

The Complexity of Maximal Common Subsequence Enumeration

GIOVANNI BUZZEGA, University of Pisa, Italy

ALESSIO CONTE, University of Pisa, Italy

YASUAKI KOBAYASHI, Hokkaido University, Japan

KAZUHIRO KURITA, Nagoya University, Japan

GIULIA PUNZI, University of Pisa, Italy

Frequent pattern mining is widely used to find “important” or “interesting” patterns in data. While it is not easy to mathematically define such patterns, maximal frequent patterns are promising candidates, as frequency is a natural indicator of relevance and maximality helps to summarize the output. As such, their mining has been studied on various data types, including itemsets, graphs, and strings. The complexity of mining maximal frequent itemsets and subtrees has been thoroughly investigated (e.g., [Boros et al., 2003], [Uno et al., 2004]) in the literature. On the other hand, while the idea of mining frequent subsequences in sequential data was already introduced in the seminal paper [Agrawal et al., 1995], the complexity of the problem is still open.

In this paper, we investigate the complexity of the maximal common subsequence enumeration problem, which is both an important special case of maximal frequent subsequence mining and a generalization of the classic longest common subsequence (LCS) problem. We show the hardness of enumerating maximal common subsequences between multiple strings, ruling out the possibility of an *output-polynomial time* enumeration algorithm under $P \neq NP$, that is, an algorithm that runs in time $\text{poly}(|I| + N)$, where $|I|$ and N are the size of the input and number of output solutions, respectively. To circumvent this intractability, we also investigate the parameterized complexity of the problem, and show several results when the alphabet size, the number of strings, and the length of a string are taken into account as parameters.

CCS Concepts: • **Mathematics of computing** → **Enumeration**; • **Theory of computation** → **Complexity classes**.

Additional Key Words and Phrases: Maximal frequent subsequence mining, Maximal common subsequence enumeration, Output-sensitive complexity, NP-hardness of enumeration problems

ACM Reference Format:

Giovanni Buzzega, Alessio Conte, Yasuaki Kobayashi, Kazuhiro Kurita, and Giulia Punzi. 2025. The Complexity of Maximal Common Subsequence Enumeration. *Proc. ACM Manag. Data* 3, 2 (PODS), Article 115 (May 2025), 19 pages. <https://doi.org/10.1145/3725252>

1 Introduction

Frequent pattern mining problems are a central topic in data mining. Mining algorithms typically aim to find “interesting” or “important” patterns, but mathematically defining an interesting or important pattern is not easy. A reasonable implementation of this concept, that has been widely studied, is using frequency as an indicator of importance, and enumerating all high-frequency patterns. In particular, structures such as itemsets, graphs, and strings are often sought after [2, 32, 39].

Authors’ Contact Information: Giovanni Buzzega, University of Pisa, Pisa, Italy, giovanni.buzzega@phd.unipi.it; Alessio Conte, University of Pisa, Pisa, Italy, alessio.conte@unipi.it; Yasuaki Kobayashi, Hokkaido University, Sapporo, Japan, koba@ist.hokudai.ac.jp; Kazuhiro Kurita, Nagoya University, Nagoya, Japan, kurita@i.nagoya-u.ac.jp; Giulia Punzi, University of Pisa, Pisa, Italy, giulia.punzi@unipi.it.



This work is licensed under a Creative Commons Attribution 4.0 International License.

© 2025 Copyright held by the owner/author(s).

ACM 2836-6573/2025/5-ART115

<https://doi.org/10.1145/3725252>

The fundamental problem of *frequent itemset mining*, for example, can be formulated as follows. Given a transaction database $\mathcal{T} = \{T_1, \dots, T_k\}$ and a threshold t , enumerate all sets V that are frequent, i.e., contained in at least t transactions in $\mathcal{T}$. When the database is a collection of graphs, the problem is called *frequent subgraph mining* [25, 28], and when it is a collection of strings, it is called *frequent subsequence mining*, where a *subsequence* of a string S is a string that can be obtained by deleting arbitrary occurrences of characters of S . In the frequent subsequence mining problem, each element of a sequence can sometimes be defined as a subset of a given itemset. In this paper, we consider the less general case in which each element of a sequence is a simple character.

Frequency, however, is not the end of the story, as high-frequency patterns are not always all useful. For example, a frequent pattern that is simply a subset of another frequent pattern may not be considered useful, as its information is subsumed by the bigger pattern. Thus, research in the field focused on additional constraints for pruning redundant information, such as closedness and maximality: a frequent pattern is called *maximal* if no other frequent pattern strictly contains it, and it is called *closed* if it is only contained in patterns with lower frequency than itself. Discarding non-maximal or non-closed patterns greatly reduces the number of patterns, with minimal loss of information, but comes at a computational cost, as enumerating just the maximal or closed patterns can be significantly more challenging.

To analyze this cost, we typically have to look at the *output size*: the number of solutions may be exponential in the size of the input, so we seek algorithms that run, at the very least, in time polynomial in the output size [41]. We call these algorithms *output-polynomial*, or *output-sensitive*.

For the itemset mining problem, the output-sensitive enumeration of maximal/closed frequent itemset has been thoroughly studied [7, 34]. For closed frequent itemset mining, Uno et al. [34] proposed a linear-delay enumeration algorithm, where the delay is the maximum time elapsed between outputs. In contrast, the enumeration of maximal frequent itemsets is known to be intractable [7]. More precisely, Boros et al. showed that the decision problem of determining, given a set of maximal frequent itemsets, whether there exists another one or not, is NP-complete. This hardness result implies that no output-polynomial time algorithm for enumerating maximal frequent itemsets can exist unless $P = NP$.

In the case of subgraph mining, computing frequency often leads to NP-complete graph problems, e.g., maximum clique, subgraph isomorphism, Hamiltonian path, and so on [24, 25]. Moreover, the maximal/closed frequent subgraph mining is intractable even if the graphs are two trees [28]. Thus, the maximal/closed frequent subgraph mining is hard even when frequency can be computed in polynomial time.

As for maximal/closed frequent *subsequence* mining, due to its countless applications in domains such as text analysis or biological data, many practical algorithms have been developed [4, 19, 33, 37, 38]. Surprisingly, however, there are still no results on its theoretical complexity.

In this paper, we seek to characterize this complexity, and to do so we start from the problem of enumerating maximal common subsequences, hereafter MCS ENUMERATION. For a set of strings $\mathcal{S} = \{S_1, \dots, S_k\}$ and a string S , the *frequency of S* is the number of strings in $\mathcal{S}$ that contain S as a subsequence. MCS ENUMERATION asks for every string S whose frequency is exactly k , i.e., is common to all strings in $\mathcal{S}$, and that is maximal in the sense that S is not a subsequence of any other common subsequence of $\mathcal{S}$.

MCSs are a more general variant of the classic longest common subsequence (LCS) problem, as LCSs are simply MCSs of maximum length, and their complexity was thoroughly investigated [1]. In turn, MCSs are a special case of the *closed* frequent subsequences, that are all the strings S whose superstrings have lower frequency than S , and its complexity remains open.

The main result of this paper is a negative one, as we prove that no efficient enumeration algorithm exists for MCS ENUMERATION unless $P = NP$. Moreover, we investigate restricted instances of the

problem, as well as how this complexity affects related problems such as counting, assessment and indexing. A summary of the problems studied and our results is given in Section 1.2.

1.1 Enumeration, counting, assessment, and indexing

Enumeration problems are related to, but still fundamentally different from, counting problems. Both problems concern a specific pattern of which we wish to identify occurrences in a given input instance. The counting problem is tasked with simply returning *the number* of such occurrences, while the enumeration problem has the aim of *outputting* all occurrences of the pattern. When addressing an enumeration problem, the total number of solutions can be exponentially larger than the input size, and therefore, simply outputting the solution requires exponential time [40]. This is also the case for the problem of enumerating MCSs, which is the focus of this paper. To address this inherent problem of exponential complexity, we can take two different approaches.

Indeed, while it is clear that enumeration also provides an answer to the counting problem, there are cases where counting can be performed faster. Even if enumeration intrinsically provides more information than counting, in some data mining applications it is sometimes sufficient to know the number of solutions, instead of finding them all [31]. Unfortunately, counting is also often hard to perform efficiently. For instance, both the problem of counting all frequent itemsets [20] and the aforementioned maximal frequent itemsets/subsequences/subgraphs problems [40] have been shown to be #P-complete, meaning that no polynomial-time counting algorithm exists, unless $P = NP$. There is also a further intermediate step between counting and enumeration, called assessment: the problem of determining whether there are more solutions than a given integer threshold z . When z is not too large, this type of problem can be efficiently solved by using an enumeration algorithm. However, there are several problems where assessment is NP-hard even when the threshold z is small [8, 9, 27]. For example, it is known that the problem of determining whether there are more maximal frequent itemsets than a given integer z is NP-complete [7].

Another alternative approach to circumvent the exponential explosion of direct enumeration is to compute an index that stores the solutions [10, 21, 29, 30, 33], which can be seen as an “implicit” form of enumeration. This approach aims to construct a small index, defining operations to efficiently handle several useful queries, such as membership queries, random access and (random order) enumeration of solutions [11]. Such an index can take the form of one of several data structures, like labeled Directed Acyclic Graphs (DAGs), variants of decision diagrams, and d -DNNF [14, 15, 22]. If an enumeration problem admits a small index that can be quickly computed, then the increase in computation time due to output size can be disregarded as we can still perform practically fast enumeration. For our problem of MCS ENUMERATION, several such indices are known when the input strings are two: two of them of provably polynomial size [13, 22], and more recently a practically efficient one, with no worst-case guarantees [10]. It is natural to ask whether these results can be generalized to an index for k strings.

The complexity of enumeration algorithms and related problems. As mentioned above, the complexity of enumeration is usually expressed with an output-sensitive analysis [26]. Consider an instance I of an enumeration problem, of size $|I|$, and let N be the number of solutions to output. An enumeration algorithm is called *output-polynomial* if it runs in $O(\text{poly}(|I| + N))$ time. Moreover, an algorithm is called *quasi-polynomial* if it runs in $O((|I|)^{\text{poly}(\log(|I|))})$ time, and *output-quasi-polynomial* if it runs in $O((|I| + N)^{\text{poly}(\log(|I| + N))})$ time.

If an enumeration algorithm runs in $O(N)$ total time, we say that algorithm takes *Constant Amortized Time* (CAT). Finally, we sometimes measure the *delay* of an enumeration algorithm, that is the the maximum computation time elapsed between two consecutive outputs. An algorithm whose

Problem	General	$ \Sigma = 2$	$k = 2$	$k = O(1)$	$n = O(1)$
ANOTHER MCS	NP-C even if $ \mathcal{Z} = O(n)$	-	P [13]	P	P
MCS ASSESSMENT for $z = \text{poly}(n)$	NP-H even if $z = O(n)$	-	P [13]	P	P
MCS COUNTING	#P-C	#P-C	P [13]	-	P
MCS ENUMERATION	No OP alg.	MIS-H	CAT [13]	P-delay	P
MCS INDEXING	No OP alg.	MIS-H	P [13]	-	P

Table 1. Summary of our results on the complexity of MCS enumeration and related problems, where k is the number of input strings and n their maximum length. NP-H, NP-C and #P-C are short for NP-hard, NP-complete and #P-Complete, respectively. No OP alg. means no Output-Polynomial time algorithm exists unless $P = NP$. MIS-H means the problem is at least as hard as MIS ENUMERATION. The complexity of problems marked with ‘-’ is open. Results for $k = 2$ are included for completeness, but follow from [13]. Results for $n = O(1)$ can be generalized up to values of n polylogarithmic in the input size, but for readability the discussion is only given in Section 5.

delay is $O(\text{poly}(|\mathcal{I}|))$ is called *polynomial-delay*, and the algorithm is clearly output-polynomial since its total running time is $O(N \cdot \text{poly}(|\mathcal{I}|))$.

We also evaluate the efficiency of computing an index, i.e., any compact data structure that allows efficient membership test and enumeration on the solutions.¹ Let $\mathcal{D}$ be an index that stores all solutions of an instance of an enumeration problem. Classic examples are indices that can be represented with edge-labeled graphs, such as tries, labeled DAGs, and SeqBDD/ZDD, where the size $|\mathcal{D}|$ is the sum of the number of vertices and edges. Such indices are particularly interesting when they can be computed in just polynomial time.

Finally, we study and evaluate the complexity of assessment and counting problems. The analogous of NP-hardness (resp. NP-completeness) for counting problems is given by #P-hardness (resp. #P-completeness) [36]: a #P-hard problem cannot have a polynomial-time counting algorithm, unless $P = NP$. To be able to gain insight on such hard problems, assessment, which can be seen as a “partial” form of counting, is a valid tool. Indeed, even #P-complete problems may allow efficient assessment algorithms for polynomial values of z [31]. We note that if a counting problem is polynomial-time, then assessment is trivially polynomial-time as well. On the other hand, if the counting problem is #P-complete, we cannot perform assessment in time polynomial in the size of the input, as a binary search on the threshold z would allow us to solve the original counting problem in polynomial time as well. Thus, an assessment algorithm is often considered efficient if it runs in $O(\text{poly}(|\mathcal{I}| + z))$ time. Since z can be exponential in the size of the input, the strongest hardness results are the ones where we assume $z = O(|\mathcal{I}|)$.

1.2 Problem definitions and results

In this work, we investigate the complexity of MCS ENUMERATION, as well as related problems on MCS, with the aim of giving a complete picture of which problems are tractable, and which are not. In the following, for a given set of strings $\mathcal{S}$, we denote with $\|\mathcal{S}\|$ its total size, i.e. the sum of the length of the strings in $\mathcal{S}$.

Formally, the MCS enumeration problem can be stated as follows:

¹Indices can provide more refined operations like random access and random order enumeration [11], but at least membership and enumeration should be guaranteed.

Problem 1 (MCS ENUMERATION). *Given a set $\mathcal{S}$ of strings, the task of MCS ENUMERATION is to output all maximal common subsequences of $\mathcal{S}$.*

As our main result, we show that there is no output-polynomial-time algorithm for MCS ENUMERATION in multiple strings, unless $P = NP$. To show this hardness, we start from the following decision problem:

Problem 2 (ANOTHER MCS). *Given a set $\mathcal{S}$ of strings and a set $\mathcal{Z}$ of maximal common subsequences of $\mathcal{S}$, ANOTHER MCS asks whether there exists a maximal common subsequence of $\mathcal{S}$ that is not contained in $\mathcal{Z}$.*

In Section 3, we prove the hardness of ANOTHER MCS through a reduction from 3-SAT, yielding:

THEOREM 1.1. *ANOTHER MCS is NP-complete, even for instances where $|\mathcal{Z}| = O(n)$, where n is the maximum length of an input string.*

This type of *another solution* problems (also called finished decision problems [6], or additional problems [3]) are frequently used to show the hardness of the corresponding enumeration problems. This implication is folklore [6–8, 27, 28]; intuitively, if ANOTHER MCS has negative answer (and thus $\mathcal{Z}$ is the whole set of MCS), then any output-sensitive enumeration algorithm for MCS would terminate in time $O(\text{poly}(|\mathcal{Z}|))$. More details concerning this are given in Section 1.3. Thus, the NP-hardness of ANOTHER MCS implies the following hardness result for MCS ENUMERATION:

Corollary 1. *There is no output-polynomial time algorithm for MCS ENUMERATION, unless $P = NP$.*

We also focus on the task of creating an index that stores all MCSs, which we call MCS INDEXING. Specifically, we are concerned with indexes that allow us to perform efficient membership queries and enumeration for the MCSs. Examples of such indexes are tries, labeled DAGs and SeqBDD/ZDD. The complexity of this problem has already been studied in the literature [13, 22] for the case of $k = 2$ strings, we here study the general case:

Problem 3 (MCS INDEXING). *MCS INDEXING asks, given any input set of strings $\mathcal{S}$, to output an index $\mathcal{D}$ in time and space $O(\text{poly}(|\mathcal{S}|))$, which stores the set of maximal common subsequences of $\mathcal{S}$, allowing output-polynomial enumeration and polynomial-time membership testing of the maximal common subsequences in $\mathcal{S}$.*

Corollary 1 immediately implies the hardness of MCS INDEXING: if we can construct such an index, then we can enumerate all MCSs in output-polynomial time. Thus, we arrive at the following:

Corollary 2. *There is no output-polynomial time algorithm for MCS INDEXING unless $P = NP$.*

To give a complete picture, we also investigate other problems related to enumeration and mining, namely assessment and counting. In the MCS setting, the assessment problem is formally stated as:

Problem 4 (MCS ASSESSMENT). *Given a set $\mathcal{S}$ of strings and an integer z , MCS ASSESSMENT asks whether or not the number of maximal common subsequences in $\mathcal{S}$ is more than z .*

For this problem, Theorem 1.1 directly implies that it is NP-hard to even determine whether the number of MCSs exceeds the maximum length n of the input strings, leading to:

Corollary 3. *MCS ASSESSMENT is NP-Hard, even if $z = O(n)$ where n is the maximum length of an input string.*

The final problem we address is the problem of MCS counting, formally defined as:

Problem 5 (MCS COUNTING). *Given a set $\mathcal{S}$ of strings, MCS COUNTING asks for the number of maximal common subsequences in $\mathcal{S}$.*

In this regard, we dedicate Section 4 to showing a strong direct link between MCSs and maximal independent sets in hypergraphs, in the form of a one-to-one reduction from the problem of enumerating maximal independent sets in hypergraphs (MIS ENUMERATION) to MCS ENUMERATION.

THEOREM 1.2. *Given a hypergraph $\mathcal{H} = (V, \mathcal{E})$, there exists a set of binary strings $\mathcal{S}(\mathcal{H})$, computable in time polynomial in $\|\mathcal{H}\|$, and a bijection between maximal independent sets in $\mathcal{H}$ and the set $\text{MCS}(\mathcal{S}(\mathcal{H})) \setminus \{w\}$, where the string w consists of $|V| - 1$ repetition of the string “01”.*

Notably, the reduction holds even when the strings are constrained to use a binary alphabet. MIS ENUMERATION (also called the minimal transversal enumeration, the minimal set cover enumeration, and the minimal hitting set enumeration) is a long-standing open problem in enumeration algorithm area [17]. The existence of an output-polynomial enumeration algorithm for MIS ENUMERATION has long been questioned, thus we call MIS-H the class of problems that can be reduced to MIS ENUMERATION, i.e., that are at least as hard.

The reduction has several implications; first of all, designing an output-polynomial-time enumeration algorithm for MCS ENUMERATION is challenging even if the alphabet is binary:

Corollary 4. *If MCS ENUMERATION for binary strings can be solved in output-polynomial time, MIS ENUMERATION can be solved in output-polynomial time.*

Similarly as before, the hardness result for indexing follows:

Corollary 5. *If MCS INDEXING has an output-polynomial time algorithm, then MIS ENUMERATION in hypergraphs can be solved in output-polynomial time.*

Moreover, while the exact complexity of MIS ENUMERATION is open, MIS COUNTING is known to be #P-complete, even in the restricted case of simple graphs (i.e., hypergraphs where all hyperedges have size 2) [35]. The above bijection implies that any algorithm for MCS COUNTING can solve MIS COUNTING, so we obtain the following:

Corollary 6. *MCS COUNTING is #P-complete, even on binary strings.*

Given the previous negative results, one is left to wonder: are there tractable instances of MCS ENUMERATION and its related problems? As the alphabet size does not seem promising in this regard, we investigate the remaining parameters n (the length of the strings) and k (the number of strings), and finally present two positive results in Section 5.

As a first positive result, Section 5.1 shows that bounding the length l of the *shortest* input string already yields an efficient solution to MCS ENUMERATION.²

THEOREM 1.3. *MCS ENUMERATION can be solved in FPT total time with respect to l . Specifically, when l is logarithmic in $\|\mathcal{S}\|$, all MCSs can be enumerated in polynomial time in n and k , and when l is polylogarithmic in $\|\mathcal{S}\|$, enumeration takes quasi-polynomial time in n and k .*

Secondly, MCS ENUMERATION has been shown to be output-sensitive for the special case of two input strings, i.e., $k = 2$ [12]. In Section 5.2 we generalize the algorithm of [12] showing that output-polynomial enumeration is possible even when $k = O(1)$.

THEOREM 1.4. *MCS ENUMERATION for $k \geq 2$ strings of maximum length n can be solved in XP-delay with respect to k . Specifically, MCSs can be enumerated in $O(kn^{2k+1})$ time delay, after $O(k|\Sigma|n^k)$ time preprocessing.*

We summarize both the negative and the positive results in Table 1.

²Clearly, $l \leq n$, so this is strictly stronger than a parametrization in n .

1.3 MCS ENUMERATION and MCS INDEXING are at least as hard as ANOTHER MCS

In this section, we give a brief overview of the relationship between *another solution* problems (also called finished decision problems [6] and additional problems [3]) and enumeration problems. Intuitively, to solve an enumeration problem, we can repeatedly solve its *another solution* version until all solutions are found. In this section we formally show that the NP-hardness of the another solution problem indeed rules out an output-polynomial time algorithm for the enumeration problem, assuming $P \neq NP$. Although a similar discussion has already been used in several papers [6–8, 27, 28], we retrace it here in order to make our paper self-contained. For the sake of simplicity, here we only focus on MCS ENUMERATION and ANOTHER MCS.

THEOREM 1.5. *If there exists an output-polynomial time algorithm for MCS ENUMERATION, then ANOTHER MCS can be solved in polynomial time.*

PROOF. Let $(S, \mathcal{Z})$ be an instance of ANOTHER MCS and $\mathcal{A}$ be an output-polynomial time algorithm for MCS ENUMERATION. Since $\mathcal{A}$ runs in output-polynomial time, the running time of $\mathcal{A}$ is bounded by $(\|S\| + \|\mathcal{M}(S)\|)^c$, where c is some constant and $\mathcal{M}(S)$ is the set of maximal common subsequences of S . We run $\mathcal{A}$ for $(\|S\| + \|\mathcal{Z}\|)^c$ steps. If $\mathcal{A}$ terminates, we obtain $\mathcal{M}(S)$, and it is easy to determine whether $\mathcal{M}(S)$ contains a maximal common subsequence not contained in $\mathcal{Z}$. If $\mathcal{A}$ does not terminate within $(\|S\| + \|\mathcal{Z}\|)^c$ steps, it implies that $|\mathcal{M}(S)|$ is larger than $|\mathcal{Z}|$. Therefore, $\mathcal{Z} \subset \mathcal{M}(S)$ and there is a maximal common subsequence not contained in $\mathcal{Z}$. In both cases, we can solve ANOTHER MCS in $(\|S\| + \|\mathcal{Z}\|)^c$ time. $\square$

2 Preliminaries

Let Σ be an *alphabet*. An element in Σ^* is a *string*. If $|\Sigma| = 2$, an element in Σ^* is called a *binary string*. For a string X , we denote the length of X as $|X|$. The string with length zero is called an *empty string*. We denote the empty string as ε .

For strings X and Y , $X \cdot Y$ denotes the concatenation of two strings. As a shorthand notation, we may use XY instead of $X \cdot Y$. For a string $W = XY$, X is a *prefix* of W . The i -th symbol of a string X is denoted as $X[i]$, where $1 \leq i \leq |X|$. For a string X and two integers $1 \leq i \leq j \leq |X|$, $X[i, j]$ denotes the *substring* of X that begins at position i and ends at position j . For a string X and an integer k , X^k is the string obtained by repeating X k -times. For two strings X and Y , Y is a *subsequence* of X if there is an increasing sequence of integers $(\delta_1, \dots, \delta_{|Y|})$ such that $\delta_1 < \dots < \delta_{|Y|}$, and $X[\delta_1]X[\delta_2] \dots X[\delta_{|Y|}]$ equals Y . Moreover, we call such integer sequence $(\delta_1, \dots, \delta_{|Y|})$ an *arrangement* of Y for X . In addition, we call that X is a *supersequence* of Y or X contains Y as a subsequence.

For a set of k strings $\mathcal{S} = \{S_1, \dots, S_k\}$, we denote with $\|\mathcal{S}\|$ the size of $\mathcal{S}$: $\|\mathcal{S}\| = \sum_{i=1}^k |S_i|$. We say that a string X is a *common subsequence* of $\mathcal{S}$ if for any $1 \leq i \leq k$, S_i contains X as a subsequence. If any supersequence of X is not common subsequence of $\mathcal{S}$, X is an *MCS* of $\mathcal{S}$.

3 NP-Hardness of ANOTHER MCS

In this section, we focus on proving Theorem 1.1. From this, the results of Corollaries 1-3 will immediately follow. To prove the theorem, we give a reduction from 3-SAT to ANOTHER MCS, creating an instance of the latter where set $\mathcal{Z}$ has linear size.

Let ϕ be a formula in 3-Conjunctive Normal Form (CNF), i.e., composed of conjunctions of clauses, where each clause is made of a disjunction of exactly three literals. We denote the total number of variables and the number of clauses as v and m , respectively. Without loss of generality, we assume that no clause contains both x_i and $\bar{x}_i$, and no variable appears in every clause (as otherwise, we can split the problem into two easy 2-SAT instances). Formally, we consider $\phi = \bigwedge_{i=1}^m C_i$,

where $C_i = \ell_1^{(i)} \vee \ell_2^{(i)} \vee \ell_3^{(i)}$ with $\ell_1^{(i)} \in \{x_\alpha, \bar{x}_\alpha\}$, $\ell_2^{(i)} \in \{x_\beta, \bar{x}_\beta\}$, and $\ell_3^{(i)} \in \{x_\gamma, \bar{x}_\gamma\}$ for some $1 \leq \alpha < \beta < \gamma \leq v$.

We construct the corresponding instance $(\mathcal{S}(\phi), \mathcal{Z}(\phi))$ of ANOTHER MCS as follows. For each clause C_i in ϕ , we define a string $S_i = R^{\alpha-1} \ell_1^{(i)} R^{\beta-\alpha} \ell_2^{(i)} R^{\gamma-\beta} \ell_3^{(i)} R^{v-\gamma}$, where $R = x_v \bar{x}_v \dots x_1 \bar{x}_1$. Moreover, we define S_0 as $x_1 \bar{x}_1 \dots x_v \bar{x}_v$. The set of strings $\mathcal{S}(\phi)$ is defined as $\{S_0, \dots, S_m\}$. For each $1 \leq i \leq v$, we let $Z_i = x_1 \bar{x}_1 \dots x_{i-1} \bar{x}_{i-1} x_{i+1} \bar{x}_{i+1} \dots x_v \bar{x}_v$, and define $\mathcal{Z}(\phi)$ as $\{Z_1, \dots, Z_v\}$. In other words, Z_i is the string obtained by removing $\bar{x}_i x_i$ from S_0 . It is clear that, for each 3-SAT instance ϕ , this reduction constructs a corresponding instance of ANOTHER MCS in polynomial time. Furthermore, note that $|\mathcal{Z}(\phi)| = v$ is linear in the size of the input. We denote the instance $(\mathcal{S}(\phi), \mathcal{Z}(\phi))$ as $\text{MCS}(\phi)$.

Let us give an example of our reduction: let $\phi = (x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (x_2 \vee \bar{x}_3 \vee \bar{x}_4) \wedge (\bar{x}_1 \vee x_3 \vee x_4)$. Then, the set $\mathcal{S}(\phi)$ is defined as follows (parentheses are added for readability, but they are not part of the strings):

$$\begin{aligned} S_0 &= x_1 \bar{x}_1 x_2 \bar{x}_2 x_3 \bar{x}_3 x_4 \bar{x}_4, \\ S_1 &= x_1 (x_4 \bar{x}_4 x_3 \bar{x}_3 x_2 \bar{x}_2 x_1 \bar{x}_1) \bar{x}_2 (x_4 \bar{x}_4 x_3 \bar{x}_3 x_2 \bar{x}_2 x_1 \bar{x}_1) x_3 (x_4 \bar{x}_4 x_3 \bar{x}_3 x_2 \bar{x}_2 x_1 \bar{x}_1), \\ S_2 &= (x_4 \bar{x}_4 x_3 \bar{x}_3 x_2 \bar{x}_2 x_1 \bar{x}_1) x_2 (x_4 \bar{x}_4 x_3 \bar{x}_3 x_2 \bar{x}_2 x_1 \bar{x}_1) \bar{x}_3 (x_4 \bar{x}_4 x_3 \bar{x}_3 x_2 \bar{x}_2 x_1 \bar{x}_1) \bar{x}_4, \text{ and} \\ S_3 &= \bar{x}_1 (x_4 \bar{x}_4 x_3 \bar{x}_3 x_2 \bar{x}_2 x_1 \bar{x}_1) (x_4 \bar{x}_4 x_3 \bar{x}_3 x_2 \bar{x}_2 x_1 \bar{x}_1) x_3 (x_4 \bar{x}_4 x_3 \bar{x}_3 x_2 \bar{x}_2 x_1 \bar{x}_1) x_4, \end{aligned}$$

and the set $\mathcal{Z}(\phi)$ is defined as follows:

$$\begin{aligned} Z_1 &= x_2 \bar{x}_2 x_3 \bar{x}_3 x_4 \bar{x}_4, \\ Z_2 &= x_1 \bar{x}_1 x_3 \bar{x}_3 x_4 \bar{x}_4, \\ Z_3 &= x_1 \bar{x}_1 x_2 \bar{x}_2 x_4 \bar{x}_4, \text{ and} \\ Z_4 &= x_1 \bar{x}_1 x_2 \bar{x}_2 x_3 \bar{x}_3. \end{aligned}$$

Notice that, in this example, $x_1 x_2 x_3 x_4 \bar{x}_4$ is an MCS of $\mathcal{S}(\phi)$, and ϕ is satisfied by the assignment $(x_1, x_2, x_3, \bar{x}_4) = (\text{True}, \text{True}, \text{True}, \text{False})$.

We first show that our reduction generates a valid instance of ANOTHER MCS. More precisely, we show that each string in $\mathcal{Z}(\phi)$ is an MCS of $\mathcal{S}(\phi)$. To this end, the following observation is trivial but helpful.

Observation 1. *Let S and T be two strings. Fix a value $1 \leq j \leq |T|$ and let i be an index such that $T[1, j]$ is a subsequence of $S[1, i]$, but $T[1, j+1]$ is not. Then, T is a subsequence of S if and only if $T[j+1, |T|]$ is a subsequence of $S[i+1, |S|]$.*

PROOF. If $T[j+1, |T|]$ is a subsequence of $S[i+1, |S|]$, and $T[1, j]$ is a subsequence of $S[1, i]$, then T is a subsequence of S .

Let us now show the opposite direction. Since T is a subsequence of S , it has an arrangement $(\delta_1, \dots, \delta_{|T|})$. Since $T[1, j+1]$ is not a subsequence of $S[1, i]$, we have that $\delta_{j+1} > i$. This means that $T[j+1, |T|]$ is a subsequence of $S[i+1, |S|]$ with $(\delta_{j+1}, \dots, \delta_{|T|})$ as an arrangement. $\square$

Now, we are ready to show that our reduction is valid.

Lemma 1. *Let $\phi = C_1 \wedge \dots \wedge C_m$ be a 3-SAT formula. Then, each $Z_i \in \mathcal{Z}(\phi)$ is an MCS of $\mathcal{S}(\phi)$.*

PROOF. From the construction of S_0 , Z_i is a subsequence of S_0 . For each $1 \leq j \leq m$, S_j contains Z_i as a subsequence since S_j contains R^{v-1} , where $R = x_v \bar{x}_v \dots x_1 \bar{x}_1$.

We show each Z_i is maximal by contradiction. Let $Z' \neq Z_i$ be a common subsequence of $\mathcal{S}(\phi)$ that contains Z_i as a subsequence. Since Z' is a subsequence of S_0 , it is of the form $Z' = x_1 \bar{x}_1 \dots x_{i-1} \bar{x}_{i-1} X x_{i+1} \bar{x}_{i+1} \dots x_v \bar{x}_v$, where X is a subsequence of $x_i \bar{x}_i$. Since each S_j does not contain S_0 as a subsequence, $X \neq x_i \bar{x}_i$. Therefore, suppose, without loss of generality, that $X = \bar{x}_i$. Since there is no variable that appears in every clause, we can also assume that no literal appears in every

clause; thus, there is a clause $C_j = \ell_1^{(j)} \vee \ell_2^{(j)} \vee \ell_3^{(j)}$ such that $\ell_1^{(j)}, \ell_2^{(j)}, \ell_3^{(j)} \notin \{x_i, \bar{x}_i\}$. We show that Z' is not a subsequence of S_j . Let α, β and γ be the integers with $\alpha < \beta < \gamma$ such that $\ell_1^{(j)} \in \{x_\alpha, \bar{x}_\alpha\}$, $\ell_2^{(j)} \in \{x_\beta, \bar{x}_\beta\}$, and $\ell_3^{(j)} \in \{x_\gamma, \bar{x}_\gamma\}$. Suppose that $i < \alpha$. By Observation 1, since $Z'[1, 2(i-1)+1] = x_1 \bar{x}_1 \dots x_{i-1} \bar{x}_{i-1} \bar{x}_i$ is a subsequence of R^i , but $Z'[1, 2(i-1)+2]$ is not, we have that Z' is a subsequence of S_j if and only if $x_{i+1} \bar{x}_{i+1}, \dots, x_v \bar{x}_v$ is a subsequence of $R^{\alpha-i-1} \ell_1^{(j)} R^{\beta-\alpha} \ell_2^{(j)} R^{\gamma-\beta} \ell_3^{(j)} R^{v-\gamma}$. Since the number of repetitions of R is $v-i-1$, $R^{\alpha-i-1} \ell_1^{(j)} R^{\beta-\alpha} \ell_2^{(j)} R^{\gamma-\beta} \ell_3^{(j)} R^{v-\gamma}$ does not contain $x_{i+1} \bar{x}_{i+1}, \dots, x_v \bar{x}_v$ as a subsequence. Suppose that $\alpha \leq i < \beta$. By applying the same approach, Z' is a subsequence of S_j if and only if $x_\alpha \bar{x}_\alpha \dots X x_{i+1} \bar{x}_{i+1} \dots x_v \bar{x}_v$ is a subsequence of $\ell_1^{(j)} R^{\beta-\alpha} \ell_2^{(j)} R^{\gamma-\beta} \ell_3^{(j)} R^{v-\gamma}$. Since $\ell_1^{(j)} \neq x_\alpha \bar{x}_\alpha$, removing $x_\alpha \bar{x}_\alpha$ and $\ell_1^{(j)}$ does not change the subsequence relationship. Thereafter, by the same argument as for $i < \alpha$, Z' is not a subsequence of S_j . From the same discussion in other cases, Z' is not a subsequence of S_j , regardless of i . $\square$

Lemma 1 proves that $\text{MCS}(\phi)$ is a valid instance of ANOTHER MCS. We next show that there is a common subsequence of $\mathcal{S}(\phi)$ that is not included in $\mathcal{Z}(\phi)$ if and only if ϕ is satisfiable, which needs a preliminary lemma.

Lemma 2. *Let X be a subsequence of S_0 that contains exactly one of x_j or $\bar{x}_j$ for each $1 \leq j \leq v$. Let $1 \leq i \leq m$ and $C_i = \ell_1^{(i)} \vee \ell_2^{(i)} \vee \ell_3^{(i)}$. Then, X is a subsequence of S_i if and only if X does not contain $\bar{\ell}_1^{(i)} \bar{\ell}_2^{(i)} \bar{\ell}_3^{(i)}$ as a subsequence.*

PROOF. Let α, β , and γ be the integers with $\alpha < \beta < \gamma$ that satisfy $\ell_1^{(i)} \in \{x_\alpha, \bar{x}_\alpha\}$, $\ell_2^{(i)} \in \{x_\beta, \bar{x}_\beta\}$, and $\ell_3^{(i)} \in \{x_\gamma, \bar{x}_\gamma\}$. Observe that $S_i[1 + 2v(\alpha-1)] = \ell_1^{(i)}$, $S_i[2 + 2v(\beta-1)] = \ell_2^{(i)}$, and $S_i[3 + 2v(\gamma-1)] = \ell_3^{(i)}$.

Suppose that X does not contain $\bar{\ell}_1^{(i)} \bar{\ell}_2^{(i)} \bar{\ell}_3^{(i)}$ as a subsequence, that is, X contains one of $\ell_1^{(i)}$, $\ell_2^{(i)}$, or $\ell_3^{(i)}$. Suppose that $X[\omega] = \ell_j^{(i)}$ for some $(j, \omega) \in \{(1, \alpha), (2, \beta), (3, \gamma)\}$. Since S_i contains $R^{\omega-1} \ell_j^{(i)} R^{v-\omega}$, and R contains the whole alphabet, X is a subsequence of S_i .

We show the opposite direction. Suppose by contradiction that $\bar{\ell}_1^{(i)} \bar{\ell}_2^{(i)} \bar{\ell}_3^{(i)}$ is a subsequence of X . By Observation 1, X is a subsequence of S_i if and only if $X[\alpha, |X|]$ is a subsequence of $S_i[1 + 2v(\alpha-1), |S_i|]$, as $X[1, \alpha-1]$ is a subsequence of $S_i[1, 2v(\alpha-1)] = R^{\alpha-1}$ but $X[1, \alpha]$ is not. As $X[\alpha] \neq S_i[1 + 2v(\alpha-1)]$ and $X[\beta] \neq S_i[2 + 2v(\beta-1)]$, we can repeat the same argument: X is a subsequence of S_i if and only if $X[\beta, |X|]$ is a subsequence of $S_i[2 + 2v(\beta-1), |S_i|]$ which holds if and only if $X[\gamma, |X|]$ is a subsequence of $S_i[3 + 2v(\gamma-1), |S_i|]$. However, $X[\gamma, |X|]$ is not a subsequence of $S_i[3 + 2v(\gamma-1), |S_i|] = \ell_3^{(i)} R^{v-\gamma}$, as $X[\gamma] \neq \ell_3^{(i)}$ and $|X[\gamma, |X|]| = v - \gamma + 1$ contains more literals than the $v - \gamma$ repetitions of R in $S_i[3 + 2v(\gamma-1), |S_i|]$, a contradiction. $\square$

Lemma 3. *Let ϕ be a 3-CNF formula. Then, ϕ is satisfiable if and only if $\mathcal{S}(\phi)$ has an MCS that is not contained in $\mathcal{Z}(\phi)$.*

PROOF. We first show that if ϕ is satisfiable, then $\mathcal{S}(\phi)$ has an MCS that is not contained in $\mathcal{Z}(\phi)$. Let f be a satisfying assignment of ϕ . Let $S = s_1 s_2 \dots s_v$ be the string obtained by $s_j = x_j$ if $f(x_j) = \text{True}$, otherwise, $s_j = \bar{x}_j$. Note that S is a subsequence of S_0 . Since f is a satisfying assignment, each $C_i = \ell_1^{(i)} \vee \ell_2^{(i)} \vee \ell_3^{(i)}$ has a literal $\ell_j^{(i)} \in \{x_\alpha, \bar{x}_\alpha\}$ that is assigned True. Thus, S is a subsequence of S_i for each $1 \leq i \leq m$ by matching s_α with $\ell_j^{(i)}$ and each other literal with one occurrence of R . Since S is not a subsequence of any string $\mathcal{Z}(\phi)$, $\mathcal{S}(\phi)$ has an MCS that is not contained in $\mathcal{Z}(\phi)$.

We next show the opposite direction. Suppose that there is a common subsequence X of $\mathcal{S}(\phi)$ that is not contained in $\mathcal{Z}(\phi)$. Since each string Z_i contains both x_j and $\bar{x}_j$ except for $j = i$, we

have that X contains at least one of x_j or $\bar{x}_j$ for each $1 \leq j \leq v$. Let X' be a subsequence of X such that, for each $1 \leq j \leq v$, X' contains exactly one of x_j or $\bar{x}_j$. From Lemma 2, for each $C_i = \ell_1^{(i)} \vee \ell_2^{(i)} \vee \ell_3^{(i)}$, X' does not contain $\bar{\ell}_1^{(i)} \bar{\ell}_2^{(i)} \bar{\ell}_3^{(i)}$ as a subsequence. From X' , we obtain an assignment f : if X' contains x_j , $f(x_j) = \text{True}$, and otherwise $f(x_j) = \text{False}$. From Lemma 2, f is a satisfying assignment of ϕ . $\square$

Note that ANOTHER MCS is in NP, as we can verify in polynomial time that a given string is not contained in the input set $\mathcal{Z}$ and that it is indeed maximal for $\mathcal{S}$ [23]. Overall, the provided reduction proves the NP-completeness of ANOTHER MCS, and we obtain:

THEOREM 1.1. *ANOTHER MCS is NP-complete, even for instances where $|\mathcal{Z}| = O(n)$, where n is the maximum length of an input string.*

As mentioned in Section 1.2, ANOTHER MCS can be solved output-polynomial time under the assumption that MCS ENUMERATION admits an output-polynomial time algorithm. This together with Theorem 1.1 implies that MCS ENUMERATION has no output-polynomial time algorithm unless $P = NP$. Similarly, MCS INDEXING also has no output-polynomial time algorithm. Finally, the NP-hardness of MCS ASSESSMENT with linear sized z follows as well. Indeed, for any 3-SAT instance ϕ with v literals, we have $|\mathcal{Z}(\phi)| = v$, and by setting $z = v$, polynomial-time algorithms for MCS ASSESSMENT solve ANOTHER MCS in polynomial time as well. Thus, Corollaries 1-3 are proved.

We believe that these hardness results provide strong evidence for the conjecture that MCS ENUMERATION and MCS INDEXING do not have an output-quasi-polynomial time algorithm unless $P = NP$. If MCS ENUMERATION or MCS INDEXING had an output-quasi-polynomial time algorithm, then NP-complete problem could be solved in quasi-polynomial time. This would imply that any problem in NP could be solved in quasi-polynomial time.

4 Hardness of MCS ENUMERATION on a binary alphabet

The reduction described in the previous section only works for strings on unbounded alphabet size. A natural question that arises is whether the problem becomes easier when the alphabet is small. In this section, we show that MCS ENUMERATION, and other related problems, are still challenging even when the input strings are binary. More precisely, we give a one-to-one reduction from MIS ENUMERATION to MCS ENUMERATION for binary strings, similar to the one used in [5, Proposition 1].

Before explaining our reduction, we must first introduce some terms related to hypergraphs. A hypergraph is a pair $\mathcal{H} = (V, \mathcal{E})$, where $\mathcal{E} \subseteq \mathcal{P}(V)$. For a hypergraph $\mathcal{H} = (V, \mathcal{E})$, a set of vertices U is *independent* if for any hyperedge $E \in \mathcal{E}$, $E \not\subseteq U$ holds. An independent set U is a *maximal independent set* of $\mathcal{H}$ if any proper subset $W \supset U$ is not independent. The task of MIS ENUMERATION is to output all maximal independent sets in $\mathcal{H}$. An output-quasi-polynomial time enumeration of MIS ENUMERATION is proposed by Fredman and Khachiyan [18]. No output-polynomial time algorithm for MIS ENUMERATION is known so far, and its existence is considered to be a long-standing open problem in this field. In what follows, we regard V as the set of integers $\{1, \dots, |V|\}$ to simplify the discussion.

Let $\mathcal{H} = (V, \mathcal{E})$ be a hypergraph. To simplify our proof, we assume that $\mathcal{H}$ has at least two vertices. Furthermore, we can assume that $\mathcal{H}$ has no independent sets with cardinality $|V| - 1$. Indeed, let us add a vertex v that is contained in all hyperedges, and let us denote as $\mathcal{H}'$ the resulting hypergraph. It is easy to verify that the collection of maximal independent sets containing v in $\mathcal{H}'$ corresponds to the collection of maximal independent sets in the original hypergraph $\mathcal{H}$. Moreover, every maximal independent set that does not contain v in $\mathcal{H}'$ contains all the vertices in V , meaning that V itself is the only such maximal independent set. If we have an output-polynomial time

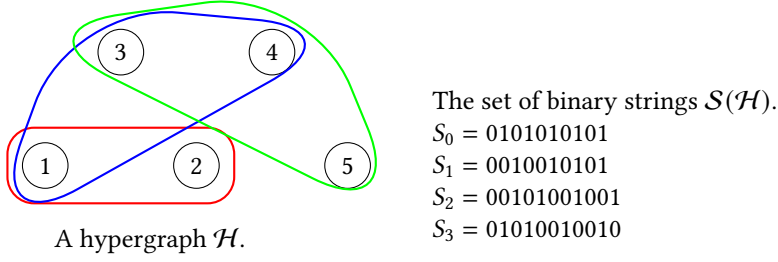


Fig. 1. An example of our reduction. Let $E_1 = \{1, 2\}$, $E_2 = \{1, 3, 4\}$, and $E_3 = \{3, 4, 5\}$. The maximal independent sets in $\mathcal{H}$ are $\{1, 3, 5\}$, $\{1, 4, 5\}$, $\{2, 3, 4\}$, $\{2, 4, 5\}$ and $\{2, 3, 5\}$. The MCSs in S_0 , S_1 , S_2 , and S_3 are 01010101, 01001001, 01001010, 00100101 and 00101001.

algorithm for such restricted hypergraphs, MIS ENUMERATION can be solved in output-polynomial time for general hypergraphs. Therefore, without loss of generality, we can assume that $\mathcal{H}$ has no independent sets of cardinality $|V| - 1$. This assumption is equivalent to assuming that there exists no vertex belonging to every hyperedge.

We now give our reduction from MIS ENUMERATION to MCS ENUMERATION (see Figure 1 for an example). We first define S_0 as $(01)^{|V|}$. Let $E_i = \{u_1, \dots, u_h\}$ be a hyperedge in which for each $1 \leq j \leq |E_i|$, $u_j \in V = \{1, \dots, n\}$. Suppose without loss of generality that $u_j < u_{j+1}$ for $1 \leq j < |E_i|$. For each hyperedge E_i , we define string $S_i = T_1 \dots T_{|V|+|E_i|-1}$, where $T_j = 0$ if $j = u_k + k - 1$ for some $1 \leq k \leq |E_i|$, otherwise $T_j = 01$. In other words, we start to build S_i from $(01)^{|V|-1}$, and for each $u_j \in E_i$, we add a further 0 before the u_j -th occurrence of 01 of the string, or at the end of the string for $u_j = |V|$. As an example, consider $E_3 = \{3, 4, 5\}$ in the hypergraph of Figure 1: to build the corresponding S_3 we start from $(01)^4 = 01010101$, and we add two zeros before the third and fourth one, and a last zero in the last position of the string, yielding $S_3 = 01010010010$. A key observation of this construction is that S_i contains $(01)^{j-1}0(01)^{|V|-j}$ as a subsequence for each j . We denote the set of strings $S_0, \dots, S_{|\mathcal{E}|}$ by $\mathcal{S}(\mathcal{H})$. This reduction can be done in polynomial time.

Next, we show that for a set of binary strings $\mathcal{S}(\mathcal{H})$, any MCS does not contain 11 as a substring.

Observation 2. Any MCS of $\mathcal{S}(\mathcal{H})$ does not contain 11 as a substring.

PROOF. Let X be an MCS such that $X[i, i+1] = 11$ for some i . For any $S \in \mathcal{S}(\mathcal{H})$, there is an arrangement $(\delta_1, \dots, \delta_{|X|})$ such that $S[\delta_j] = X[j]$ for all $1 \leq j \leq |X|$. By construction, in each string of $\mathcal{S}(\mathcal{H})$, there is at least one occurrence of 0 between any two occurrences of 1. Therefore, $S[\delta_i, \delta_{i+1}]$ contains 101 as a subsequence, which contradicts the maximality. $\square$

Note that any MCS must also be a subsequence of $S_0 = (01)^{|V|}$, so it can be obtained by deleting some characters from S_0 . Specifically, by Observation 2, when we delete a 0 we must also delete one of its adjacent 1s; otherwise such subsequence would not be maximal, as it would contain 11 as a substring.

Let $\mathcal{X}$ be the set of MCSs in $\mathcal{S}(\mathcal{H})$. The goal of our proof is to show that there is a bijection between the collection of maximal independent sets in $\mathcal{H}$ and $\mathcal{X} \setminus \{(01)^{|V|-1}\}$. We first show that $(01)^{|V|-1}$ is an MCS of $\mathcal{S}(\mathcal{H})$ under our assumption.

Lemma 4. Suppose that $\mathcal{H}$ has at least two vertices and has no independent set of size $|V| - 1$, that is, no vertex belonging to every hyperedge. Then, the string $(01)^{|V|-1}$ is an MCS of $\mathcal{S}(\mathcal{H})$.

PROOF. Since for each $1 \leq i \leq |\mathcal{E}|$, S_i is obtained from $(01)^{|V|+|E_i|-1}$ by replacing $|E_i|$ occurrences of 01 with one 0 each, it has $(01)^{|V|-1}$ as a subsequence. By contradiction, let X be a common subsequence of $\mathcal{S}(\mathcal{H})$ that contains $(01)^{|V|-1}$ as a proper subsequence, so that $(01)^{|V|-1}$ is not maximal. Since X is a subsequence of S_0 , it can be represented by $(01)^\alpha W (01)^\beta$ for some W , where $\alpha + \beta = |V| - 1$ and W is either 0 or 01 from Observation 2. However, since each S_i contains exactly $|V| - 1$ ones, it follows that $W = 0$ and $X = (01)^\alpha 0 (01)^\beta$.

By hypothesis $\mathcal{H}$ has no independent set with cardinality $|V| - 1$, so there is a hyperedge E_i contained in $V \setminus \{\alpha + 1\}$. In other words, E_i does not contain vertex $\alpha + 1$.

We consider a greedy arrangement of X for S_i . In this arrangement, the α -th 1 in X is arranged to the α -th 1 in S_i . This is possible by Observation 2. Since E_i does not contain $\alpha + 1$, the two characters after the α -th 1 in S_i are 01. However, three characters after the α -th 1 in X is 001. To arrange 001, we need 0101 or 01001 in S_i , hence we consume one more 1. Therefore, after arranging 001 to S_i , the remaining occurrences of 1s in S_i are $|V| - \alpha - 3$. However, the remainder of X contains $|V| - \alpha - 2$ occurrences of 1s, so the greedy arrangement is not possible and X cannot be a subsequence of S_i , a contradiction. $\square$

In what follows, we show a bijection between the collection of maximal independent sets in $\mathcal{H}$ and the set of MCSs without $(01)^{|V|-1}$. Let $E_i = \{u_1, \dots, u_h\}$ be a hyperedge in $\mathcal{H}$. This hyperedge E_i forbids any independent set in $\mathcal{H}$ from containing all vertices in E_i . In our reduction, the role of S_i is to forbid a common subsequence in $\mathcal{S}(\mathcal{H})$ containing $R(E_i) = R_1 R_2 \dots R_{|V|}$ as a subsequence, where $R_j = 01$ if $j \in E_i$, and $R_j = 0$ otherwise.

Lemma 5. *Let $E_i = \{u_1, \dots, u_h\}$ be a hyperedge in $\mathcal{H}$. Then, S_i does not contain $R(E_i) = R_1 R_2 \dots R_n$ as a subsequence.*

PROOF. We prove the lemma by induction on the cardinality of E . If E_i consists of only one vertex u_1 , $R(E_i) = 0^{u_1-1} 010^{|V|-u_1}$ and $S_i = (01)^{u_1-1} 0 (01)^{|V|-u_1}$. In this case, $0^{u_1-1} 0$ is a subsequence of $(01)^{u_1-1} 0$. Moreover, $0^{u_1-1} 01$ is not a subsequence of $(01)^{u_1-1} 0$. From observation 1, $R(E_i)$ is a subsequence of S_i if and only if $T' = 10^{|V|-u_1}$ is a subsequence of $S' = (01)^{|V|-u_1}$. Thus, $R(E_i)$ is not a subsequence of S_i .

As an induction step, assume that E_i contains $h > 1$ vertices with $u_1 < u_2 < \dots < u_h$. Let $R = R(E_i)$. Since $R[1, u_1] = 0^{u_1-1} 0$, $R[1, u_1]$ is a subsequence of $S_i[1, 2u_1 - 1] = (01)^{u_1-1} 0$. However, $R[1, u_1 + 1] = 0^{u_1-1} 01$ is not a subsequence of $S_i[1, 2u_1 - 1]$. Therefore, from Observation 1, R is a subsequence of S_i if and only if $R[u_1 + 1, |R|]$ is a subsequence of $S_i[2u_1, |S_i|]$. For a set of vertices $E' = \{u_2 - u_1, u_3 - u_1, \dots, u_h - u_1\}$, we obtain the same strings $R[u_1 + 1, |R|]$ and $S_i[2u_1, |S_i|]$. Since $|E'| < h$, $S_i[2u_1, |S_i|]$ does not contain $R[u_1 + 1, |R|]$ as a subsequence. $\square$

From Observation 2 and Lemma 5, any MCS different from $(01)^{|V|-1}$ contains $|V|$ occurrences of 0 and does not contain $R(E)$ as a subsequence for each $E \in \mathcal{E}$. We are ready to prove that there is a bijection between the collection of maximal independent sets in $\mathcal{H}$ and the set of MCSs of $\mathcal{S}(\mathcal{H})$ different from $(01)^{|V|-1}$.

Lemma 6. *Let X be the set of MCSs of $\mathcal{S}(\mathcal{H})$. Then, there is a bijection ψ from the collection of maximal independent sets in $\mathcal{H}$ to $X \setminus \{(01)^{|V|-1}\}$.*

PROOF. We define a function ψ on a larger domain, the collection of all independent sets in $\mathcal{H}$, and then prove that its restriction of ψ on the collection of maximal independent sets in $\mathcal{H}$ is an intended bijection. For an independent set U , $\psi(U)$ is defined as $X_1 \dots X_{|V|}$, where $X_i = 01$ if $i \in U$, otherwise, $X_i = 0$. We show that $\psi(U)$ is a common subsequence of $\mathcal{S}(\mathcal{H})$. From the definition of $\psi(U)$, $\psi(U)$ is a subsequence of S_0 . Thus, we show that $\psi(U)$ is contained in S_i as a subsequence for $1 \leq i \leq |\mathcal{E}|$.

Since U is an independent set in $\mathcal{H}$, we have $E_i \setminus U \neq \emptyset$. Let h be the minimum integer in $E_i \setminus U$. From the construction of S_i , S_i contains $Y = (01)^{h-1}0(01)^{|V|-h}$. We decompose Y into $Y_1 \dots, Y_{|V|}$, where $Y_j = 01$ for each j with $j \neq h$ and $Y_h = 0$. For each $1 \leq j \leq |V|$, X_j is a subsequence of Y_j . Therefore, $\psi(U)$ is a subsequence of S_i . We next show that if U is maximal, $\psi(U)$ is also maximal. Suppose by contradiction that $\psi(U)$ is non-maximal. This means that there is a string T that is a supersequence of $\psi(U)$ and a subsequence of S_0 . Since $\psi(U)$ contains $|V|$ occurrences of 0, T must be obtained by adding some 1s to $\psi(U)$. Without loss of generality, let T be the string that is obtained by adding only one 1 to $\psi(U)$, between X_j and X_{j+1} , say, for a given j such that $X_j = 0$. By construction, $\psi(U \cup \{j\}) = T$. By maximality of U , there exists a hyperedge $E \subseteq U \cup \{j\}$. Then, T contains $R(E)$ as a subsequence. From Lemma 5, T is not a common subsequence of $\mathcal{S}(\mathcal{H})$.

Conversely, let X be an MCS in $\mathcal{S}(\mathcal{H})$ different from $(01)^{|V|-1}$. By Observation 2, X does not contain 11 as a substring, so it has to contain $|V|$ occurrences of 0, otherwise it would be a subsequence of $(01)^{|V|-1}$. Therefore, X can be decomposed by $X_1 \dots X_{|V|}$, where each X_i is either 0 or 01. We define a set of vertices $\psi^{-1}(X)$ as follows: $X_i = 01$ if and only if $\psi^{-1}(X)$ contains i . Let $E_i = \{u_1, \dots, u_h\}$ be a hyperedge in $\mathcal{E}$. By Lemma 5 and by transitivity of the subsequence relation, X does not contain $R(E_i)$ as a subsequence. Therefore, $\psi^{-1}(X)$ does not contain E_i and $\psi^{-1}(X)$ is an independent set of $\mathcal{H}$. Finally, suppose by contradiction $\psi^{-1}(X)$ is not a maximal independent set. Then, by construction, the string $\psi(\psi^{-1}(X))$ is a supersequence of X and $\psi(\psi^{-1}(X))$ a common subsequence of $\mathcal{S}(\mathcal{H})$. But this contradicts the maximality of X . $\square$

The reduction presented in this section, together with the previous lemma, concludes the proof of Theorem 1.2:

THEOREM 1.2. *Given a hypergraph $\mathcal{H} = (V, \mathcal{E})$, there exists a set of binary strings $\mathcal{S}(\mathcal{H})$, computable in time polynomial in $\|\mathcal{H}\|$, and a bijection between maximal independent sets in $\mathcal{H}$ and the set $\text{MCS}(\mathcal{S}(\mathcal{H})) \setminus \{w\}$, where the string w consists of $|V| - 1$ repetition of the string “01”.*

This bijection implies that if there is an output-polynomial time algorithm for MCS ENUMERATION for binary strings, then an output-polynomial time algorithm for MIS ENUMERATION and the following corollary holds.

Corollary 4. *If MCS ENUMERATION for binary strings can be solved in output-polynomial time, MIS ENUMERATION can be solved in output-polynomial time.*

This result implies that output-polynomial time enumeration of MCS ENUMERATION for binary strings is challenging, since output-polynomial time enumeration of MIS ENUMERATION is a long-standing open problem. Moreover, since we give a bijection between the collection of maximal independent sets and the set of MCSs in binary strings, counting MCSs is at least as hard as counting the maximal independent sets in hypergraphs: this latter problem is in fact #P-complete not just for hypergraphs, but even in the special case of simple graphs, which can be modeled as hypergraphs where all hyperedges have size 2 [35]. Thus, we obtain the following result as well:

Corollary 6. *MCS COUNTING is #P-complete, even on binary strings.*

This also immediately implies that there is no assessment algorithm running in time polynomial in the input size, unless $P = NP$: using such an assessment algorithm in a binary search manner on z , we could obtain the exact number of MCSs with at most $|V|$ calls to the algorithm, as the number of maximal independent sets is at most $2^{|V|}$. On the other hand, an efficient assessment algorithm, in the sense of running time polynomial in the input and in z , is an open problem, both for general z and for small (polynomial) values of z .

5 Positive results for parameterized cases

While we have shown MCS ENUMERATION to be hard in general, it is worth investigating what parameters can make the problem tractable. In this section, we present some positive results concerning the parameterized complexity of MCS ENUMERATION with respect to the string length n , and the number of input strings k .

To look at a related problem, we can observe how the longest common subsequence between k strings of length n cannot be found in $O(n^{k-\epsilon})$, for any $\epsilon > 0$, unless the Strong Exponential Time Hypothesis (SETH) is false [1], but this bound becomes tractable for constant values of k . Do MCS present a similar situation?

We note that all the positive results presented here are also valid for the ANOTHER MCS, because of its relationship with MCS ENUMERATION described in Section 1.2. Furthermore, since by enumerating we are also implicitly counting, we also derive the positive results shown in Table 1 for the MCS COUNTING and MCS ASSESSMENT problems.

In the following, we will consider how the problem complexity behaves if we consider parts of the input size as parameters. To correctly classify the complexity, we introduce two complexity classes for parametrized problems. We say that a problem is FPT with respect to a parameter p if it can be solved in $O(f(p)|\mathcal{I}|^c)$ time for some constant c and computable functions f [16], where $|\mathcal{I}|$ is the non-parametrized size of the input. Moreover, we say that a parameterized problem is in XP with respect to a parameter p if it can be solved in $O(f(p)|\mathcal{I}|^{g(p)})$ time for some computable functions f, g [16]. Clearly, all problems that are FPT are also in XP. For an enumeration algorithm, we say it is XP-delay for parameter p if its delay is bounded by $O(f(p)|\mathcal{I}|^{g(p)})$ for some computable functions f, g .

Before presenting the results, we remark that we can check whether a given string is an MCS of a set $\mathcal{S}$ of strings in $O(k\|\mathcal{S}\| \log(\|\mathcal{S}\|))$ time by using the algorithm of [23]: indeed, what they denote as “total length of the strings” corresponds to what we denote $\|\mathcal{S}\|$, and their m is our k .

5.1 MCS ENUMERATION parameterized by minimum string length

Consider a set of k strings $\mathcal{S} = \{S_1, \dots, S_k\}$, and let l be the *minimum* of their lengths (clearly, $l \leq n$). Without loss of generality, assume l is the length of S_1 . We show here how MCS enumeration can be efficiently parameterized in l .

Firstly, observe that any common subsequence X of $\mathcal{S}$ is a subsequence of the shortest string S_1 . Consequently, we can encode each common subsequence of $\mathcal{S}$ (even non-maximal ones) as a binary string of length l , through one of its occurrences in S_1 : it suffices to indicate for each position of S_1 whether we take, or discard, the corresponding character.

The number of such encodings, and consequently the number of CS and MCS of $\mathcal{S}$, is then surely bounded by $2^{|S_1|} = 2^l$. Since we can check if a given string is an MCS of $\mathcal{S}$ in $O(k\|\mathcal{S}\| \log(\|\mathcal{S}\|))$ time, we can naively enumerate all MCSs by iterating over all possible subsequences of S_1 , and discarding those that do not pass the maximality check, in $O(k\|\mathcal{S}\| \log(\|\mathcal{S}\|) \cdot 2^l)$ time, which is significant for small values of l .

Indeed when $l \leq c \log(\|\mathcal{S}\|)$, for some constant c , the bound reduces to a polynomial-time algorithm, running in $O(k\|\mathcal{S}\|^{1+c} \log(\|\mathcal{S}\|))$ time. Analogously, when $l \leq c \log^d(\|\mathcal{S}\|)$ for some constants c and d , we obtain a quasi-polynomial-time algorithm, running in $O(k\|\mathcal{S}\| \log(\|\mathcal{S}\|) 2^{c \log^d(\|\mathcal{S}\|)})$ time. We can thus derive the following.

THEOREM 1.3. *MCS ENUMERATION can be solved in FPT total time with respect to l . Specifically, when l is logarithmic in $\|\mathcal{S}\|$, all MCSs can be enumerated in polynomial time in n and k , and when l is polylogarithmic in $\|\mathcal{S}\|$, enumeration takes quasi-polynomial time in n and k .*

Finally, we note that the value of parameter l can sometimes be reduced, obtaining more tractable instances, due to the following observation:

Observation 3. *A character occurring in a common subsequence occurs in every input string. Therefore, a character not occurring in every input string can be deleted from the input without altering the set of MCSs.*

It follows that we can preprocess any instance $\mathcal{S} = \{S_1, \dots, S_k\}$ by removing every occurrence of such characters, possibly reducing the minimum string length l . We can also observe that, in this preprocessed instance, $|\Sigma| \leq l$ because any remaining character must occur in the shortest string.

5.2 MCS ENUMERATION parameterized by the number of strings

Consider now the other potential parameter: the number k of strings in the input instance $\mathcal{S} = \{S_1, \dots, S_k\}$, which we here assume to be constant. We show that the enumeration algorithm in [12] can be generalized to handle any number k of strings of maximum length n in $O(kn^{k+1}(n^k + kn \log(kn)))$ delay. In this section, we give a brief sketch of the generalization.

Recall that we can check for maximality of a given subsequence in $O(k\|\mathcal{S}\| \log(\|\mathcal{S}\|))$ time. The algorithm in [12] uses two main concepts that need to be generalized to k strings: the set of unshiftables $\mathcal{U}$, given by pairs of matching positions in the two strings that satisfy further specific properties, and, for every valid prefix P of an MCS, the set of valid candidates for extension $Ext_P \subseteq \mathcal{U}$.

The former can be easily generalized to k strings, where each unshiftable becomes a k -tuple of matching positions with some constraints, hence $|\mathcal{U}| \leq n^k$. In essence, a k -tuple $u = (u_1, \dots, u_k)$, corresponding to character $c(u) := S_i[u_i]$, is unshiftable if there is another unshiftable $v = (v_1, \dots, v_k)$ (or, as a base case, $v = (|S_1| + 1, \dots, |S_k| + 1)$) such that, for each i , u_i is the rightmost occurrence of $c(u)$ in $S[1, v_i - 1]$ (i.e. before v_i in S_i). We can trivially extend the characterization of [12, Fact 1] to give a constructive recursive definition of $\mathcal{U}$, running in $O(k|\Sigma|n^k)$.

Secondly, consider Ext_P : for a given string P that is known to be a prefix of an MCS, the set Ext_P is a set of unshiftables corresponding to candidate extensions of P . Being able to compute Ext_P (and refine it with maximality checks), immediately gives us a backtracking enumeration algorithm for MCSs, as we can find every c such that $P \cdot c$ is still a prefix of an MCS. Again, we can extend its definition of Ext_P from [12] to the case of k strings: [12, Definitions 7-8] define a set of unshiftable 2-tuples immediately following P that are “not dominated” (i.e., no other unshiftable occurs completely in between them and P), and this can be trivially considered with k -tuples on k strings instead of 2. Even if some of the more refined optimizations in [12] do not immediately extend to the case of k strings, the Ext_P set can still be naively computed in $O(kn^{2k})$ time by pairwise checking all the elements in $\mathcal{U}$, and eliminating dominated ones.

With these definitions, the characterization given by [12, Theorem 3] generalizes as well, yielding the following binary partition enumeration algorithm (see Algorithm 1): first, we compute the set $\mathcal{U}$ in $O(k|\Sigma|^2 n^k)$ time. Then, in each binary partition recursive call, associated with a given valid prefix P , we (1) check if P is an MCS, and in that case output P ; otherwise (2) we compute the Ext_P set in $O(kn^{2k})$ time, and then check for each of the $O(n^k)$ elements $u = (u_1, \dots, u_k) \in Ext_P$ whether P is an MCS of $\{S_1[1, u_1 - 1], \dots, S_k[1, u_k - 1]\}$. If this is the case, we recurse on prefix $P \cdot c(u)$. Therefore, since the binary partition tree depth is $O(n)$, the delay of such an algorithm is bounded by $O(n(k\|\mathcal{S}\| \log(\|\mathcal{S}\|) + kn^{2k} + kn^k\|\mathcal{S}\| \log(\|\mathcal{S}\|))) = O(kn^{2k+1} + kn^{k+1}\|\mathcal{S}\| \log(\|\mathcal{S}\|))$ time, and since $k \geq 2$ the factor $\|\mathcal{S}\| \log(\|\mathcal{S}\|) \leq kn \log(kn)$ is subsumed, as well as the $O(k|\Sigma|^2 n^k)$ preprocessing time. We can thus conclude:

Algorithm 1: (XP-Delay Enumeration Algorithm for MCS)**Input:** $S = \{S_1, \dots, S_k\}$ **Output:** $MCS(S)$

```

1 Algorithm EnumerateMCS( $S$ )
2   Let  $\#, \$ \notin \Sigma$ ; add them resp. at positions 0 and  $|S_i| + 1$  of each  $S_i$ 
3    $\mathcal{U} = \text{FindUnshiftables}((|S_1| + 1, \dots, |S_k| + 1))$ 
4    $\text{BinaryPartition}(\#, (0, \dots, 0))$ 

Input: a valid prefix  $P$ , the indices of the shortest prefixes of  $S_1, \dots, S_k$  containing  $P$ 
Output: all strings in  $MCS(S)$  having  $P$  as prefix
5 Procedure BinaryPartition( $P, (\ell_1, \dots, \ell_k)$ )
6   Let  $\mathcal{U}_P = \{(u_1, \dots, u_k) \in \mathcal{U} \mid u_i > \ell_i \forall i\}$ 
7   Compute  $\text{Ext}_P$  by eliminating dominated elements from  $\mathcal{U}_P$ 
8   if  $\text{Ext}_P = \emptyset$  then
9     | output  $P$ 
10  else
11    | for each  $c \in \Sigma$  corresponding to some  $(u_1, \dots, u_k) \in \text{Ext}_P$  do
12      | | if  $P \in MCS(S_1[0, u_1], \dots, S_k[0, u_k])$  then
13        | | | For each  $j$ , let  $i_j$  be the first occurrence of  $c$  after  $\ell_j$  in  $S_j$ 
14        | | |  $\text{BinaryPartition}(Pc, (i_1, \dots, i_k))$ 

Input: An unshiftable  $(i_1, \dots, i_k)$  of strings  $S_1, \dots, S_k$ 
Output: The set of unshiftable  $k$ -tuples of strings  $S_1[1, i_1 - 1], \dots, S_k[1, i_k - 1]$ 
15 Function FindUnshiftables( $((i_1, \dots, i_k))$ )
16   Let  $\mathcal{U} = \emptyset$ 
17   for  $c \in \Sigma$  do
18     | For each  $j$ , let  $r_j$  be the rightmost occurrence of  $c$  in  $S_j[1, i_j - 1]$ 
19     | if  $r_j \neq -1$  for all  $j$ , and  $(r_1, \dots, r_k) \notin \mathcal{U}$  then
20       | | Add  $(r_1, \dots, r_k)$  to  $\mathcal{U}$ 
21       | |  $\mathcal{U} = \mathcal{U} \cup \text{FindUnshiftables}((r_1, \dots, r_k))$ 
22   return  $\mathcal{U}$ 

```

THEOREM 1.4. *MCS ENUMERATION for $k \geq 2$ strings of maximum length n can be solved in XP-delay with respect to k . Specifically, MCSs can be enumerated in $O(kn^{2k+1})$ time delay, after $O(k|\Sigma|n^k)$ time preprocessing.*

It is worth remarking that the space requirement is $O(n^k)$ due to storing the unshiftables, and that the above complexity can be given in the standard RAM model because of our assumption on k : the natural choice of word size is $O(k \log n)$ bits to index the unshiftables data structure, and the word size should be logarithmic in the input size, which is $\|S\| = O(nk)$. We observe how, since k is a constant, this requirement is met. However, further efforts to develop an enumeration algorithm for non-constant values of k should take this issue into account.

6 Conclusion

In this paper we studied the complexity of enumerating Maximal Common Subsequences (MCS), a special case of Maximal Frequent Subsequence Mining, and some related problems such as counting

and assessing the number of MCSs. We showed that, in general, MCS ENUMERATION is intractable, as no algorithm can enumerate all MCSs in polynomial time with respect to the combined size of the input and the output, unless $P = NP$. This result is based on a reduction from 3-SAT to a decision problem that, given a set of input strings and a set $\mathcal{Z}$ of MCSs, asks whether there is another MCS not included in $\mathcal{Z}$.

We also show that no output-polynomial algorithms exist for constructing polynomial-sized indexes that store all MCSs and allow to efficiently query any specific solution. Furthermore, we prove that MCS assessment, where we are asked to decide whether a given instance has at least z solutions, is NP-hard even when z is linear in the input size.

In light of these negative results, we explore potential parametrizations for the studied problems. In the case of binary strings, we construct a one-to-one reduction between the set of Maximal Independent Sets (MIS) in hypergraphs and the set of MCSs of a particular instance. This allows us to conclude that counting MCSs is #P-Complete, and that MCS ENUMERATION is as hard as MIS ENUMERATION on hypergraphs, for which the existence of an efficient algorithm is a long-standing open problem. On the positive side, for bounded string lengths we identify an FPT algorithm for solving all the problems. When the number of strings is instead bounded, we provide an XP-delay algorithm for MCS ENUMERATION.

However, it remains open whether efficient algorithms exist for MCS COUNTING or MCS INDEXING when the number of strings is treated as a parameter, as well as whether we can perform efficient assessment (for z polynomial in the input size) in the case of binary strings.

Acknowledgments

We wish to thank Roberto Grossi and Rossano Venturini for useful conversations. This work is partially supported by JSPS KAKENHI Grant Numbers JP21K17812, JP22H03549, JP23K28034, JP24H00686, JP24H00697, and JP25K21273, and JST ACT-X Grant Number JPMJAX2105. Author GP is supported by the Italian Ministry of Research, under the complementary 36 actions to the NRRP “Fit4MedRob - Fit for Medical Robotics” Grant (#PNC0000007). Authors GB and AC are partially supported by MUR PRIN 2022 project EXPAND: scalable algorithms for EXploratory Analyses of heterogeneous and dynamic Networked Data (#2022TS4Y3N).

References

- [1] A. Abboud, A. Backurs, and V. V. Williams. Tight hardness results for LCS and other sequence similarity measures. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 59–78, 2015.
- [2] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22(2):207–216, June 1993.
- [3] M. A. Babin and S. O. Kuznetsov. Dualization in lattices given by ordered sets of irreducibles. *Theoretical Computer Science*, 658:316–326, 2017. Horn formulas, directed hypergraphs, lattices and closure systems: related formalism and application.
- [4] A. Bechini, A. Bondielli, P. Dell’Oglio, and F. Marcelloni. From basic approaches to novel challenges and applications in sequential pattern mining. *Applied Computing and Intelligence*, 3(1):44–78, 2023.
- [5] G. Blin, L. Bulteau, M. Jiang, P. J. Tejada, and S. Vialette. Hardness of longest common subsequence for sequences with bounded run-lengths. In J. Kärkkäinen and J. Stoye, editors, *Combinatorial Pattern Matching*, pages 138–148, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
- [6] F. Bökler. *Output-sensitive complexity of multiobjective combinatorial optimization with an application to the multiobjective shortest path problem*. PhD thesis, Dortmund University, Germany, 2018.
- [7] E. Boros, V. Gurvich, L. Khachiyan, and K. Makino. On maximal frequent and minimal infrequent sets in binary matrices. *Ann. Math. Artif. Intell.*, 39(3):211–221, 2003.
- [8] E. Boros and K. Makino. Generating minimal redundant and maximal irredundant subhypergraphs. *Discret. Appl. Math.*, 358:217–229, 2024.
- [9] C. Brosse, O. Defrain, K. Kurita, V. Limouzy, T. Uno, and K. Wasa. On the hardness of inclusion-wise minimal separators enumeration. *Inf. Process. Lett.*, 185:106469, 2024.

- [10] G. Buzzega, A. Conte, R. Grossi, and G. Punzi. Mcdag: Indexing maximal common subsequences in practice. In S. P. Pissis and W. Sung, editors, *24th International Workshop on Algorithms in Bioinformatics, WABI 2024, September 2-4, 2024, Royal Holloway, London, United Kingdom*, volume 312 of *LIPICs*, pages 21:1–21:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.
- [11] N. Carmeli, S. Zeevi, C. Berkholz, A. Conte, B. Kimelfeld, and N. Schweikardt. Answering (unions of) conjunctive queries using random access and random-order enumeration. *ACM Transactions on Database Systems (TODS)*, 47(3):1–49, 2022.
- [12] A. Conte, R. Grossi, G. Punzi, and T. Uno. Enumeration of maximal common subsequences between two strings. *Algorithmica*, 84(3):757–783, 2022.
- [13] A. Conte, R. Grossi, G. Punzi, and T. Uno. A Compact DAG for Storing and Searching Maximal Common Subsequences. In S. Iwata and N. Kakimura, editors, *34th International Symposium on Algorithms and Computation (ISAAC 2023)*, volume 283 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:15, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- [14] A. Darwiche and P. Marquis. A knowledge compilation map. *J. Artif. Intell. Res.*, 17:229–264, 2002.
- [15] S. Denzumi. *Studies on Decision Diagrams for Efficient Manipulation of Sets and Strings*. PhD thesis, Hokkaido University, Division of Computer Science, Graduate School of Information Science and Technology, mar 2015.
- [16] R. G. Downey, M. R. Fellows, and U. Stege. Parameterized complexity: A framework for systematically confronting computational intractability. *Contemporary Trends in Discrete Mathematics*, 49:49–99, 1997.
- [17] T. Eiter, K. Makino, and G. Gottlob. Computational aspects of monotone dualization: A brief survey. *Discrete Applied Mathematics*, 156(11):2035–2049, 2008.
- [18] M. L. Fredman and L. Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21(3):618–628, 1996.
- [19] F. Fumarola, P. F. Lanotte, M. Ceci, and D. Malerba. Clofast: closed sequential pattern mining using sparse and vertical id-lists. *Knowledge and Information Systems*, 48(2):429–463, 2016.
- [20] D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, and R. S. Sharm. Discovering all most specific sentences. *ACM Trans. Database Syst.*, 28(2):140–174, 2003.
- [21] K. Hayase, K. Sadakane, and S. Tani. Output-size sensitiveness of OBDD construction through maximal independent set problem. In D.-Z. Du and M. Li, editors, *Computing and Combinatorics*, pages 229–234, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.
- [22] M. Hirota and Y. Sakai. Efficient algorithms for enumerating maximal common subsequences of two strings. *CoRR*, abs/2307.10552, 2023.
- [23] M. Hirota and Y. Sakai. A fast algorithm for finding a maximal common subsequence of multiple strings. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 106(9):1191–1194, 2023.
- [24] T. Horváth, B. Bringmann, and L. D. Raedt. Frequent hypergraph mining. In S. H. Muggleton, R. P. Otero, and A. Tamaddoni-Nezhad, editors, *Inductive Logic Programming, 16th International Conference, ILP 2006, Santiago de Compostela, Spain, August 24-27, 2006, Revised Selected Papers*, volume 4455 of *Lecture Notes in Computer Science*, pages 244–259. Springer, 2006.
- [25] T. Horváth, K. Otaki, and J. Ramon. Efficient frequent connected induced subgraph mining in graphs of bounded tree-width. In H. Blockeel, K. Kersting, S. Nijssen, and F. Zelezny, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part I*, volume 8188 of *Lecture Notes in Computer Science*, pages 622–637. Springer, 2013.
- [26] D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3):119–123, 1988.
- [27] L. Khachiyan, E. Boros, K. M. Elbassioni, and V. Gurvich. On enumerating minimal dicuts and strongly connected subgraphs. *Algorithmica*, 50(1):159–172, 2008.
- [28] B. Kimelfeld and P. G. Kolaitis. The complexity of mining maximal frequent subgraphs. *ACM Trans. Database Syst.*, 39(4), Dec. 2015.
- [29] S. Minato. *Data Mining Using Binary Decision Diagrams*, pages 97–110. Springer International Publishing, Cham, 2010.
- [30] S. Minato, T. Uno, and H. Arimura. Lcm over zbdds: Fast generation of very large-scale frequent itemsets using a compact graph-based representation. In T. Washio, E. Suzuki, K. M. Ting, and A. Inokuchi, editors, *Advances in Knowledge Discovery and Data Mining*, pages 234–246. Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
- [31] G. Punzi, A. Conte, R. Grossi, and A. Marino. *An Efficient Algorithm for Assessing the Number of st-Paths in Large Graphs*, pages 289–297. Society for Industrial and Applied Mathematics, 2023.
- [32] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In P. Apers, M. Bouzeghoub, and G. Gardarin, editors, *Advances in Database Technology – EDBT ’96*, pages 1–17, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.
- [33] Y. Udagawa. Maximal frequent sequence mining for finding software clones. In *Proceedings of the 18th International Conference on Information Integration and Web-Based Applications and Services, iiWAS ’16*, page 26–33, New York, NY,

- USA, 2016. Association for Computing Machinery.
- [34] T. Uno, M. Kiyomi, and H. Arimura. LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In R. J. B. Jr., B. Goethals, and M. J. Zaki, editors, *FIMI '04, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Brighton, UK, November 1, 2004*, volume 126 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2004.
 - [35] S. P. Vadhan. The complexity of counting in sparse, regular, and planar graphs. *SIAM Journal on Computing*, 31(2):398–427, 2001.
 - [36] L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.
 - [37] H. Wang. All common subsequences. In M. M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 635–640, 2007.
 - [38] Y. Wu, C. Zhu, Y. Li, L. Guo, and X. Wu. Netncsp: Nonoverlapping closed sequential pattern mining. *Knowledge-Based Systems*, 196:105812, 2020.
 - [39] X. Yan and J. Han. gSpan: graph-based substructure pattern mining. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 721–724, 2002.
 - [40] G. Yang. The complexity of mining maximal frequent itemsets and maximal frequent patterns. In W. Kim, R. Kohavi, J. Gehrke, and W. DuMouchel, editors, *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004*, pages 344–353. ACM, 2004.
 - [41] M. Yannakakis. Algorithms for acyclic database schemes. In *Proceedings of the Seventh International Conference on Very Large Data Bases - Volume 7, VLDB '81*, page 82–94. VLDB Endowment, 1981.

Received December 2024; revised February 2025; accepted March 2025