Finding Diverse Strings and Longest Common Subsequences in a Graph

Yuto Shida
Hokkaido University, Japan

Giulia Punzi
National Institute of Informatics, Japan

Yasuaki Kobayashi
Hokkaido University, Japan

Takeaki Uno
National Institute of Informatics, Japan

Hiroki Arimura
Hokkaido University, Japan

Abstract

In this paper, we study for the first time the *Diverse Longest Common Subsequences* (LCSs) problem under Hamming distance. Given a set of a constant number of input strings, the problem asks to decide if there exists some subset of K longest common subsequences whose *diversity* is no less than a specified threshold Δ. We analyze the computational complexity of this problem with *Max-Min* and *Max-Sum diversity measures* under Hamming distance, considering both approximation algorithms and parameterized complexity. Our results are summarized as follows. When K is bounded, both problems are polynomial time solvable. In contrast, when K is unbounded, both problems become NP-hard, while Max-Sum Diverse LCSs problem admits a PTAS. Furthermore, we analyze the parameterized complexity of both problems with various combinations of parameters K, r, and Δ, where r is the length of the candidate strings to be selected. Importantly, all *positive results* above are proven in more general setting, where an input is an edge-labeled directed acyclic graph (DAG) that succinctly represents a set of strings of the same length. *Negative results* are proven in the setting where an input is explicitly given as a set of strings. The latter results are equipped with an encoding such a set as the longest common subsequences of a specific input string set.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Sequence analysis, longest common subsequence, Hamming distance, dispersion, approximation algorithms, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

The problem of finding a longest common subsequence of a set of m strings (the LCS problem) is a fundamental problem in computer science, extensively studied in theory and applications for over fifty years [8, 27, 29, 33, 35]. In application areas such as computational biology, pattern recognition, and data compression, longest common subsequences are used for consensus pattern discovery and multiple sequence alignment [22, 35]. It is also common to use the length of longest common subsequence as a similarity measure between two strings.

Table 1 Longest common subsequences of two input strings X and Y over $\Sigma = \{A, B, C, D, E\}$

$\epsilon, A, B, C, D, E,$
$AA, AB, AC, AD, AE, BA, \ldots, CD, CE, DD, EE,$
$ABA, ABB, ABC, ABD, \ldots, CEE,$
$ABAD, ABAE, ABBD, \ldots, BCEE,$
$\underline{ABADD}, \underline{ABAEE}, \underline{ABBDD},$
$\underline{ABBEE}, \underline{ABCDD}, \underline{ABCEE}$

The LCS problem can be solved in polynomial time for constant $m \geqslant 2$ using dynamic programming by Irving and Fraser [29] requiring $O(n^m)$ time, where n is the maximum length of m strings. When m is unrestricted, LCS is NP-complete [33]. From the view of parameterized complexity, Bodlaender, Downey, Fellows, and Wareham [8] showed that the problem is W[t]-hard parameterized with m for all t, is W[2]-hard parameterized with the length ℓ of a longest common subsequence, and is W[1]-complete parameterized with ℓ and m. Bulteau, Jones, Niedermeier, and Tantau [9] presented a *fixed-paraemter tractable* (FPT) algorithm with different parameterization.

Recent years have seen increasing interest in efficient methods for finding a *diverse set of solutions* [5, 19, 24]. In this paper, we consider the problem of finding a diverse set of solutions for *longest common subsequences* of a set $\mathcal{S}$ of input strings under Hamming distance. As an example, Table 1 shows six longest common subsequences (underlined) of the input strings $X = ABABCDDEE$ and $Y = ABCBAEEDD$. The task is to select, say, $K = 3$ longest common subsequences, maximizing the minimum pairwise Hamming distance among them. In general, a set of m strings of length n may have exponentially many longest common subsequences in n. Hence, efficiently finding such a diverse subset of solutions for longest common subsequences is challenging.

Formally, let $d_H(X, Y)$ denote the Hamming distance between two strings $X, Y \in \Sigma^r$ of the equal length $r \geqslant 0$, called r-strings. Throughout this paper, we consider two diversity measures for a multiset $\mathcal{X} = \{X_1, \ldots, X_K\} \subseteq \Sigma^r$ of solutions, allowing repeated elements:

$$D^{\mathrm{sum}}_{d_H}(\mathcal{X}) := \textstyle\sum_{i<j} d_H(X_i, X_j), \qquad \text{(Max-Sum diversity)}, \tag{1}$$

$$D^{\mathrm{min}}_{d_H}(\mathcal{X}) := \min_{i<j} d_H(X_i, X_j), \qquad \text{(Max-Min diversity)}. \tag{2}$$

For $\tau \in \{\mathrm{sum}, \min\}$, $D^{\tau}_{d_H}$ denotes one of $D^{\mathrm{sum}}_{d_H}$ and $D^{\mathrm{min}}_{d_H}$. A subset $\mathcal{X} \subseteq \Sigma^r$ is said to be Δ-*diverse* w.r.t. $D^{\tau}_{d_H}$ if $D_d(\mathcal{X}) \geqslant \Delta$ for a given $\Delta \geqslant 0$. Let $LCS(\mathcal{S})$ denotes the *set of all longest common subsequences* of a set $\mathcal{S}$ of strings. Now, we state our first problem.

▶ **Problem 1.** Diverse LCSs with Diversity Measure $D^{\tau}_{d_H}$
Input: A set $\mathcal{S} = \{S_1, \ldots, S_m\}$ of $m \geqslant 2$ strings over Σ, integers $K \geqslant 1$, and $\Delta \geqslant 0$;
Question: Is there some set $\mathcal{X} \subseteq LCS(\mathcal{S})$ of longest common subsequences of $\mathcal{S}$ such that $|\mathcal{X}| = K$ and $D^{\tau}_{d_H}(\mathcal{X}) \geqslant \Delta$?

Then, we analyze the computational complexity of Diverse LCSs from the viewpoints of approximation algorithms [37] and parameterized complexity [14, 20]. For this purpose, we actually work with a more general setting, called the Diverse String Set, where a set of string to select is implicitly represented by an edge-labeled DAG G, called a Σ-DAG, that succinctly stores a collection $L(G)$ of strings as the string labels spelled out by all paths from the source s to the sink t. We state our second problem, where $\tau \in \{\text{sum}, \min\}$.

▶ **Problem 2.** DIVERSE STRING SET WITH DIVERSITY MEASURE $D^{\tau}_{d_H}$
Input: Integers K, r, and Δ, and a Σ-DAG G for a set $L(G) \subseteq \Sigma^r$ of r-strings.
Question: Decide if there exists some subset $\mathcal{X} \subseteq L(G)$ such that $|\mathcal{X}| = K$ and $D^{\tau}_{d_H}(\mathcal{X}) \geqslant \Delta$.

Main results. Let $K \geqslant 1$, $r \leqslant 0$, and $\Delta \geqslant 0$ be integers and Σ be an alphabet. The underlying distance is always Hamming distance d_H over r-strings. In DIVERSE STRING SET, an input is a set $L \subseteq \Sigma^r$ of r-strings, which is represented by either a Σ-DAG G or the set L itself. In DIVERSE LCS, an input is $\mathcal{S} \subseteq \Sigma^*$, r denotes the length of all strings in $LCS(\mathcal{S})$, and the number $m = |\mathcal{S}|$ of input strings in our problem is assumed to be constant throughout. Then, the main results of this paper are summarized as follows.

1. When K is bounded, both MAX-SUM and MAX-MIN versions of DIVERSE STRING SET and DIVERSE LCSs can be solved in polynomial time using dynamic programming (DP). (see Theorem 3.1, Theorem 3.2)
2. When K is part of the input, the MAX-SUM version of DIVERSE STRING SET and DIVERSE LCSs admit a PTAS by local search showing that the Hamming distance is a *metric of negative type*[1]. (see Theorem 4.2)
3. The MAX-MIN version of DIVERSE STRING SET and MAX-MIN DIVERSE LCSs are fixed-parameter tractable (FPT) when parameterized by K, r, and Δ (see Theorem 5.1), while the MAX-SUM version of both problems are FPT when parameterized by K and r (see Theorem 5.2). These results are shown by combining Alon, Yuster, and Zwick's *color coding technique* [1] and the DP method in Result 1 above.
4. When K is part of the input, the MAX-SUM and MAX-MIN versions of DIVERSE STRING SET and DIVERSE LCSs are NP-hard for any constant $r \geqslant 3$ (Theorem 6.1, Corollary 6.1).
5. When parameterized by K, the MAX-SUM and MAX-MIN versions of DIVERSE STRING SET and DIVERSE LCSs are W[1]-hard (see Theorem 6.2, Corollary 6.2).

A summary of these results is presented in Table 2. We remark that the DIVERSE STRING SET problem coincides the original LCS problem when $K = 1$. It is generally believed that a W[1]-hard problem is unlikely to be FPT [16, 20].

1.1 Related work

Diversity maximization for point sets in metric space and graphs has been studied since 1970s under various names in the literature [7, 10, 11, 17, 26, 30, 34, 36], including metric spaces and graphs (see Ravi, Rosenkrantz, and Tayi [34] and Chandra and Halldórsson [11] for overview). There are two major versions: MAX-MIN version is known as *remote-edge*, *p-Dispersion*, and *Max-Min Facility Dispersion* [17, 36, 38]; MAX-SUM version is known as *remote-clique*, *Maxisum Dispersion*, and *Max-Average Facility Dispersion* [7, 10, 26, 34]. Both problems are shown to be NP-hard with unbounded K for general distance and metrics (with triangle inequality) [17, 26], while they are polynomial time solvable for 1- and 2-dimensional ℓ_2-spaces [38]. It is trivially solvable in $n^{O(k)}$ time for bounded K.

Diversity maximization in combinatorial problems. However, extending these results for finding diverse solutions to combinatorial problems is challenging [5, 19]. While

[1] It is a finite metric satisfying a class of inequalities of negative type [15]. For definition, see Sec. 4.

Table 2 Summary of results on Diverse String Set and Diverse LCSs problems, where K, r, and Δ stand for the number, the length, and the diversity threshold for a subset $\mathcal{X}$ of r-strings, and α: const, α: param, and α: input indicate that α is a constant, a parameter, and part of an input, respectively. A representation of an input set L is always both of Σ-DAG and LCS otherwise stated.

Problem	Type	K: const	K: param	K: input
Max-Sum Diverse String & LCS	Exact	Poly-Time (Theorem 3.2)	W[1]-hard on Σ-DAG (Theorem 6.2))	NP-hard on Σ-DAG if $r \geqslant 3$:const (Theorem 6.1)
			W[1]-hard on LCS (Corollary 6.2))	NP-hard on LCS (Corollary 6.1)
	Approx. or FPT	—	FPT if r: param (Theorem 5.2)	PTAS (Theorem 4.2)
Max-Min Diverse String & LCS	Exact	Poly-Time (Theorem 3.1)	W[1]-hard on Σ-DAG (Theorem 6.2)	NP-hard on Σ-DAG if $r \geqslant 3$:const (Theorem 6.1)
			W[1]-hard on LCS (Corollary 6.2)	NP-hard on LCS (Corollary 6.1)
	Approx. or FPT	—	FPT if r, Δ: param (Theorem 5.1)	Open

methods such as *random sampling*, *enumeration*, and *top-K optimization* are commonly used for the diversity of solution sets in optimization, they lack theoretical guarantee of the diversity [5, 6, 19, 24]. In this direction, Baste, Fellows, Jaffke, Masarík, de Oliveira Oliveira, Philip, and Rosamond [5, 6] pioneered the study of finding diverse solutions in combinatorial problems, investigating the parameterized complexity of well-know graph problems such as *Vertex Cover* [6]. Subsequently, Hanaka, Kiyomi, Kobayashi, Kobayashi, Kurita, and Otachi [25] explored the fixed-parameter tractability of finding various *subgraphs*. They further proposed a framework for *approximating* diverse solutions, leading to efficient approximation algorithms for diverse matchings, and diverse minimum cuts [24]. While previous work has focused on diverse solutions in graphs and set families, the complexity of finding diverse solutions in *string problems* remains unexplored. Arrighi, Fernau, de Oliveira Oliveira, and Wolf [2] conducted one of the first studies in this direction, investigating a problem of finding a diverse set of subsequence-minimal synchronizing words.

DAG-based representation for all longest common subsequences have appeared from time to time in the literature. The LCS algorithm by Irving and Fraser [29] for more than two strings can be see as DP on a grid DAG for LCSs. Lu and Lin's parallel algorithm [32] for LCS on the CREW PRAM model used a similar grid DAG. Hakata and Imai [23] presented a faster algorithm based on a DAG of *dominant matches*. Conte, Grossi, Punzi, and Uno [12] study succinct DAGs of maximal common subsequences of two strings for enumeration.

The relationship between Hamming distance and other metrics has been explored in string and geometric algorithms. Lipsky and Porat [31] presented linear-time reductions from String Matching problems under Hamming distance to equivalent problems under ℓ_1-metric. Gionis, Indyk, and Motwani [21] used an *isometry* (a distance preserving mapping) from an ℓ_1-metric to Hamming distance over binary strings with a polynomial increase in dimension. Cormode and Muthukrishnan [13] showed an efficient ℓ_1-embedding of edit distance allowing moves over strings into ℓ_1-metric with small distortion. Despite these advancements, existing techniques haven't been successfully applied to to our problems.

2 Preliminaries

We denote by $\mathbb{Z}$, $\mathbb{N} = \{ x \in \mathbb{Z} \mid x \geqslant 0 \}$, $\mathbb{R}$, and $\mathbb{R}_{\geqslant 0} = \{ x \in \mathbb{R} \mid x \geqslant 0 \}$ the sets of *all integers*, *all non-negative integers*, *all real numbers*, and *all non-negative real numbers*, respectively. For any $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, \ldots, n\}$. Let A be any set. Then, $|A|$ denotes the *cardinality* of A, and A^* denotes the set of all finite strings over A.

2.1 Σ-DAGs

Let Σ be an *alphabet* of symbols. A *string set* or a *language* is a set $L = \{X_1, \ldots, X_n\} \subseteq \Sigma^*$ of $n \geqslant 0$ strings over Σ. The *total length* of a string set L is denoted by $||L|| = \sum_{X \in L} |X|$, while the length of the longest strings in L is denoted by $\mathrm{maxlen}(L) := \max_{S \in L} |S|$. We call any string X an *r-string* if its length is r, i.e., $|X| = r$. A *Σ-labeled directed acyclic graph* (*Σ-DAG*, for short) is an edge-labeled directed acyclic graph (DAG) $G = (V, E, s, t)$ satisfying: (i) V is a set of vertices; (ii) $E \subseteq V \times \Sigma \times V$ is a set of labeled directed edges, where each edge $e = (v, c, w)$ is labeled with a symbol $c = \mathrm{lab}(e)$ taken from Σ; (iii) G has the unique *source* s and *sink* t in V such that every vertex lies on a path from s to t. We define the *size* of G, denoted by $\mathrm{size}(G)$, as the number of its labeled edges. Fig. 1 shows an example of Σ-DAG. For any vertex v, we denote the *set of its outgoing edges* by $E^+(v) = \{ (v, c, w) \in E \mid w \in V \}$. Any path $P = (e_1, \ldots, e_n) \in E^n$ *spells out* a string $\mathrm{str}(P) = \mathrm{lab}(e_1) \cdots \mathrm{lab}(e_n) \in \Sigma^n$, where $n \geqslant 0$. A Σ-DAG G represents the string set, or language, denoted $L(G) \subseteq \Sigma^*$, as the collection of all strings spelled out by its (s,t)-paths. Essentially, G is equivalent to a non-deterministic finite automaton (NFA) [28] over Σ with a initial state s, a final state t, and no ε-edges.

▶ **Remark 2.1.** *For any set L of strings, there exists a Σ-DAG G such that $L(G) = L$ and $\mathrm{size}(G) \leqslant ||L||$. Moreover, G can be constructed from L in $O(||L|| \log |\Sigma|)$ time.*

Sometimes, a Σ-DAG with m edges can succinctly represent a string set by its language $L(G)$. Actually, the size of G can be logarithmic in $|L(G)|$ in the best case,[2] while $\mathrm{size}(G)$ can be arbitrary larger than $||L(G)||$ (see Lemma 5.1 in Sec. 5). The next property is useful for handling Σ-DAGs in the following sections.

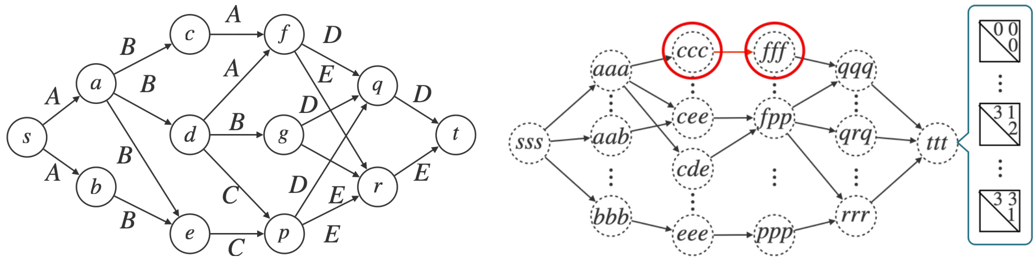
▶ **Remark 2.2.** *If a Σ-DAG G represents a set L of r-strings ($L \subseteq \Sigma^r$, $r \geqslant 0$), then all paths from the source s to any vertex v spell out strings of the same length, say $d \leqslant r$.*

By Remark 2.2, the *depth* of a vertex v in G, denoted $\mathrm{depth}(v)$, is the length of any path P from the source s to v, called a *length-d prefix (path)*. In other words, $\mathrm{depth}(v) = |\mathrm{str}(P)|$. Then, the vertex set V is partitioned into a collection of disjoint subsets $V_0 = \{s\} \cup \cdots \cup V_r = \{t\}$, where V_d is the subset of all vertices with depth d for all $d \in [r] \cup \{0\}$.

2.2 Longest common subsequences

A string X is a *subsequence* of another string Y if X is obtained from Y by removing some characters retaining the order. X is a *common subsequence* (CS) of any set $\mathcal{S} = \{S_1, \ldots, S_m\}$ of m strings if X is a subsequence of any member of $\mathcal{S}$. A CS of $\mathcal{S}$ is called a *longest common subsequence* (LCS) if it has the maximum length among all CSs of $\mathcal{S}$. We denote by $LCS(\mathcal{S})$ the set of all LCSs of $\mathcal{S}$. Naturally, all LCSs in $LCS(\mathcal{S})$ has the same length. While a string

[2] For example, for any $r \geqslant 1$, $L = \{a, b\}^r$ with $|L| = 2^r$ has a Σ-DAG with $2r$ edges, where $\Sigma = \{a, b\}$.



(a) An input Σ-DAG G_1 for $LCS(X_1, Y_1)$ **(b)** Example run of Algorithm 1 on G_1

Figure 1 (a) An input Σ-DAG G_1 over $\Sigma = \{A, B, C, D, E\}$ for the set of all longest common subsequences of two strings $X_1 = ABABCDDEE$ and $Y_1 = ABCBAEEDD$ in Table 1 and (b) an example run of Algorithm 1 based on dynamic programming with $K = 3$ on an input G_1.

set $\mathcal{S}$ can contain exponentially many LCSs compared to the total length $||\mathcal{S}||$ of its strings, we can readily see the next lemma.

▶ **Lemma 2.1** (Σ-DAG for LCSs)**.** *For any constant $m \geqslant 1$ and any set $\mathcal{S} = \{S_1, \ldots, S_m\} \subseteq \Sigma^*$ of m strings, there exists a Σ-DAG G of polynomial size in $\ell := \mathrm{maxlen}(\mathcal{S})$ such that $L(G) = LCS(\mathcal{S})$, and G can be computed in polynomial time in ℓ.*

Proof. By Irving and Fraser's algorithm [29], we can construct a m-dimensional grid graph N in $O(\ell^m)$ time and space, where (i) source and sink: $s = (0, \ldots, 0)$ and $t = (|S_1|, \ldots, |S_m|)$, respectively. (ii) edge labels: symbols from $\Sigma \cup \{\varepsilon\}$. (iii) number of edges: $\mathrm{size}(N) = \prod_{i=1}^{m} |S_i| \leqslant O(\ell^m)$. (iv) path property: all of (s,t)-paths spell out $LCS(\mathcal{S})$. Then, applying the ε-removal algorithm (see, e.g., Hopcroft and Ullman [28]) yields a Σ-DAG G in $O(|\Sigma| \cdot \mathrm{size}(N))$ time and space, where G has $O(|\Sigma| \cdot \mathrm{size}(N)) = O(|\Sigma|\ell^m)$ edges. This completes the proof. ◀

▶ **Remark 2.3.** *As a direct consequence of Lemma 2.1, we observe that if* Max-Min *(resp.* Max-Sum*)* Diverse String Set *can be solved in $f(M, K, r, \Delta)$ time and $g(M, K, r, \Delta)$ space, then* Max-Min *(resp.* Max-Sum*)* Diverse LCSs *on $\mathcal{S} \subseteq \Sigma^r$ can be solvable in $t = O(|\Sigma| \cdot \ell^m + f(\ell^m, K, r, \Delta))$ time and $s = O(\ell^m + g(\ell^m, K, r, \delta))$ space, where $\ell = \mathrm{maxlen}(\mathcal{S})$.*

From Remark 2.3, for any constant $m \geqslant 2$, there exist a polynomial time reduction from Max-Min (resp. Max-Sum) Diverse LCSs for m strings to Max-Min (resp. Max-Sum) Diverse String Set on Σ-DAGs, where the distance measure is Hamming distance.

2.3 Computational complexity

A problem with parameter κ is said to be *fixed-parameter tractable* (FPT) if there is an algorithm that solves it, whose running time on an instance x is $f(\kappa(x)) \cdot |x|^c$ for some computable function $f(\kappa)$ and constant $c > 0$. A many-one reduction ϕ is called an *fpt-reduction* if it can be computed in FPT and the transformed parameter $\kappa(\phi(x))$ is upper-bounded by a computable function of $\kappa(x)$. For notions not defined here, we refer to Ausiello *et al.* [3] for *approximability* and to Flum and Grohe [20] for *parameterized complexity.*

3 Exact Algorithms for Bounded Number of Diverse Strings

In this section, we show that both of Max-Min and Max-Sum versions of Diverse String Set problems can be solved by dynamic programming in polynomial time and space in the size an input Σ-DAG and integers r and Δ for any constant K. The corresponding results for Diverse LCSs problems will immediately follow from Remark 2.3.

Algorithm 1 An exact algorithm for solving Max-Min Diverse r-String problem. Given a Σ-DAG $G = (V, E, s, t)$ representing a set $L(G)$ of r-strings and integers $K \geqslant 1, \Delta \geqslant 0$, decide if there exists some Δ-diverse set of K r-strings in $L(G)$.

Set $\texttt{Weights}(\boldsymbol{s}, Z) := 0$ for all $Z \in (\Delta \cup \{0\})^{K \times K}$, and set $\texttt{Weights}(\boldsymbol{s}, \texttt{Zero}) \leftarrow 1$;
for $d \leftarrow 1, \ldots, r$ **do**
  for $\boldsymbol{v} \leftarrow (v_1, \ldots, v_K) \in (V_d)^K$ **do**
    for $(v_1, c_1, w_1) \in E^+(v_1), \ldots, (v_K, c_K, w_K) \in E^+(v_K)$ **do**
      Set $\boldsymbol{w} \leftarrow (w_1, \ldots, w_K)$;
      for $U \in (\Delta \cup \{0\})^{K \times K}$ such that $\texttt{Weights}(\boldsymbol{v}, U) = 1$ **do**
        Set $Z = (Z_{i,j})_{i<j}$ with $Z_{i,j} \leftarrow \min(\Delta, U_{i,j} + \mathbb{1}\{c_i \neq c_j\}), \forall i, j \in [K]$;
        Set $\texttt{Weights}(\boldsymbol{w}, Z) \leftarrow 1$; ▷ *Update*
Answer YES if $\texttt{Weights}(\boldsymbol{t}, Z) := 1$ and $D^{\min}_{d_H}(Z) \geqslant \Delta$ for some Z, and NO otherwise;

3.1 Computing Max-Min Diverse Solutions

We describe our dynamic programming algorithm for the Max-Min Diverse String Set problem. Given an Σ-DAG $G = (V, E, s, t)$ with n vertices, representing a set $L(G) \subseteq \Sigma^r$ of r-strings, we consider integers $\Delta \geqslant 0$ and $r \geqslant 0$ and constant $K \geqslant 1$. A brute-force approach could solve the problem in $O(|L(G)|^K)$ time by enumerating all combinations of K (s,t)-paths in G and selecting a Δ-diverse solution $\mathcal{X} \subseteq L(G)$. However, this is impractical even for constant K because $|L(G)|$ can be exponential in the number of edges.

The DP-table. Our algorithm relies on *patterns* to efficiently solve the problem using dynamic programming, provided that the number of patterns is manageable. Pattern capture the pairwise Hamming distances for all possible K-tuples of length-d prefixes in a DAG G and for all d $(0 \leqslant d \leqslant r)$. For each d, consider a K-tuple of length-d paths $\boldsymbol{P} = (P_1, \ldots, P_K) \in (E^d)^K$. We define the *pattern* of $\boldsymbol{P}$ as the pair $\texttt{Pattern}(\boldsymbol{P}) = (\boldsymbol{w}, Z)$, where

- $\boldsymbol{w} = (w_1, \ldots, w_K) \in V_d^K$ is the K-tuple of vertices in G such that for all $i \in [K]$, the i-th path P_i starts from the source s and ends at the i-th vertex w_i of $\boldsymbol{w}$.
- $Z = (Z_{i,j})_{i<j} \in ([\Delta] \cup \{0\})^{K \times K}$ is an upper triangular matrix, called the *weight matrix* for $\boldsymbol{P}$. For all $1 \leqslant i < j \leqslant K$, $Z_{i,j} = \min\{\Delta, d_H(\mathrm{str}(P_i), \mathrm{str}(P_j))\} \in [\Delta] \cup \{0\}$ is the Hamming distance between the string labels of P_i and P_j truncated by the threshold Δ.

Then, we define the DP-table $\texttt{Weights}$ as follows.

▶ **Definition 3.1.** $\texttt{Weights} : V^K \times (\Delta \cup \{0\})^{K \times K} \to \{0, 1\}$ *is a Boolean table such that for every* $\boldsymbol{w} \in V^K$ *and* $Z \in (\Delta \cup \{0\})^{K \times K}$*,* $\texttt{Weights}(\boldsymbol{w}, Z) = 1$ *holds if and only if* $(\boldsymbol{w}, Z) = \texttt{Pattern}(\boldsymbol{P})$ *for some* K*-tuple* $\boldsymbol{P}$ *of length-*d *paths from the source* s *to* $\boldsymbol{w}$ *in* G.

We estimate the size of the table $\texttt{Weights}$. Since Z takes at most $\Gamma = O(\Delta^{K^2} K^2)$ distinct values, it can be encoded in $\log \Gamma = O(K^2 \log \Delta)$ bits. Therefore, $\texttt{Weights}$ has at most $|V|^K \times \Gamma = O(\Delta^{K^2} K^2 M^K)$ entries, where $M = \mathrm{size}(G)$. Consequently, $\texttt{Weights}$ can be stored in a multi-dimensional table of polynomial size in M and Δ $t = O(\log |V| + \log \Delta)$ time random access using ballanced binary search trees for any constant K.

Computation of the DP-table. We denote the K-tuples of copies of the source s and sink t as $\boldsymbol{s} := (s, \ldots, s)$ and $\boldsymbol{t} := (t, \ldots, t) \in V^K$, respectively. The *zero matrix* $\texttt{Zero} = (\texttt{Zero}_{i,j})_{i<j}$ is a special matrix where $\texttt{Zero}_{i,j} = 0$ for all $i < j$. Now, we present the recurrence for the DP-table $\texttt{Weights} = (\texttt{Weights}(\boldsymbol{w}, Z))_{\boldsymbol{w}, Z}$.

▶ **Lemma 3.1** (recurrence for `Weights`). *For any* $\boldsymbol{w} \in V^K$ *and* $Z = (Z_{i,j})_{i<j} \in ([\Delta]\cup\{0\})^{K\times K}$, *the entry* $\texttt{Weights}(\boldsymbol{w}, Z) \in \{0,1\}$ *satisfies the following:*

(1) Base case: If $\boldsymbol{w} = \boldsymbol{s}$ *and* $Z = \texttt{Zero}$, *then* $\texttt{Weights}(\boldsymbol{w}, Z) = 1$.

(2) Induction case: If $\boldsymbol{w} \neq \boldsymbol{s}$ *and all vertices in* $\boldsymbol{w}$ *have the same depth* d $(1 \leqslant d \leqslant r)$, *namely,* $\boldsymbol{w} \in V_d^K$, *then* $\texttt{Weights}(\boldsymbol{w}, z) = 1$ *if and only if there exist*

- $\boldsymbol{v} = (v_i)_{i=1}^K \in V_{d-1}^K$ *such that each* v_i *is a parent of* w_i, *i.e.,* $(v_i, c_i, w_i) \in E$, *and*
- $U = (U_{i,j})_{i<j}$ *in* $([\Delta]\cup\{0\})^{K\times K}$, *such that (i)* $\texttt{Weights}(\boldsymbol{v}, U) = 1$, *and (ii)* $Z_{i,j} = U_{i,j} + \mathbb{1}\{c_i \neq c_j\}$ *for all* $1 \leqslant i < j \leqslant K$.

(3) Otherwise: $\texttt{Weights}(\boldsymbol{w}, Z) = 0$.

Proof sketch. Since the proof proceeds by induction on the depth $0 \leqslant d \leqslant r$ of the K components of $\boldsymbol{v}$ and is straightforward, we omit the proof. See Appendix B for details. ◀

Fig. 1 (b) shows an example run of Algorithm 1 on a Σ-DAG G_1 representing the string set $L(G_1) = LCS(X_1, Y_1)$, where G_1 is shown in (a) of Fig. 1. From Lemma 3.1, we show Theorem 3.1 on the correctness and time complexity of Algorithm 1.

▶ **Theorem 3.1** (Polynomial time complexity of Max-Min Diverse String Set). *For any* $K \geqslant 1$ *and* $\Delta \geqslant 0$, *the modified version of Algorithm 1 above solves* MAX-MIN DIVERSE STRING SET *in* $O(\Delta^{K^2} K^2 M^K (\log|V| + \log\Delta))$ *time and space when an input string set* L *is represented by a* Σ*-DAG, where* $M = \text{size}(G)$ *is the number of edges in* G.

3.2 Computing Max-Sum Diverse Solutions

Similar to Algorithm 1, we can solve MAX-SUM DIVERSE STRING SET problem by modifying it as follows. We observe that instead of maintaining the entire $(K \times K)$-weight matrix Z, we only need the sum $z = \sum_{i<j} d_H(\text{str}(P_i), \text{str}(P_j))$ of all pairwise Hamming distances for computing the MAX-SUM diversity.

The new table `Weights'`**:**. For any $\boldsymbol{w} = (w_1, \dots, w_K)$ of the same depth $0 \leqslant d \leqslant r$ and any integer $0 \leqslant z \leqslant rK$, we define: $\texttt{Weights}'(\boldsymbol{w}, z) = 1$ if and only if there exists a K-tuple of length-d prefix paths $(P_1, \dots, P_K) \in (E^d)^K$ from s to $w_1, \dots, w_K$, respectively, such that the sum of their pairwise Hamming distances is z, namely, $z = \sum_{i<j} d_H(\text{str}(P_i), \text{str}(P_j))$.

▶ **Lemma 3.2** (recurrence for `Weights'`). *For any* $\boldsymbol{w} = (w_1, \dots, w_K) \in V^K$ *and any integer* $0 \leqslant z \leqslant rK$, *the entry* $\texttt{Weights}'(\boldsymbol{w}, z) \in \{0,1\}$ *satisfies the following:*

(1) Base case: If $\boldsymbol{w} = \boldsymbol{s}$ *and* $z = 0$, *then* $\texttt{Weights}(\boldsymbol{w}, z) = 1$.

(2) Induction case: If $\boldsymbol{w} \neq \boldsymbol{s}$ *and all vertices in* $\boldsymbol{w}$ *have the same depth* d $(1 \leqslant d \leqslant r)$, *namely,* $\boldsymbol{w} \in V_d^K$, *then* $\texttt{Weights}(\boldsymbol{w}, z) = 1$ *if and only if there exist*

- $\boldsymbol{v} = (v_i)_{i=1}^K \in V_{d-1}^K$ *such that each* v_i *is a parent of* w_i, *i.e.,* $(v_i, c_i, w_i) \in E$, *and*
- $0 \leqslant u \leqslant rK$ *such that (i)* $\texttt{Weights}(\boldsymbol{v}, u) = 1$, *and (ii)* $z = u + \sum_{i<j} \mathbb{1}\{c_i \neq c_j\}$.

(3) Otherwise: $\texttt{Weights}(\boldsymbol{w}, z) = 0$.

From the above modification of Algorithm 1 and Lemma 3.2, we have Theorem 3.2. From this theorem, we see that the MAX-SUM version of DIVERSE STRING SET can be solved faster than the MAX-MIN version by factor of $O(\Delta^{K-1})$.

▶ **Theorem 3.2** (Polynomial time complexity of Max-Sum Diverse String Set). *For any constant* $K \geqslant 1$, *the modified version of Algorithm 1 solves* MAX-SUM DIVERSE STRING SET *under Hamming Distance in* $O(\Delta K^2 M^K (\log|V| + \log\Delta))$ *time and space, where* $M = \text{size}(G)$ *is the number of edges in* G *and the input set* L *is represented by a* Σ*-DAG.*

4 Approximation Algorithm for Unbounded Number of Diverse Strings

Algorithm 2 A $(1-2/K)$-approximation algorithm for Max-Sum Diversification for a metric d of negative type on $\mathcal{X}$.

---

procedure LocalSearch$(\mathcal{L}, K, d)$;
$\mathcal{X} \leftarrow$ arbitrary K solutions in $\mathcal{L}$;
for $i \leftarrow 1, \ldots, \lceil \frac{K(K-1)}{(K+1)} \ln \frac{(K+2)(K-1)^2}{4} \rceil$ **do**
  for $X \in \mathcal{X}$ such that $\mathcal{L} \setminus \mathcal{X} \neq \emptyset$ **do**
    $Y \leftarrow \operatorname{argmax}_{Y \in \mathcal{L} \setminus \mathcal{X}} \sum_{X' \in \mathcal{X} \setminus \{X\}} d(X', Y)$;
    $\mathcal{X} \leftarrow (\mathcal{X} \setminus \{X\}) \cup \{Y\}$;
Output $\mathcal{X}$;

To solve Max-Sum Diverse String Set, we employ the local search procedure in Algorithm 2 by Cevallos, Eisenbrand, and Zenklusen [10] for computing approximate solutions $\mathcal{X} \subseteq \mathcal{L}$ with $|\mathcal{X}| = K$ on a finite metric space $\mathcal{L}$ under distance $d : \mathcal{L} \times \mathcal{L} \to \mathbb{R}_{\geqslant 0}$. We introduce some notions on metrics according to [15]. Let $\mathcal{X} = \{x_1, \ldots, x_n\}$ be a finite set of $n \geqslant 1$ elements, and d be a semi-metric over $\mathcal{X}$. A semi-metric is a function $f : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geqslant 0}$ satisfying the following conditions (i)–(iii): (i) $d(x, y) = 0, \forall x \in \mathcal{X}$ (zero); (ii) $d(x, y) = d(y, x), \forall x, y \in \mathcal{X}$ (symmetry). (iii) $d(x, z) \leqslant d(x, y) + d(y, z), \forall x, y, z \in \mathcal{X}$ (triangle inequalities). Consider an inequality condition in the following form, called a *negative inequality*:

$$\boldsymbol{b}^\top D\, \boldsymbol{b} := \textstyle\sum_{i<j} b_i b_j d(x_i, x_j) \leqslant 0, \qquad \forall \boldsymbol{b} = (b_1, \ldots, b_n) \in \mathbb{Z}^n, \tag{3}$$

where $\boldsymbol{b}$ is a column vector and $D = (d_{ij})$ with $d_{ij} = d(x_i, x_j)$. For the vector $\boldsymbol{b}$ above, we define $\sum \boldsymbol{b} := \sum_{i=1}^n b_i$. A semi-metric d is said to be *of negative type* if it satisfies the inequalities Eq. (3) for all $\boldsymbol{b} \in \mathbb{Z}^n$ such that $\sum \boldsymbol{b} = 0$. The class $\mathbb{NEG}$ of semi-metrics of negative type satisfies the following properties.

▶ **Lemma 4.1** (Deza and Laurent [15]). *For the class $\mathbb{NEG}$, the following properties hold: (1) All ℓ_1-metrics over $\mathbb{R}^r$ are semi-metrics of negative type for any $r \geqslant 1$. (2) The class $\mathbb{NEG}$ is closed under linear combination with nonnegative coefficients.*

Cevallos *et al.* [10] showed that when the distance d is a semi-metric of *negative type*, the procedure LocalSearch in Algorithm 2 has improved approximation ratio $(1 - \frac{2}{K})$. As a direct consequence of this result, [10] showed the following theorem (See also [24]).

▶ **Theorem 4.1** (Cevallos *et al.* [10]). *Suppose that d is a metric of negative type over $\mathcal{X}$ in which the Farthest Point problem can be solved in polynomial time. For any $K \geqslant 1$, the procedure LocalSearch in Algorithm 2 has approximation ratio $(1 - \frac{2}{K})$.*

We show that the Hamming distance actually has the desired property.

▶ **Lemma 4.2.** *For any integer r, the Hamming distance d_H over the set Σ^r of r-strings is a semi-metric of negative type over Σ^r.*

Proof. We can give an isometry ϕ (see Sec. 1.1) from the Hamming distance (Σ^r, d_H) to the ℓ_1-metric (W, d_{ℓ_1}) over a subset W of $\mathbb{R}^m$ for constant $m = r\sigma$. Let $\Sigma = [\sigma]$ be any alphabet. For any symbol $i \in \Sigma$, we extend ϕ by $\phi_\Sigma(i) := 0^{i-1}(0.5)0^{n-i} \in \{0, 0.5\}^\sigma$

Algorithm 3 An exact algorithm for solving the MAX-SUM FARTHEST r-STRING problem. Given a Σ-DAG $G = (V, E, s, t)$ representing a set $L(G)$ of r-strings, a set $\mathcal{X} = \{X_1, \ldots, X_K\}$ of r-strings, and an integer $\Delta \geqslant 0$, it decides if there exists some r-string Y in $L(G)$ such that $D^{\text{sum}}_{d_H}(\mathcal{X} \cup \{Y\}) \geqslant \Delta$, where $[\Delta]_+ = [\Delta] \cup \{0\}$.

1 Set $\texttt{Weights}(s, z) := 0$ for all $z \in [\Delta]_+$, and $\texttt{Weights}(s, 0) := 1$;
2 **for** $d := 1, \ldots, r$ **do**
3   **for** $v \in V_d$ and $(v, c, w) \in E^+(v)$ **do**
4   **for** $u \in [\Delta]_+$ such that $\texttt{Weights}(v, u) := 1$ **do**
5   Set $\texttt{Weights}(w, z) := 1$ for $z := u + \sum_{i \in [K]} \mathbb{1}\{c \neq X_i[d]\}$; ▷ *Update*
6 Answer YES if $\texttt{Weights}(t, \Delta) = 1$, and NO otherwise ; ▷ *Decide*

be a bitvector with 0.5 at i-th position and 0 at other bit positions such that for each $c, c' \in \Sigma$, $||\phi_\Sigma(c) - \phi_\Sigma(c')||_1 = \mathbb{1}\{c \neq c'\}$. For any r-string $S = S[1] \ldots S[r] \in \Sigma^r$, we let $\phi(S) := \phi_\Sigma(S[1]) \cdots \phi_\Sigma(S[r]) \in W$, where $W := \{0, 0.5\}^m$ and $m := r\sigma$. For any $S, S' \in \Sigma^r$, we can show $d_{\ell_1}(\phi(S), \phi(S')) = ||\phi(S)_j - \phi(S')_j||_1 = \sum_{i \in [r]} ||\phi_\Sigma(S[i]) - \phi_\Sigma(S'[i])||_1 = \sum_{i \in [r]} \mathbb{1}\{S[i] \neq S'[i]\} = d_H(S, S')$. Thus, $\phi : \Sigma^r \to W$ is an *isometry* [15] from (Σ^r, d_H) to $(\{0, 0.5\}^m, d_{\ell_1})$. By Lemma 4.1, ℓ_1-metric is a metric of negative type [10, 15], and thus, the lemma is proved. ◀

The remaining thing is efficiently solving the subproblem at Line 5, called the FARTHEST STRING problem in our case, that asks to find the *farthest* Y from all elements but X in $\mathcal{X}$ by maximizing the sum $\sum_{X' \in \mathcal{X} \setminus \{X\}} d(X', Y)$ over all elements $Y \in \mathcal{L} \setminus \mathcal{X}$. For the class of r-strings, we see that the Farthest String can be efficiently found when K is a constant.

▶ **Lemma 4.3** (MAX-SUM FARTHEST r-STRING). *For any $K \geqslant 1$ and $\Delta \geqslant 0$, Algorithm 3 computes the farthest r-string $Y \in L(G)$ that maximizes $D^{\text{sum}}_{d_H}(\mathcal{X} \cup \{Y\})$ over all r-strings in $L(G)$ in $O(K\Delta m)$ time and space, where M is the number of edges in G.*

Proof. We show Algorithm 3 for MAX-SUM FARTHEST r-STRING problem. It is a modification of Algorithm 1 by fixing $K - 1$ paths and searching only a remaining path in G. Therefore, its correctness and time complexity immediately follows from that of Theorem 3.1. ◀

Combining Theorem 4.1, Lemma 4.3, and Lemma 4.2, we obtain the following theorem on the existence of a *polynomial time approximation scheme* (PTAS) [3] for MAX-SUM DIVERSE STRING SET on Σ-DAGs. From Theorem 4.2 and Remark 2.3, the corresponding result for MAX-SUM DIVERSE LCSs will immediately follow.

▶ **Theorem 4.2** (PTAS for unbounded K). *When K is part of an input, MAX-SUM DIVERSE STRING SET problem on a Σ-DAG G admits PTAS.*

Proof. We show the theorem following the discussion in [10, 24]. Let $\varepsilon > 0$ be any constant. Suppose that $\varepsilon < 2/K$ holds. Then, $K < 2/\varepsilon$, and thus, k is a constant. In this case, by Theorem 3.1, we can exactly solve the problem in polynomial time using Algorithm 1. Otherwise, $2/K \leqslant \varepsilon$. Then, the $(1 - 2/K)$ approximation algorithm in Algorithm 2 equipped with Algorithm 3 achieves factor $1 - \varepsilon$ since d_H is a negative type metric by Lemma 4.2. Hence, MAX-SUM DIVERSE STRING SET admits a PTAS. This completes the proof. ◀

5 FPT Algorithms for Bounded Number and Length of Diverse Strings

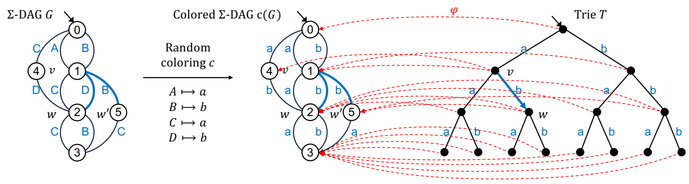


Figure 2 Illustration of the procedure in the proof for Lemma 5.1, where red dashed lines indicates a correspondence φ.

In this section, we present fixed-parameter tractable (FPT) algorithms for the Max-Min and Max-Sum Diverse String Set parameterized with combinations of K, r, and Δ. Recall that a problem parameterized with κ is said to be *fixed-parameter tractable* if there exists an algorithm for the problem running on an input x in time $f(\kappa(x)) \cdot |x|^c$ for some computable function $f(\kappa)$ and constant $c > 0$ [20].

For our purpose, we combine the *color-coding technique* by Alon, Yuster, and Zwick [1] and the dynamic programming algorithms given in Sec. 3. Before presenting the main result of this section, we present a technical lemma for a subproblem on reduction of an input Σ-DAG G. Consider a random C-coloring $c : \Sigma \to C$ from a set C of $k \geqslant 1$ colors. We denote by $c(G)$ the colored C-DAG obtained from G by coloring all edges with c.

▶ **Lemma 5.1** (computing a reduced C-DAG in FPT)**.** *For any set C of k colors, there exists some C-DAG H obtained by reducing $c(G)$ such that $L(H) = L(c(G))$ and $||H|| \leqslant k^r$. Furthermore, such a C-DAG H can be computed from G and C in $t_{\mathrm{pre}} = O(k^r \cdot \mathrm{size}(G))$ time and space.*

Proof sketch. Since $L(G) \subseteq \Sigma^r$, we see that the C-DAG $c(G)$ represents $L(c(G)) \subseteq C^r$ of size at most $||L(c(G))|| \leqslant k^r$. By Remark 2.1, there exists a C-DAG H for $L(H) = L(c(G))$ with at most k^r edges. However, it is not straightforward how to compute such a succinct H directly from G and c in $O(k^r \cdot \mathrm{size}(G))$ time and space since $||L(G)||$ can be much larger than $k^r + \mathrm{size}(G)$. For this purpose, using a procedure BuildColoredTrie (see Algorithm 4 in Appendix C), we first build a trie T for $L(H)$ top-down using breadth-first search of G from the source s by maintaining a correspondence $\varphi \subseteq V \times U$ between vertices in G and T (Fig. 2). Then, we identify all leaves of T to make the sink t. This runs in $O(k^r \cdot \mathrm{size}(G))$ time and $O(k^r + \mathrm{size}(G))$ space. See Appendix C for the details. ◀

Fig. 2 illustrates computation of reduced C-DAG in Lemma 5.1, which shows an input Σ-DAG G over alphabet $\Sigma = \{A, B, C, D\}$ (left), a random coloring c on a color set $C = \{a, b\}$, a colored C-DAG $c(G)$ (middle), and a C-DAG H in the form of a trie T (right). Combining Lemma 5.1, Theorem 3.1, and Alon *et al.* [1], we show the next theorem.

▶ **Theorem 5.1.** *When r, K, and Δ are parameters, the Max-Min Diverse String Set on a Σ-DAG for r-strings is fixed-parameter tractable (FPT), where $m = \mathrm{size}(G)$ is part of an input.*

Proof. We show a sketch of the proof. See Appendix C for the full proof. We show a randomized algorithm using Alon *et al.*'s color-coding technique [1]. Let $L(G) \subseteq \Sigma^r, k = rK$, and $C = [rK]$. We randomly color edges of G from C. Then, we perform two phases below.

- *Preprocessing phase*: Using the fpt-algorithm of Lemma 5.1, reduce the colored C-DAG $c(G)$ with $\mathrm{size}(G)$ into another C-DAG H with $L(H) = L(c(G)) \subseteq C^r$ and size bounded by $(rK)^r$. Lemma 5.1 shows that this requires $t_{\mathrm{pre}} = O((rK)^r \cdot \mathrm{size}(G))$ time and space.
- *Search phase*: Find a Δ-diverse subset $\mathcal{Y}$ in $L(H)$ of size $|\mathcal{Y}| = K$ from H using a modified version of Algorithm 1 in Sec. 3 (details in footnote[3]). If such $\mathcal{Y}$ exists and c is

[3] This modification of Algorithm 1 is easily done at Line 7 of Algorithm 1 by checking if both of $\mathbb{1}\{c(lab(e_i)) \neq c(lab(e_j))\}$ and $\mathbb{1}\{lab(e_i) \neq lab(e_j)\}$ hold, instead of checking if $\mathbb{1}\{lab(e_i) \neq lab(e_j)\}$.

invertible, then $\mathcal{X} = c^{-1}(\mathcal{Y})$ is a Δ-diverse solution for the original problem. The search phase takes $t_{\text{search}} = O(K^2\Delta^{K^2}(rK)^{rK}) =: g(K, r, \Delta)$ time.

With the probability $p = (rK)!/(rK)^{rK} \geqslant 2^{-rK}$, for $C = [rK]$, the random C-coloring yields a colorful Δ-diverse subset $\mathcal{Y} = c(\mathcal{X}) \subseteq L(H)$. Repeating the above process 2^{rK} times and derandomizing it using Alon *et al.* [1] yields an FPT algorithm with total running time $t = 2^{rK} r \log(rK)(t_{\text{pre}} + t_{\text{search}}) = f(K, r, \Delta) \cdot \text{size}(G)$, where $f(K, r, \Delta) = O(2^{rK} r \log(rK) \cdot \{(rK)^r + g(K, r, \Delta)\}$ depends only on parameters. This completes the proof. ◀

For Max-Sum Diversity, we only need the parameterization with only K and r, excluding Δ, to obtain the following result.

▶ **Theorem 5.2.** *When r and K are parameters, the Max-Sum Diverse String Set on Σ-graphs for r-strings is fixed-parameter tractable (FPT), where $m = \text{size}(G)$ is part of an input.*

Proof. The proof proceeds by a similar discussion to one in the proof of Theorem 5.1. The only difference is the time complexity of t_{search}. In the case of Max-Sum diversity, the search time of the modified algorithm in Theorem 3.2 is $t_{\text{search}} = O(\Delta K^2 M_*^K)$. By substituting this bound $M \leqslant (rK)^k$, we have $t_{\text{search}} = g'(K, r)\Delta$, where $g'(K, r) := O(K^2(rK)^{rK})$. Since $g'(K, r)$ depends only on parameters K and r, the algorithm is FPT. ◀

6 Hardness results

To complement the positive results in Sec. 3 and Sec. 4, we show some negative results in classic and parameterized complexity. In what follows, $\sigma = |\Sigma|$ is an alphabet size, K is the number of strings to select, r is the length of equi-length strings, and Δ is a diversity threshold. In all results below, we assume that σ are constants, and without loss of generality that an input set L of r-strings is explicitly given as the set itself.

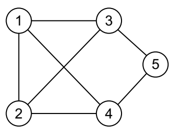
6.1 Hardness of Diverse String Set for Unbounded K

Firstly, we observe the NP-hardness of Max-Min and Max-Sum Diverse String Set holds for unbounded K even for constants $r \geqslant 3$.

▶ **Theorem 6.1** (NP-hardness for unbounded K)**.** *When K is part of an input, Max-Min and Max-Sum Diverse String Set on Σ-graphs for r-strings are NP-hard even for any constant $r \geqslant 3$.*

Proof sketch. We reduce an NP-hard problem 3DM to Max-Min Diverse String Set. See Appendix D for the details. ◀

We remark that 3DM is shown to be in FPT by Fellows, Knauer, Nishimura, Ragde, Rosamond, Stege, Thilikos, and Whitesides [18]. Besides, we showed in Sec. 5 that Diverse r-String Set is FPT when parameterized with $K + r$ (Max-Sum) or $K + r + \Delta$ (Max-Min), respectively. We show that the latter problem is W[1]-hard parameterized with K.



	12	13	14	15	23	24	25	34	35	45
S_1	1	1	1	0	1	1	1	1	1	1
S_2	2	2	2	2	2	2	0	2	2	2
S_3	3	3	3	3	3	3	3	0	3	3
S_4	4	4	4	4	4	4	4	0	4	4
S_5	5	5	5	0	5	5	0	5	5	5

Figure 3 An example of reduction from Clique (left) to Diverse r-String Set (right).

▶ **Theorem 6.2** (W1-hardness of the string set and Σ-DAG versions for unbounded K)**.** *When parameterized with K, Max-Min and Max-Sum Diverse String for a set L of r-strings are W[1]-hard whether a string set L is represented by either a string set L or a Σ-DAG for L, where r and Δ are part of an input.*

Proof. We establish the W[1]-hardness of Max-Min Diverse String parameterized with K by reduction from Clique parameterized with K. This builds on the NP-hardness of r-Set packing in Ausiello *et al.* [4] with minor modifications (see also [18]). Let $\mathcal{E} := \{ \{i, j\} \mid i, j \in V, i \neq j \}$. Given a graph $G = (V, E)$ with n vertices and a parameter $K \in \mathbb{N}$, where $V = [n]$ and $E \subseteq \mathcal{E}$, we define the transformation ϕ_1 from $\langle G, K\rangle$ to $\langle \Sigma, r, F, \Delta\rangle$ and $\kappa(K) = K$ as follows. We let $\Sigma = [n] \cup \{0\}$, $r = |\mathcal{E}| = \binom{n}{2}$, and $\Delta = r$. We view each r-string S as a mapping $S : \mathcal{E} \to \Sigma$ assigning symbol $S(e) \in \Sigma$ to each unordered pair $e \in \mathcal{E}$. We construct a family $F = \{ S_i \mid i \in V \}$ of r-strings such that G has a clique of K elements $\iff$ there exists a subset $M \subseteq F$ with (i) size $|M| \geqslant \kappa(K) = K$, and (ii) diversity $d_H(S, S') \geqslant r = \Delta$ for all distinct $S, S' \in M$ (*). Each r-string S_i is defined based on the existence of the edges in E: (i) $S_i(e) = 0$ if $(i \in e) \wedge (e \notin E)$, and (ii) $S_i(e) = i$ otherwise. Fig. 3 illustrates construction of F from G and $K = 3$ (left), where $n = 5$. This ensures that connecting vertices in E have no potisions with the same symbol (i.e., *match* in [23]) in their corresponding strings, while unconnected vertices have some positions with the same symbols. Therefore, a Δ-diverse solution of size K exists if and only if the corresponding vertices in G form a K-clique. This establish the correctness of the reduction, proving that ϕ_1 and κ form an fpt-reduction from clique to Max-Min Diverse String Set. ◀

6.2 Hardness of Diverse LCSs for Unbounded K

Consider the set $LCS(S_1, S_2)$ of all longest common subsequence of two strings S_1, S_2. To each LCS $X \in LCS(S_1, S_2)$ of length $r = |X|$, we can associate a *non-crossing* matching M in a bipartite graph $\mathcal{B} = (Pos_1 \cup Pos_2, E)$, where Pos_1 and Pos_2 are the sets of all positions in S_1 and S_2, respectively, and $\mathcal{B}$ has an edge (i, j) if and only if positions i and j have the same symbols in S_1 and S_2, respectively.

▶ **Theorem 6.3.** *Under Hamming distance, Max-Min (resp. Max-Sum) Diverse String Set for $m \geqslant 2$ strings parameterized with K is fpt-reducible to Max-Min (resp. Max-Sum) Diverse LCSs for two string ($m = 2$) parameterized with K, where m is part of an input. Moreover, the reduction is also a polynomial time reduction from Max-Min (resp. Max-Sum) Diverse String Set to Max-Min (resp. Max-Sum) Diverse LCSs.*

Proof sketch. We omit the proof due to space constraint. See Appendix D for the proof. ◀

Combining Theorem 6.1, Theorem 6.2, and Theorem 6.3, we have the corollaries.

▶ **Corollary 6.1** (NP-hardness). *When K is part of an input, Max-Min and Max-Sum Diverse LCSs for two r-strings are NP-hard, where r and Δ are part of an input.*

▶ **Corollary 6.2** (W1-hardness). *When parameterized with K, Max-Min and Max-Sum Diverse LCSs for two r-strings are W[1]-hard, where r and Δ are part of an input.*

7 Conclusion

We studied the computational complexity of the Max-Min and Max-Sum versions of Diverse String Set and Diverse LCSs problems. We show the complexity of exact, approximate, and parameterized solutions for each problem. The approximability of Max-Min Diverse String Set remains an interesting open problem. Future work includes extending our results to other string distances, e.g., *edit distance*, and applying our proposed methods to other diversity maximization problems on strings, whose feasible solutions are represented by Σ-DAGs. Details will be provided in the full paper.

References

1. Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995.
2. Emmanuel Arrighi, Henning Fernau, Mateus de Oliveira Oliveira, and Petra Wolf. Synchronization and diversity of solutions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37(10), pages 11516–11524, 2023.
3. Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela, and Marco Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 2012.
4. Giorgio Ausiello, Alessandro D'Atri, and Marco Protasi. Structure preserving reductions among convex optimization problems. *Journal of Computer and System Sciences*, 21(1):136–153, 1980.
5. Julien Baste, Michael R Fellows, Lars Jaffke, Tomáš Masařík, Mateus de Oliveira Oliveira, Geevarghese Philip, and Frances A Rosamond. Diversity of solutions: An exploration through the lens of fixed-parameter tractability theory. *Artificial Intelligence*, 303:103644, 2022.
6. Julien Baste, Lars Jaffke, Tomáš Masařík, Geevarghese Philip, and Günter Rote. FPT algorithms for diverse collections of hitting sets. *Algorithms*, 12(12):254, 2019.
7. Benjamin Birnbaum and Kenneth J Goldman. An improved analysis for a greedy remote-clique algorithm using factor-revealing LPs. *Algorithmica*, 55(1):42–59, 2009.
8. Hans L Bodlaender, Rodney G Downey, Michael R Fellows, and Harold T Wareham. The parameterized complexity of sequence alignment and consensus. *Theoretical Computer Science*, 147(1-2):31–54, 1995.
9. Laurent Bulteau, Mark Jones, Rolf Niedermeier, and Till Tantau. An FPT-algorithm for longest common subsequence parameterized by the maximum number of deletions. In *33rd Annual Symposium on Combinatorial Pattern Matching (CPM 2022)*, pages 6:1–6:11, 2022.
10. Alfonso Cevallos, Friedrich Eisenbrand, and Rico Zenklusen. An improved analysis of local search for max-sum diversification. *Mathematics of Operations Research*, 44(4):1494–1509, 2019.
11. Barun Chandra and Magnús M Halldórsson. Approximation algorithms for dispersion problems. *Journal of Algorithms*, 38(2):438–465, 2001.
12. Alessio Conte, Roberto Grossi, Giulia Punzi, and Takeaki Uno. A compact DAG for storing and searching maximal common subsequences. In *ISAAC 2023*, pages 21:1–21:15, 2023.
13. Graham Cormode and Shan Muthukrishnan. The string edit distance matching problem with moves. *ACM Transactions on Algorithms (TALG)*, 3(1):1–19, 2007.
14. Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 1st edition, 2015.
15. Michel Deza and Monique Laurent. *Geometry of Cuts and Metrics*, volume 15 of *Algorithms and Combinatorics*. Springer, 1997.
16. Rodney G Downey and Michael Ralph Fellows. *Parameterized complexity*. Springer, 2012.
17. Erhan Erkut. The discrete p-dispersion problem. *European journal of operational research*, 46(1):48–60, 1990.
18. Michael R Fellows, Christian Knauer, Naomi Nishimura, Prabhakar Ragde, F Rosamond, Ulrike Stege, Dimitrios M Thilikos, and Sue Whitesides. Faster fixed-parameter tractable algorithms for matching and packing problems. *Algorithmica*, 52:167–176, 2008.
19. Michael R. Fellows and Frances Rosamond. The DIVERSE X Paradigm (Open problems). In Henning Fernau, Petr Golovach, Marie-France Sagot, et al., editors, *Algorithmic enumeration: Output-sensitive, input-sensitive, parameterized, approximative (Dagstuhl Seminar 18421), Dagstuhl Reports, 8(10)*, 2019.
20. J. Flum and M. Grohe. *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series)*. Springer, 2006.

21 Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *VLDB*, volume 99(6), pages 518–529, 1999.

22 Dan Gusfield. Efficient methods for multiple sequence alignment with guaranteed error bounds. *Bulletin of Mathematical Biology*, 55(1):141–154, 1993.

23 Koji Hakata and Hiroshi Imai. The longest common subsequence problem for small alphabet size between many strings. In *ISAAC'92*, pages 469–478. Springer, 1992.

24 Tesshu Hanaka, Masashi Kiyomi, Yasuaki Kobayashi, Yusuke Kobayashi, Kazuhiro Kurita, and Yota Otachi. A framework to design approximation algorithms for finding diverse solutions in combinatorial problems. In *AAAI 2023*, pages 3968–3976, 2023.

25 Tesshu Hanaka, Yasuaki Kobayashi, Kazuhiro Kurita, and Yota Otachi. Finding diverse trees, paths, and more. In *AAAI 2021*, pages 3778–3786, 2021.

26 P Hansen and D Moon. *Dispersing facilities on a network.* Rutgers University. Rutgers Center for Operations Research [RUTCOR], 1988. Presentation at the TIMS/ORSA Joint National Meeting, Washington, D.C.

27 Daniel S Hirschberg. *Recent results on the complexity of common-subsequence problems, in Time warps, String edits, and Macromolecules*, pages 323–328. Addison-Wesley, 1983.

28 John E. Hopcroft and Jeff D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

29 Robert W. Irving and C. B. Fraser. Two algorithms for the longest common subsequence of three (or more) strings. In *CPM 1992*, pages 214–229, 1992.

30 Michael J Kuby. Programming models for facility dispersion: The p-dispersion and maxisum dispersion problems. *Geographical Analysis*, 19(4):315–329, 1987.

31 Ohad Lipsky and Ely Porat. L1 pattern matching lower bound. *Information Processing Letters*, 105(4):141–143, 2008.

32 Mi Lu and Hua Lin. Parallel algorithms for the longest common subsequence problem. *IEEE Transactions on Parallel and Distributed Systems*, 5(8):835–848, 1994.

33 David Maier. The complexity of some problems on subsequences and supersequences. *Journal of the ACM*, 25(2):322–336, 1978.

34 Sekharipuram S Ravi, Daniel J Rosenkrantz, and Giri Kumar Tayi. Heuristic and special case algorithms for dispersion problems. *Operations research*, 42(2):299–310, 1994.

35 David Sankoff. Matching sequences under deletion/insertion constraints. *Proceedings of the National Academy of Sciences*, 69(1):4–6, 1972.

36 Douglas R Shier. A min-max theorem for p-center problems on a tree. *Transportation Science*, 11(3):243–252, 1977.

37 Vijay V. Vazirani. *Approximation Algorithms.* Springer, 2010.

38 Da-Wei Wang and Yue-Sun Kuo. A study on two geometric location problems. *Information Processing Letters*, 28(6):281–286, 1988.

A Proofs for Section 2 (Preliminaries)

▶ **Remark 2.1.** *For any set L of strings, there exists a Σ-DAG G such that $L(G) = L$ and* $\text{size}(G) \leqslant ||L||$. *Moreover, G can be constructed from L in $O(||L|| \log |\Sigma|)$ time.*

Proof. We can construct a *trie* [Gus97] T for a set L of strings over Σ in $O(||L|| \log \sigma)$ time, which is a deterministic finite automaton for recognizing L in the shape of a rooted trees and has at most $O(||L||)$ vertices and edges. By identifying all leaves of T to form the sink, we obtain a Σ-DAG with $||L||$ edges for L. ◀

B Proofs for Section 3 (Exact Algorithms for Bounded Number of Diverse Strings)

▶ **Lemma 3.1** (recurrence for `Weights`). *For any $\boldsymbol{w} \in V^K$ and $Z = (Z_{i,j})_{i<j} \in ([\Delta] \cup \{0\})^{K \times K}$, the entry $\texttt{Weights}(\boldsymbol{w}, Z) \in \{0, 1\}$ satisfies the following:*

(1) Base case: If $\boldsymbol{w} = \boldsymbol{s}$ and $Z = \texttt{Zero}$, then $\texttt{Weights}(\boldsymbol{w}, Z) = 1$.

(2) Induction case: If $\boldsymbol{w} \neq \boldsymbol{s}$ and all vertices in $\boldsymbol{w}$ have the same depth d $(1 \leqslant d \leqslant r)$, namely, $\boldsymbol{w} \in V_d^K$, then $\texttt{Weights}(\boldsymbol{w}, z) = 1$ if and only if there exist

- *$\boldsymbol{v} = (v_i)_{i=1}^K \in V_{d-1}^K$ such that each v_i is a parent of w_i, i.e., $(v_i, c_i, w_i) \in E$, and*
- *$U = (U_{i,j})_{i<j}$ in $([\Delta] \cup \{0\})^{K \times K}$, such that (i) $\texttt{Weights}(\boldsymbol{v}, U) = 1$, and (ii) $Z_{i,j} = U_{i,j} + \mathbb{1}\{c_i \neq c_j\}$ for all $1 \leqslant i < j \leqslant K$.*

(3) Otherwise: $\texttt{Weights}(\boldsymbol{w}, Z) = 0$.

Proof. The lemma is proved by induction on $0 \leqslant d \leqslant r$. (1) Base case: If $d = 0$, the claim is obvious since s is the only vertices of depth 0, and $\boldsymbol{P} = (\varepsilon)_{i=1}^K$ is the K-tuple of empty paths. Case (3) is obvious. (2) Induction case: Suppose that $d > 0$. Let $\boldsymbol{w} = (w_1, \ldots, w_K)$. By definition, there are some K-tuple $\boldsymbol{P} = (P_i)_{i=1}^K \in (E^d)^K$ of length-d prefixes. Since $d > 0$, we let $P_i = Q_i \cdot e_i$ for some $Q_i = P_i[1..d-1] \in E^{d-1}$ and some $e_i = (v_i, c_i, w_i) \in E$. By induction hypothesis, we observe that $\texttt{Weights}(\boldsymbol{v}, U) = 1$ for some weight matrix $U = (U_{i,j})_{i<j}$ such that $U_{i,j} = d_H(Q_i, Q_j)$ (*). Now, we have that

$$
\begin{aligned}
W_{i,j} &= d_H(P_i, P_j) \\
&= d_H(Q_i, Q_j) + d_H(P_i[d], P_j[d]) \\
&= U_{i,j} + \mathbb{1}\{c_i \neq c_j\},
\end{aligned}
$$

for all i, j with $i < j$. Since $\texttt{Weights}(\boldsymbol{v}, U) = 1$, the claim of case (2) follows from the above derivation. Finally, the case (3) is obvious. Combining (1)–(3), the lemma is proved. ◀

C Proofs for Section 5 (FPT Algorithms for Bounded Number and Length of Diverse Strings)

In Algorithm 4, we present the procedure BuildColoredTrie in Lemma 5.1 for computation of a reduced C-DAG from an input Σ-DAG and a random coloring c. An example of the execution of the procedure will be found in Fig. 2 of Sec. 5. The procedure uses a *trie* for a set L of strings, which is a (deterministic) finite automaton in the form of a rooted tree that represents strings in L as the string labels spelled out by all paths from the root to its leaves. We show the next lemma.

Algorithm 4 The procedure BuildColoredTrie in Lemma 5.1 for computing a C-DAG H such that $L(H) = L(c(G))$ and $||H|| \leqslant k^r$ from a Γ-DAG G and a coloring $c : \Sigma \to C$ with $|C| = k$. (See the proof of Theorem 5.1)

Procedure BuildColoredTrie($G = (V, E), c : \Sigma \to C$);
    Variable: A trie $T = (U = \bigcup_d U_d, \texttt{goto}, \mathit{root})$;
$U_0 \leftarrow \{\mathit{root}\}$ and $\varphi(\mathit{root}) \leftarrow \{s\}$ for the new vertex *root* in T;
$\texttt{goto} \leftarrow \emptyset$; $\texttt{visited} \leftarrow \emptyset$;
for $d := 1, \ldots, r$ **do**
    for $x \in U_{d-1}$ **do** ▷ *A vertex x in T*
        for $v \in \varphi(x)$ **do** ▷ *A vertex v in G*
            if $\texttt{visited}(v) = \perp$ **then**
                $\texttt{visited}(v) \leftarrow 1$;
                for $e = (v, a, w) \in E^+(v)$ **do** ▷ *An edge e in G*
                    $c \leftarrow c(a)$;
                    if $\texttt{goto}(x, c) = \perp$ **then**
                        $\texttt{goto}(x, c) \leftarrow$ a new vertex in T;
                        $U_d \leftarrow U_d \cup \{y\}$;
                    $y \leftarrow \texttt{goto}(x, c)$; ▷ *a vertex y in T*
                    $\varphi(y) \leftarrow \varphi(y) \cup \{w\}$;
Let H be the Σ-DAG obtained from the trie T by merging all leaves;
Return H;

▶ **Lemma 5.1** (computing a reduced C-DAG in FPT). *For any set C of k colors, there exists some C-DAG H obtained by reducing $c(G)$ such that $L(H) = L(c(G))$ and $||H|| \leqslant k^r$. Furthermore, such a C-DAG H can be computed from G and C in $t_{\mathrm{pre}} = O(k^r \cdot \mathrm{size}(G))$ time and space.*

Proof. Let $c : \Sigma \to C$ be a given coloring. Since all elements of $L(G)$ are r-strings, we see that $L(c(G)) \subseteq C^r$ with cardinality at most k^r. We construct H first by constructing a *trie* T for $L = L(c(G))$, and then obtain a Σ-DAG H from T by identifying all leaves of T to form the sink.Clearly, we see that T has at most $||L|| \leqslant k^r$ edges since it is deterministic automata accepting L and L has total length $||L||$ at most k^r. Therefore, this shows the first claim. The remaining thing is the time and space complexity to construct T from G and c. For this purpose, we present Algorithm 4 that constructs a trie $T = (U, \texttt{goto}, \mathit{root})$ for $L(c(G))$ by simultaneously traversing T from the root to leaves and G from s to t in a interleaved manner. In what follows, we denote by $\mathrm{str}(P)$ the string label of a path in G and by $\mathrm{str}(v)$ the string label of the unique path from the root to a node v in T.

During the traversal of T as well as G, Algorithm 4 maintains binary relation $\varphi \subseteq V \times U$ for describing a correspondence between the end points of all prefixes of elements in $L(c(G))$ as strings labels in the DAG G and the end point of their unique locus in the trie T (see Fig. 2). Precisely, the binary relation φ satisfies the condition (i) and (ii) below, which can be easily shown by induction on the length of strings labels represented by G and T: (i) for any vertex x in DAG G and any path P from s to x, there exists some pair $(x, v) \in \varphi$ with a vertex v in T such that $\mathrm{str}(P) = \mathrm{str}(v)$; (ii) for any vertex v in T, there exists some pair $(x, v) \in \varphi$ of vertices and some path P from s to x in G such that $\mathrm{str}(P) = \mathrm{str}(v)$. As a consequence, it immediately follows that $L(T) = L(c(G))$. We remark that any string has the unique locus (the end point of the string label of a path) in a trie T if it is represented by T. Thus, Algorithm 4 correctly works. For the time complexity, by construction, it is not hard to see that Algorithm 4 runs on an input DAG G and a color set C with $|C| = k$ in the input-output sensitive manner using $O(\mathrm{size}(T)\deg(G)) = O(k^r \cdot \mathrm{size}(G))$ time and $O(\mathrm{size}(T) + \mathrm{size}(G))$ space. ◀

D Proofs for Section 6 (Hardness results)

Sec. 6.1 (Hardness of Diverse String Set for Unbounded K)

▶ **Theorem 6.1** (NP-hardness for unbounded K)**.** *When K is part of an input, Max-Min and Max-Sum Diverse String Set on Σ-graphs for r-strings are NP-hard even for any constant $r \geqslant 3$.*

Proof. We reduce an NP-hard problem 3DM [GJ79] to Max-Min Diverse String Set by a trivial reduction. Recall that given an instance consists of sets $A = B = C = [h]$ for some $n \geqslant 1$ and a set family $F \subseteq [n]^3$, and 3DM asks if there exists some subset $M \subseteq F$ that is a *matching*, that is, any two vectors $X, Y \in M$ have no position $i \in [3]$ at which the corresponding symbols agree, i.e., $X[i] = Y[i]$. Then, we construct an instance of Max-Min Diverse String Set with $r = 3$ with an alphabet $\Sigma = A \cup B \cup C$, a string set $L = F \subseteq \Sigma^3$, integers $K = n$ and $\Delta = r = 3$. Obviously, this transformation is polynomial time computable. Then, it is not hard to see that for any $M \subseteq F$, M is a matching if and only if $D^{\min}_{d_H}(M) \geqslant \Delta$ holds. On the other hand, for Max-Min Diverse String Set, if we let $\Delta' = \binom{K}{2}$ then for any $M \subseteq F$, M is a matching if and only if $D^{\mathrm{sum}}_{d_H}(M) \geqslant \Delta'$ holds. Combining the above arguments, the theorem is proved. ◀

Sec. 6.2 (Hardness of Diverse LCSs for Unbounded K)

▶ **Theorem 6.3.** *Under Hamming distance, Max-Min (resp. Max-Sum) Diverse String Set for $m \geqslant 2$ strings parameterized with K is fpt-reducible to Max-Min (resp. Max-Sum) Diverse LCSs for two string ($m = 2$) parameterized with K, where m is part of an input. Moreover, the reduction is also a polynomial time reduction from Max-Min (resp. Max-Sum) Diverse String Set to Max-Min (resp. Max-Sum) Diverse LCSs.*

From now on, we show the proof of Theorem 6.3. Let $L = \{X_1, \ldots, X_s\} \subseteq \Sigma^r$ and K be an instance of Max-Sin Diverse String Set, where $r \geqslant 1, s \geqslant 2$. We let $\Gamma = \Sigma \cup \{\, a_{i,j}, b_{i,j} \mid i, j \in [s] \,\}$ be a new alphabet. We define the set $\{P_1, \ldots, P_s, Q_1, \ldots, Q_s\}$ of $2s$ strings of length s over Γ such that

$$P_i := a_{i1} \ldots a_{is}, \quad Q_i := b_{i1} \ldots b_{is}, \qquad \forall i \in [s]. \tag{4}$$

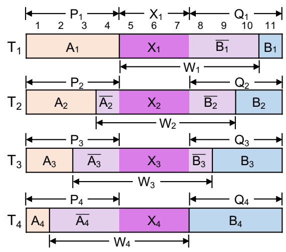


Figure 4 Construction of the set $\mathcal{T}$ of s r-strings

For all $i \in [s]$, $|X_i| = r$ and $|P_i| = |Q_i| = s$ hold. We let $\mathcal{T} = \{T_1, \ldots, T_s\}$ be the set of s strings of length $r + 2s$ over Γ such that

$$T_i := P_i X_i Q_i, \qquad \forall i \in [s]. \tag{5}$$

Now, we construct two strings $\mathcal{S}_1$ and $\mathcal{S}_2$ over Γ with intention that $LCS(\mathcal{S}_1, \mathcal{S}_2) = \mathcal{T}$. For all $i \in [s]$, we define three nonempty substrings $A_i, W_i, B_i \in \Gamma^+$ of T_i such that $A_i \cdot W_i \cdot B_i = T_i$ as follows (see Fig. 4). For all $i \in [s]$,

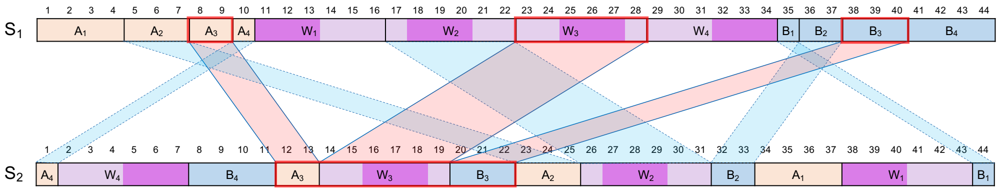


Figure 5 An example of two input strings (left) and alignment of four longest common subsequences constructed in the fpt-reduction from MAX-MIN DIVERSE STRING SET to MAX-MIN DIVERSE LCS for two string in the proof for Theorem 6.3. In the figure, a group of red parallelograms connecting segments S_1 and S_2 indicate a matching M for a longest common subsequence, while blue parallelograms indicate prohibited matchings crossing M.

- A_i is the prefix of P_i of length $s-i+1$ and
- B_i is the suffix of Q_i of length i, that is, $A_i := P_i[1..s-i+1]$ and $B_i := Q_i[s-i..s]$.
- W_i is the string $W_i := \overline{A_i} \cdot X_i \cdot \overline{B_i}$, where $\overline{A_i}$ is the suffix of P_i of length $i-1$ and $\overline{B_i}$ is the prefix of Q_i of length $s-i$.

By construction, we see that $|A_i| + |\overline{A_i}| = |B_i| + |\overline{B_i}| = s$ for all $i \in [s]$. This implies that $P_i = A_i \cdot \overline{A_i}$ and $Q_i = \overline{B_i} \cdot B_i$. Since $T_i = P_i \cdot X_i \cdot Q_i = A_i \cdot \overline{A_i} \cdot X_i \cdot \overline{B_i} \cdot B_i = A_i \cdot W_i \cdot B_i$, we have $T_i = A_i \cdot W_i \cdot B_i$ as intended.

Assuming the above definitions, we define two input strings $\mathcal{S}_1$ and $\mathcal{S}_2$ as follows:

$$\mathcal{S}_1 := (A_1 \cdots A_s) \cdot (W_1 \cdots W_s) \cdot (B_1 \cdots B_s) = \textstyle\prod_{i=1}^{s} A_i \cdot \prod_{i=1}^{s} W_i \cdot \prod_{i=1}^{s} B_i,$$

$$\mathcal{S}_2 := (A_s \cdot W_s \cdot B_s) \cdots (A_1 \cdot W_1 \cdot B_1) = \textstyle\prod_{i=s}^{1} (A_i \cdot W_i \cdot B_i). \quad (6)$$

For example, Fig. 5 illustrates construction of S_1 and S_2 for $s = 4$. Then, we can associate to two input strings S_1 and S_2 a bipartite graph $\mathcal{B} = (V_1 \cup V_2, E)$.

▶ **Definition D.1.** *$\mathcal{B}(S_1, S_2) = (V_1 \cup V_2, E)$ is the bipartite graph, where the vertex set is unions of the sets V_1 and V_2 of positions in S_1 and S_2, respectively, and the edge set $E \subseteq V_1 \times V_2$ is defined as for any pair of positions $(i_1, i_2) \in V_1 \times V_2$, $(i_1, i_2) \in E$ if and only if they are labeled with the same symbol, namely, $S_1[i_1] = S_2[i_2]$. Each pair $(i_1, i_2) \in E$ is called a match between S_1 and S_2.*

Any subset $M \subseteq E$ is called an *non-crossing matching* in $\mathcal{B}(S_1, S_2)$ if there exists no pair of distinct edges $(i_1, i_2), (j_1, j_2) \in M$ such that (i) they share an end in common, namely, $i_1 = j_1$ or $i_2 = j_2$ (matching), and (ii) they cross each other, namely, $(i_1 < j_1)$ and $(i_2 < j_2)$ (non-crossing). By definition, if M is an non-crossing matching with $|M| = \ell$, we can order the edges of M in the increasing order as $M = \{(i_1, j_1), \dots, (i_\ell, j_\ell)\}$ with $i_1 < \cdots < i_\ell$ and $j_1 < \cdots < j_\ell$. Then, we observe that the string $S_1(M) := S_1[i_1] \cdots S_1[i_\ell] \in \Sigma^\ell$ (equivalently, $S_2(M) := S_2[j_1] \cdots S_2[j_\ell]$) forms the common subsequence associated to M.

▶ **Lemma D.1.** *For any $M \subseteq V_1 \times V_2$, M is a (cardinality-) maximum non-crossing matching in $\mathcal{B}(S_1, S_2)$ if and only if the associated string $S_1(M) = S_2(M)$ is a longest common subsequence of S_1 and S_2.*

Consider three groups $\mathcal{A}, \mathcal{B}$, and $\mathcal{W}$ of all *segments* of the forms A_i's, B_i's, and W_i's, respectively. We remark that for any group $\mathcal{G}$, any segment G_i in $\mathcal{G}$ has the unique occurrences $Occ_1(G_i) = [p..p + |G| - 1]$ in S_1 and $Occ_2(G_i) = [q..q + |G| - 1]$ in S_2, respectively, for its starting positions p in S_1 and q in S_2. For any segment G, we say that a match $e = (i_1, i_2)$ *connects the occurrences of G* in both S_1 and S_2 if $i_1 \in Occ_1(G)$ and $i_2 \in Occ_2(G)$, or equivalently, $e \in Occ_1(G) \times Occ_2(G)$. Now, we show the next lemma.

▶ **Lemma D.2.** *S_1 and S_2 satisfies the following properties (see Fig. 5):*

(1) The orders of occurrences of all A_i's are reversed in two strings S_1 and S_2. The same holds for all B_i's.

(2) Considering S_2, A_i's appear in the increasing order of their lengths, while B_i's appear in the decreasing order of their lengths.

(3) For any non-crossing matching M between S_1 and S_2 and any group $\mathcal{G}$, there exists exactly one segment G in $\mathcal{G}$ whose occurrences in S_1 and S_2 are connected by a match in M.

(4) Moreover, if M has maximal $|M|$ w.r.t. set-inclusion $\subseteq$, all positions in $Occ_1(G)$ and $Occ_1(G)$ are connected, i.e., $|M \cap (Occ_1(G) \times Occ_2(G))| = |G|$ holds.

Proof. Claims (1) and (2) are obvious from the construction of A_i's, B_i's, and S_1 and S_2.

Claim (3): Consider Fig. 5. Suppose that M is non-crossing matching. Without loss of generality, we assume that $\mathcal{G} = \mathcal{A}$. Suppose that M contains a match $e \in M$ between the occurences of some segment $A_i \in \mathcal{A}$ with $i \in [s]$. For example, we assume in the figure that $A_i = A_3$ and e is either $(8, 12)$ or $(9, 13)$. In Fig. 5, e is contained in the red band connecting $Occ_1(A_3)$ and $Occ_2(A_3)$. Then, any $\mathcal{A}$-segments A_j other than A_i cannot be connected by some match $e' \in M$ since e and e' must cross each other if $i \neq j$. For example, if we take matches connecting either A_2 or A_4 (shown in blue bands in Fig. 5), this blue and red bands cross, and it contradicts the assumption that M is non-crossing. Since the similar discussion hold for $\mathcal{B}$ and $\mathcal{W}$, Claim (3) is proved.

Claim (4): Suppose that M is inclusion-wise maximal in addition to (3). Let A be any segment in $\mathcal{A}$ with $i \in [s]$. By construction of $\mathcal{A}$-segments, any match in $M \cap (Occ_1(A) \times Occ_2(A))$ must connect the symbols $S_1[pos_1 + k - 1]$ and $S_2[pos_2 + k - 1]$ for some $k \in [1..|A|]$, where $pos_h = Occ_h[1]$ for $h = 1, 2$ (*). Suppose to contradict that $|M \cap (Occ_1(A) \times Occ_2(A))| < |A|$. In this case, we observe that there exists at least one $k \in [|A|]$ such that $S_1[pos_1 + k - 1]$ and $S_2[pos_2 + k - 1]$ are not connected yet. We also any other match cross the band between $Occ_1(A)$ and $Occ_2(A)$ (see in Fig. 5 the red band between two occurrences of A_3 in S_1 and S_2). For the pair $e = (pos_1 + k - 1, pos_2 + k - 1)$, since $e \notin M$, we can add it into M still keeping that M is an non-crossing matching. This contradicts the assumption that M is inclusion-wise maximal non-crossing matching. Hence, we have $|M \cap (Occ_1(A) \times Occ_2(A))| = |A|$. Since the same discussion holds for $\mathcal{B}$. For a segment W in $\mathcal{W}$, although the claim (*) does not nessarily holds, we can still show that $|M \cap (Occ_1(W) \times Occ_2(W))| = |W|$ holds. Hence, Claim (4) is proved. This completes the lemma. ◀

▶ **Lemma D.3.** *$LCS(S_1, S_2) = \{ T_i \mid i \in [s] \}$, where $T_i = P_i X_i Q_i$ for all $i \in [s]$.*

Proof. (1) Let M be any cardinality non-crossing maximum (thus, inclusion-wise maximal) matching M in $\mathcal{B} = \mathcal{B}(S_1, S_2)$. From Claims (3) and (4) of Lemma D.2, we observe that M connects exactly one A-segment, B-segment, and W-segment from groups $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{W}$, respectively. Therefore, M must have the associated common subsequence $Y := A_i W_j B_k$ for some $i, j, k \in [s]$. In this case, if M has the maximum cardinality, the length $|Y| := |A_i| + |W_j| + |B_k|$ must be maximum. By property (2) of Lemma D.2, this must be possible only if $i = j = k$ holds (*). To see this, we fix the middle index j. Then, since $i > j$ and $k < j$ must hold by construction of S_2 (see Fig. 5), we have $|A_s| < \cdots < |A_i| < \cdots < |A_j|$ and $|B_j| > \cdots > |B_k| > \cdots > |B_1|$. Hence, claim (*) is proved. Consequently, any cardinality-maximal matching M contains matches connecting unique positions in $T_j = A_j W_j B_j$ for some $j \in [s]$ (**). Moreover, we observe that there is no match $e \in E \setminus M$ that can be added to M because the bands connecting segments A_j, W_j, and B_j (red bands in Fig. 5) prevent any other match in E. Therefore, M is inclusion-wise maximal non-crossing matching in $\mathcal{B}(S_1, S_2)$.

(2) On the other hand, Let $i \in [s]$ be any index. Then, we show that the string $T_i = A_i W_j B_k$ is actually a longest common subsequence in $LCS(S_1, S_2)$ as follows. We observe that T_i has the corresponding non-crossing matching M that connects the corresponding occurences of A_i, of B_i, and of W_i, respectively. For example, see Fig. 5 for $i = 3$, where the segments A_3, W_3, and B_3 occupy the intervals $S_1[8..9], S_1[23..28]$, and $S_1[28..40]$, while the string $A_3 \cdot W_3 \cdot B_3$ occupies the inteval $S_2[12..22]$. By combining the above discussions, we see that all of $T_1, \ldots, T_s$ are inclusion-wise maximal non-crossing matchings in $\mathcal{B}$. By construction, all of them have the same cardinality, namely, $|T_1| = \cdots = |T_s| = 2s + r$.

(3) Suppose to contradict that there exists a larger non-crossing matching M_* in $\mathcal{B}$ such that $|M_*| > 2s + r$. Suppose without loss of generality that M_* is inclusion-wise maximal. Then, it follows from Claim (**) of (1) that the corresponding common subsequence contains $T_j = A_j W_j B_j$ with $|T_j| = 2s + r$ as a subsequence for some $j \in [s]$. Let M_j be the non-cross matching associated to T_j. Since $|M_j| < |M_*|$, there exists a match $e \in M_* \setminus M_j \neq \emptyset$. By assumption, both of M_* and M_j are non-crossing. Therefore, we can add e to M_j to enlarge it. However, since M_j is inclusion-wise maximal by assumption, this contradicts. By contradiction, we conlude that there is not such non-crossing matching M_* such that $|M_*| > 2s + r$. Therefore, all of $T_1, \ldots, T_s$ are cardinality-maximum among all non-crossing machings in $\mathcal{B}$, and thus, are members of $LCS(S_1, S_2)$. Combining the above arguments, we conclude that $LCS(S_1, S_2) = \{ T_i \mid i \in [s] \}$. ◀

Using Lemma D.3, we finish the proof for Theorem 6.3.

Proof for Theorem 6.3. First, we show an fpt-reduction from Max-Min Diverse String Set parameterized with K to Max-Min Diverse LCS for two strings parameterized with K' as follows. Given $L = \{ X_i \mid i \in [K] \} \subseteq \Sigma^*$ and K, we let S_1 and S_2 be two strings constructed above, and let $\Delta' := \Delta + 2s$ and $K' = \kappa(K) := K$. By Lemma D.3, we see that $LCS(S_1, S_2) = \mathcal{T} = \{ T_i = P_i X_i Q_i \mid i \in [K] \} \subseteq \Gamma^*$ holds. Let $i \in [s]$ be any index. Since $|P_i| = |P_j| = s$, $|Q_i| = |Q_j| = s$ hold by construction, we can decompose the Hamming distance as $d_H(T_i, T_j) = d_H(P_i X_i Q_i, P_j X_j Q_j) = d_H(P_i, P_j) + d_H(X_i, X_j) + d_H(Q_i, Q_j)$. Since $d_H(P_i, P_j) = d_H(Q_i, Q_j) = s$ by construction, we have the next claim

Claim 1. $d_H(T_i, T_j) = d_H(X_i, X_j) + 2s$ for all $i, j \in [s]$. ▷

Let $I = \{i_1, \ldots, i_K\} \subseteq [K]$ be any subset of indices, $\mathcal{X} = \{X_{i_1}, \ldots, X_{i_K}\}$, and $\mathcal{Y} = \{T_{i_1}, \ldots, T_{i_K}\}$. From *Claim 1*, we have $D^{\min}_{d_H}(\mathcal{Y}) = \min_{j<k} d_H(T_{i_j}, T_{i_k}) = \min_{j<k}\{d_H(X_{i_j}, X_{i_k}) + 2s\} = 2s + \min_{j<k} d_H(X_{i_j}, X_{i_k}). = 2s + D^{\min}_{d_H}(\mathcal{X})$. Therefore, we have the next claim.

Claim 2. $D^{\min}_{d_H}(\mathcal{Y}) = 2s + D^{\min}_{d_H}(\mathcal{X})$. ▷

Then, we show that $D^{\min}_{d_H}(\mathcal{X}) \geqslant \Delta$ for some $\mathcal{X} \subseteq L$ with $|\mathcal{X}| = K$ if and only if $D^{\min}_{d_H}(\mathcal{Y}) \geqslant \Delta + 2s = \Delta'$ for some $\mathcal{Y} \subseteq LCS(S_1, S_2)$ with $|\mathcal{Y}| = K$ (*). We consider two directions.

- *Only-if direction*: We let $\mathcal{X} \subseteq L$ be any subset with $|\mathcal{X}| = K$. Without loss of generality, we assume that $\mathcal{X} = \{X_{i_1}, \ldots, X_{i_K}\} \subseteq L$ for some index set $I = \{i_1, \ldots, i_K\} \subseteq [K]$. From *Claim 2*, we have $D^{\min}_{d_H}(\mathcal{X}) \geqslant \Delta \iff D^{\min}_{d_H}(\mathcal{Y}) \geqslant 2s + \Delta = \Delta'$. Since $|\mathcal{Y}| = |\mathcal{X}|$, the only-if direction follows.

- *If direction*: We let $\mathcal{Y} \subseteq LCS(S_1, S_2)$ be any subset with $|\mathcal{Y}| = K$. From Lemma D.3, we assume without loss of generality that $\mathcal{Y} = \{T_{i_1}, \ldots, T_{i_K}\} \subseteq \mathcal{T}$ for some index set $I = \{i_1, \ldots, i_K\} \subseteq [K]$, and thus, we let $\mathcal{X} = \{X_{i_1}, \ldots, X_{i_K}\} \subseteq L$. By the same arguments to the only-if direction, we have $D^{\min}_{d_H}(\mathcal{X}) \geqslant \Delta$ iff $D^{\min}_{d_H}(\mathcal{Y}) \geqslant \Delta'$.

Since this reduction is correct (*), and it is polynomial time computable, it is a polynomial time reduction from Max-Min Diverse String Set to Max-Min Diverse LCS for two strings. Moreover, the parameter $\kappa(K) = K$ depends only on K, it is an fpt-reduction for the parameterize versions.

By modifying the previos construction, we present an fpt-reduction from Max-Sum Diverse String Set parameterized with K to Max-Sum Diverse LCS for two strings parameterized with K' as follows. Given an instance $L = \{\, X_i \mid i \in [K] \,\} \subseteq \Sigma^*$ and K, we let S_1, S_2, and $K' = \kappa(K) := K$ be the same to the previous reduction. In this case, we let $\Delta' := \Delta + \binom{K}{2}$.

Similarly, we let $\mathcal{X} = \{X_{i_1}, \ldots, X_{i_K}\}$, and $\mathcal{Y} = \{T_{i_1}, \ldots, T_{i_K}\}$ for any $I = \{i_1, \ldots, i_K\} \subseteq [K]$. From *Claim 1*, we have $D^{\mathrm{sum}}_{d_H}(\mathcal{Y}) = \sum_{j<k} d_H(T_{i_j}, T_{i_k}) = \sum_{j<k}\{d_H(X_{i_j}, X_{i_k}) + 2s\} = 2s\binom{K}{2} + \sum_{j<k} d_H(X_{i_j}, X_{i_k}). = 2s\binom{K}{2} + D^{\mathrm{sum}}_{d_H}(\mathcal{X})$. Therefore, we have the next claim.

Claim 2. $D^{\min}_{d_H}(\mathcal{Y}) = 2s\binom{K}{2} + D^{\min}_{d_H}(\mathcal{X})$. ▷

By similar discussion as above, we can show that $D^{\mathrm{sum}}_{d_H}(\mathcal{X}) \geqslant \Delta$ for some $\mathcal{X} \subseteq L$ with $|\mathcal{X}| = K$ if and only if $D^{\mathrm{sum}}_{d_H}(\mathcal{Y}) \geqslant \Delta + 2s\binom{K}{2} = \Delta'$ for some $\mathcal{Y} \subseteq LCS(S_1, S_2)$ with $|\mathcal{Y}| = K$ (**). Since this reduction is correct (**), and it is polynomial time computable, it is a polynomial time reduction from Max-Sum Diverse String Set to Max-Sum Diverse LCS for two strings. Moreover, the parameter $\kappa(K) = K$ depends only on K, it is an fpt-reduction for the parameterize versions. Combining the above arguments, the theorem is proved. ◀

▶ **Corollary 6.1** (NP-hardness). *When K is part of an input, Max-Min and Max-Sum Diverse LCSs for two r-strings are NP-hard, where r and Δ are part of an input.*

Proof. From Theorem 6.3, the claim immediately follows from Theorem 6.1. ◀

▶ **Corollary 6.2** (W1-hardness). *When parameterized with K, Max-Min and Max-Sum Diverse LCSs for two r-strings are W[1]-hard, where r and Δ are part of an input.*

Proof. From Theorem 6.3, the claim immediately follows from Theorem 6.2. ◀

References for the Appendix

GJ79 Michael R. Garey and David S. Johnson. Computers and intractability: A guide to the theory of np-completeness, 1979.

Gus97 D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997.