet. We argue that the matches added so far to $\mathcal{M}'$ are not conflicting (and we handle full matches later).

To prove this, we first claim that for each $s \in S$, we mark at most one of s or s^r as chosen. This is because every prefix match of s or suffix match of s^r contains the character $s[1]$ in their projection. Since $\mathcal{P}$ is conflict-free, we know $\mathcal{P}$ cannot contain a projection of both a prefix and suffix match of s and s^r, respectively, and thus the above cases only adds up to one match to $\mathcal{M}'$ that contains s or s^r. Second, note that we never include a match containing a reversed string $t^r \in T^r$, since each case involving a $t^r \in T^r$ reverses it after unprojecting. It trivially follows that we have not included a match involving both $t \in T$ and its reverse t^r. To argue that no two matches intersect, for each $(P(I), P(J))$ considered, we add a match $(\tilde{I}, \tilde{J})$ with $(P(\tilde{I}), P(\tilde{J})) = (P(I), P(J))$. If two matches of $\mathcal{M}'$ added so far would intersect, their projection would also intersect, but this is impossible since all projections are the same as in $\mathcal{P}$, and $\mathcal{P}$ is conflict-free. Therefore, no conflict was created so far.

It remains to include full matches. Let $(P(I), P(J)) \in \mathcal{P}$ be a projected match corresponding to a full match $(I, J) \in \mathcal{M}$. The orientation of I and J could be forward or reversed, so we shall use s_0 and t_0 for the strings of (I, J) with the understanding that the orientations could be anything. We again separate into cases:

1. Assume that $(I, J) = ([s_0, 1, |s_0|], [t_0, k, k + |s_0| - 1])$ fully matches some $s_0 \in S \cup S^r$ into some $t_0 \in T \cup T^r$. Note that neither s_0 nor s_0^r is marked as chosen at this point, since $\mathcal{P}$ cannot contain another match involving s_0 or s_0^r.
   (a) If $t_0 \in T$, then add (I, J) to $\mathcal{M}'$.
   (b) If $t_0 \in T^r$, then add $(I', J') = ([s_0^r, 1, |s_0|], [t_0^r, |t_0| - k - |s_0| + 1, |t_0| - |s_0| + 1])$ to $\mathcal{M}'$. Observe that $P(I) = P(I')$ and $P(J) = P(J')$ and that the score of (I', J') is the same as (I, J).
2. Assume that $(I, J) = ([s_0, k, k + |t_0| - 1], [t_0, 1, |t_0|])$ fully matches some $t_0 \in T \cup T^r$ into some $s_0 \in S \cup S^r$.
   (a) If s_0 is chosen, then add (I, J) to $\mathcal{M}$;
   (b) If s_0^r is chosen, then add $(I', J') = ([s_0^r, |s_0| - k - |t_0| + 1, |s_0| - |t_0| + 1], [t_0^r, 1, |t_0|])$ to $\mathcal{M}'$.

Observe that $P(I) = P(I')$ and $P(J) = P(J')$ and that the score of (I', J') is the same as (I, J).

For each $(P(I), P(J)) \in \mathcal{P}$, we have added a match (I', J') to $\mathcal{M}'$ such that $\sigma(I', J') = \sigma(P(I), P(J))$. Thus $\mathcal{M}'$ has the same score as $\mathcal{P}$, as desired.

It remains to argue that the matches added to $\mathcal{M}'$ in this second phase cause no conflict. As before, for each $(P(I), P(J))$ considered, we add a match (I', J') with $(P(I'), P(J')) = (P(I), P(J))$. Therefore, if $\mathcal{M}'$ contains two matches that intersect, their projections also intersect, contradicting that $\mathcal{P}$ is conflict-free. It follows no two matches of $\mathcal{M}'$ intersect. As for the other type of conflict, when adding full matches of $s_0 \in S \cup S^r$ into some t_0, we know $\mathcal{P}$ has only this match involving s_0 or s_0^r. Thus, these cannot cause a conflict by matching both some s and s^r. Moreover, we ensure that we add a corresponding match into some $t \in T$, so these cannot match both some t and some t^r. Similarly, when adding a full match of some t_0 into some s_0, we ensure that we match into the chosen element of $S \cup S^r$, preventing conflicts that match s and s^r. Moreover, one such full match involving t_0 can be added since $\mathcal{P}$ is conflict-free, so we never match both t_0 and t_0^r. Hence $\mathcal{M}'$ is conflict-free and has the same score as $\mathcal{P}$, which concludes the proof. □

Lemma 4. *Algorithm 2 constructs a bisimplicial elimination ordering of $G(\mathcal{P})$ in time $O(|\mathcal{P}|^2)$.*

Proof. We first show the correctness of the algorithm, i.e. that the returned sequence B is indeed a bisimplicial elimination ordering, and then we cover the complexity.

Correctness. Let $\mathcal{P}$ be any set of projected matches. First assume that all string intervals of $\mathcal{I}$ are full. Then for any $\{I, J\} \in \mathcal{P}$, its neighbors in $G(\mathcal{P})$ can be partitioned into two cliques: those that intersect with I, and those that intersect with J (this works because since every interval is full, there is no partial intersection). Thus the algorithm is correct in this case.

So assume that $\mathcal{I}$ has non-full string-intervals. Then some early-ender or late-starter must be non-full. It suffices to show that the algorithm always finds an $F = \{I, J\}$ to append to B, and that such an $\{I, J\}$ is bisimplicial in the current $G(\mathcal{P})$.

During one execution of the outer *while* loop, the algorithm assigns a sequence of values $I_1, \ldots, I_h$ to I until it finds some F to add (note that we have not shown that this sequence is finite yet). We argue inductively that for any $k \geq 1$, if F is not found when $I = I_k$, then (1) I_{k+1} is not full; and (2) $|I_k| > |I_{k+1}|$.

We know that I_1 is not full, by the choice of the initial I. We therefore assume inductively that I_k is not full, and show that (1) and (2) hold. Let J_k be the string-interval considered on line 10 (i.e. $\{I_k, J_k\}$ is the match chosen by the algorithm at this point). If J_k is an early-ender or late-starter, the loop will terminate and F will be set. Otherwise, because I_k is not full, J_k is either prefix, suffix or full. This is because if I_k belongs to a prefix or suffix match, then J_k must be prefix or suffix. If I_k is neither prefix or suffix, but not full, then it must be because $\{I_k, J_k\}$ corresponds to a full match, in which case J_k is full.

If J_k is suffix or full, then since J_k is not a late-starter, I_{k+1} is set to a late starter of w, where w is the string involved in J_k. Then, $|I_{k+1}| < |J_k|$ since I_{k+1} starts at a position strictly greater than J_k and ends at a position smaller or equal than J_k (since J_k contains the last character of w). Since $|I_{k+1}| < |J_k| = |I_k|$, we have $|I_k| > |I_{k+1}|$ as desired. Also note that I_{k+1} is not full in this case, since it is strictly smaller than $|w|$. If instead J_k is not suffix or full, then J_k is prefix and I_{k+1} is set to an early-ender of w. Again, I_{k+1} ends earlier than J_k and begins at an equal or greater position, since J_k is prefix. Therefore, in this case it also holds that I_{k+1} is not full and $|I_{k+1}| < |J_k| = |I_k|$.

We have argued that $|I_1| > \ldots > |I_h|$. Thus, the sequence of I intervals considered gets monotonically smaller in size. This implies that the second while loop stops after at most $|\mathcal{P}|$ iterations, and that a sequence B is indeed constructed.

It remains to argue that when some $F = \{I, J\}$ is added to B, it is bisimplicial in the current $G(\mathcal{P})$. That is, notice that the algorithm removes F from $\mathcal{P}$ after adding it to the

output. In the remainder, we refer to $\mathcal{P}$ as the set of matches at the moment that F is added to B. Notice that at any point of the algorithm, I is an early-ender or late-starter. Moreover, when $F = \{I, J\}$ is added to B, J is also an early-ender or late-starter. Let $I' \in \mathcal{I}$ be any string-interval that intersects with I. If I is an early-ender, then I' must contain the last position of I (otherwise, I' would end earlier than I). If I is a late-starter, then I' must contain the first position of I (otherwise, they would start later than I). In either case, this means that all string-intervals that intersect with I are pairwise intersecting since they contain a common position. The same can be said about J. Therefore, the neighbors of $\{I, J\}$ in $G(\mathcal{P})$ can be split into two cliques: the matches that intersect with I, and those that intersect with J. Therefore, $\{I, J\}$ is bisimplicial when added to B.

Complexity. An $O(|\mathcal{P}|^2)$ time can be achieved as follows. Before the start of the algorithm, for each $w \in S \cup T$, we sort all the w-intervals in order of non-decreasing ending position. We also store a copy of the w-intervals in order of non-increasing starting position. This takes $O(|\mathcal{P}| \log |\mathcal{P}|)$ time. We assume that for $I \in \mathcal{I}$, $\mathcal{P}$ provides a pointer to the set of matches that I is part of in $O(1)$ time, and that we can delete a match in $O(1)$ time (e.g. if $\mathcal{P}$ is a hash table of hast tables).

We then argue that each iteration of the main outer loop takes $O(|\mathcal{P}|)$ time. Checking whether every $I \in \mathcal{I}$ if full takes $O(|\mathcal{P}|)$ time. Finding a non-full early-ender/late-starter also takes time $O(|\mathcal{P}|)$. Finding $\{I, J\}$ containing I is $O(1)$, and finding the next I, early-ender or late-starter, takes $O(1)$ time using the sorted lists. As mentioned before, the inner loop runs for at most $|\mathcal{P}|$ iterations, and each takes constant time. Since appending to B and removing from $\mathcal{P}$ take constant time, the running time is bounded by $O(|\mathcal{P}|^2)$. □

Theorem 4. *Algorithm 3 is a 2-approximation to the problem of computing a maximum weight subset of conflict-free matches and runs in time* $O(|\mathcal{P}|^2) = O(|\mathcal{M}|^2)$.

Proof. This can be shown to be a 2-approximation by induction on the recursion depth using standard local ratio arguments. The base case with B empty is trivial. At a higher recursion,

we may assume that the solution $\mathcal{M}'$ returned by the recursion is a 2-approximation with respect to the scores σ_2.

Now let $\mathcal{M}'$ be the set of matches returned by the algorithm. Whether $F \in \mathcal{M}'$ or not, $\mathcal{M}'$ is 2-approximate with respect to σ_2, since adding F does not change its weight (because $\sigma_2(F) = 0$). Consider the weights σ_1. Then, any optimal solution w.r.t. σ_1 includes at most two matches, each of score $\sigma(F)$, since the matches that F intersects with can be partitioned into two cliques. If $F \in \mathcal{M}'$, then $\mathcal{M}'$ has total score $\sigma(F)$ and is a 2-approximation w.r.t. σ_1. If $F \notin \mathcal{M}'$, then $\mathcal{M}'$ contains some F' intersecting with F of score $\sigma(F)$, and it is also a 2-approximation w.r.t. σ_1. To finish the argument, let $OPT \subseteq \mathcal{P}$ (resp. OPT_1, OPT_2) be an optimal solution with respect to σ (resp. σ_1, σ_2). The value of OPT is

$$\begin{aligned}
\sum_{F' \in OPT} \sigma(F') &= \sum_{F' \in OPT} \sigma_1(F') + \sum_{F' \in OPT} \sigma_2(F') \\
&\leq \sum_{F' \in OPT_1} \sigma_1(F') + \sum_{F' \in OPT_2} \sigma_2(F') \\
&\leq 2 \sum_{F' \in \mathcal{M}'} \sigma_1(F') + 2 \sum_{F' \in \mathcal{M}'} \sigma_2(F') \\
&= 2 \sum_{F' \in \mathcal{M}'} \sigma(F')
\end{aligned}$$

Thus $\mathcal{M}'$ is a 2-approximation with respect to σ.

As for the complexity, notice that each call reduces the number of elements of B by 1, and thus the number of calls to $maxMatches$ made is at most the size of B, which is $|\mathcal{P}| = |\mathcal{M}|$. Moreover, each call to $maxMatches$ updates the score of $O(|\mathcal{P}|) = O(|\mathcal{M}|)$ matches, and all other operations can be performed in constant time. If follows that the complexity is $O(|\mathcal{P}|^2) = O(|\mathcal{M}|^2)$. □

Theorem 5. *Denote $n = |S| + |T|$. If $\mathcal{M}$ contains only prefix or suffix matches, then* Maximum String Matching *admits a 2-approximation that runs in time $O(n^2 \log n + n \cdot |\mathcal{M}|)$.*

Moreover, assuming that $\mathcal{M}$ contains a match between each pair of strings in S and T, the running time is $O(|\mathcal{M}|^2)$.

Proof. We have argued that to obtain a 2-approximation, it is sufficient to compute the H graph and compute a maximum weight matching. For the complexity aspect, note that H has $|S| + |T| = n$ vertices and at most $|\mathcal{M}|$ edges. The graph can be constructed in time $O(n + |\mathcal{M}|)$ (note that this includes the $O(|\mathcal{M}|)$ time needed to find the maximum score match between each S-T pair, which can be done in a single traversal of $\mathcal{M}$). A maximum weight matching can then be found in time $O(n^2 \log n + n|\mathcal{M}|)$ using e.g. Fredman & Tarjan's algorithm [FT87].

We can show that this is $O(|\mathcal{M}|^2)$ with a bit more refined analysis. Suppose without loss of generality that $|S| \geq |T|$. In this case, a maximum weight matching can be found in time $O(|T|^2 \log |T| + |T||\mathcal{M}|)$ using modifications presented in [RT12] on the classical matching algorithms. We may further assume that $|\mathcal{M}| \geq |S|$ (and thus $|\mathcal{M}| \geq |T|$), as otherwise we may discard the sequences of S that have no match. Assuming that each pair of strings in S and T have a match, we have $|\mathcal{M}| \geq |S||T| \geq |T|^2$. It is then clear that $|T|^2 \log |T| + |T||\mathcal{M}| \in O(|\mathcal{M}|^2)$. □

References for the Appendix

[FT87] Michael L Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.

[RT12] Lyle Ramshaw and Robert E Tarjan. On minimum-cost assignments in unbalanced bipartite graphs. *HP Labs, Palo Alto, CA, USA, Tech. Rep. HPL-2012-40R1*, 2012.