Approximating the Center Ranking Under Ulam

Diptarka Chakraborty
National University of Singapore, Singapore

Kshitij Gajjar
National University of Singapore, Singapore

Agastya Vibhuti Jha[1]
EPFL, Lausanne, Switzerland

Abstract

We study the problem of approximating a *center* under the *Ulam metric*. The Ulam metric, defined over a set of permutations over $[n]$, is the minimum number of move operations (deletion plus insertion) to transform one permutation into another. The Ulam metric is a simpler variant of the general edit distance metric. It provides a measure of dissimilarity over a set of rankings/permutations. In the center problem, given a set of permutations, we are asked to find a permutation (not necessarily from the input set) that minimizes the maximum distance to the input permutations. This problem is also referred to as *maximum rank aggregation under Ulam*. So far, we only know of a folklore 2-approximation algorithm for this NP-hard problem. Even for constantly many permutations, we do not know anything better than an exhaustive search over all $n!$ permutations.

In this paper, we achieve a $\left(\frac{3}{2} - \frac{1}{3m}\right)$-approximation of the Ulam center in time $n^{O(m^2 \ln m)}$, for m input permutations over $[n]$. We therefore get a polynomial time bound while achieving better than a 3/2-approximation for constantly many permutations. This problem is of special interest even for constantly many permutations because under certain dissimilarity measures over rankings, even for four permutations, the problem is NP-hard.

In proving our result, we establish a surprising connection between the approximate Ulam center problem and the *closest string with wildcards* problem (the center problem over the Hamming metric, allowing wildcards). We further study the closest string with wildcards problem and show that there cannot exist any $(2 - \epsilon)$-approximation algorithm (for any $\epsilon > 0$) for it unless P = NP. This inapproximability result is in sharp contrast with the same problem without wildcards, where we know of a PTAS.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Center Problem, Ulam Metric, Edit Distance, Closest String, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.12

Funding This work was supported by NUS ODPRT Grant, WBS No. R-252-000-A94-133.

1 Introduction

Finding a representative of a data set is a classical aggregation task heavily used in data analysis. Given a set S of points in a metric space, one of the more popular versions asks to find a point (not necessarily from S) that minimizes the maximum distance to the points in S, i.e.,

$$\min_{y} \max_{x \in S} d(y, x). \qquad (1)$$

Such a point is called a *center*. The question of finding a center in a metric space dates back to the nineteenth century [42]. In several applications, it suffices to compute an *approximate center*, i.e., a point in the metric space that approximates the objective value (1). The problem

[1] This work was done while the author was a student at IIIT-Delhi

of finding an (approximate) center has been studied widely both in theory and practice. Various metric spaces have been considered for the center problem, including Euclidean (both constant [33] and high dimension [7, 43]), Hamming [19, 29, 32, 30], Hamming with wildcard [22], the edit metric [36], Jaccard distance [10], rankings [6, 8, 38], etc. A similar task is to find a *median* point, which asks to minimize the *sum* of the distances to the data points (instead of the maximum distance). The median problem has also been studied extensively in various metric spaces [15, 40, 18, 1, 24, 41, 13, 34]. Despite being similar, finding a center is a much harder task than finding a median. For instance, in the Hamming metric, finding a median is folklore (just take a coordinate-wise majority), whereas finding a center is NP-hard [19].

In this paper, we primarily focus on approximating the center over the *Ulam metric*, which is a close variant of the edit metric. The Ulam metric of dimension n is the metric space $(\mathcal{S}_n, d)$, where $\mathcal{S}_n$ is the set of all permutations over $[n]$ and $d(x, y)$ is the minimum number of character moves needed to transform x into y [2].[2] The importance of studying the Ulam metric is twofold. First, it is an interesting measure of dissimilarity between rankings. The problem of finding a consensus ranking on a set of alternatives based on the preferences of voters arise in many application domains like sports, databases, elections, search engines, and statistics. A common ranking that best captures the preferences among the alternatives is often characterized by the center objective function (1) (a.k.a. *maximum rank aggregation*). The second aspect of the Ulam metric is that it captures some of the inherent difficulties of the edit metric. Thus, any progress in the Ulam metric may provide insights to tackle the more general edit metric which finds numerous applications in computational biology [21, 37], DNA storage system [20, 39], speech recognition [26], and classification [31]. The Ulam metric has also been studied from different algorithmic perspectives [17, 12, 3, 4, 35, 9].

There is a folklore algorithm that finds a 2-approximate center by simply reporting the best input permutation that minimizes the objective (1). This 2-approximation, in fact, holds for every metric space. Unfortunately, so far, we do not know any polynomial-time algorithm that attains better than the folklore 2-approximation, even when the number of input permutations is constant. For the exact computation (or even to beat the 2-factor), nothing better than the exhaustive search (over $n!$ permutations) is known, even for constantly many input permutations. On the hardness side, we only know that it is NP-complete [6]. On the contrary, for the Ulam median problem, very recently, [11] broke below the 2-factor in polynomial time. [11] also provided a polynomial-time 3/2-approximation algorithm for constantly many input permutations. It is not difficult to show that the Ulam center is at least as hard as the Ulam median, even for constantly many inputs (see Appendix A).

Our main result is a deterministic polynomial-time algorithm that breaks below the 3/2-approximation for the Ulam center problem for constantly many inputs.

▶ **Theorem 1.** *There is a deterministic algorithm that, given as input a set of m permutations $S \subseteq \mathcal{S}_n$, computes a $\left(\frac{3}{2} - \frac{1}{3m}\right)$-approximate center of S in time $n^{O(m^2 \ln m)}$.*

The above running time could be improved by increasing the approximation factor slightly. More specifically, for every $\epsilon > 0$, we can compute a $\left(\frac{3}{2} + \epsilon - \frac{1}{m}\right)$-approximate center in time $n^{3m} + n^{O\left(\frac{\ln m}{\epsilon^2}\right)}$ (see Remark 10). It is straightforward to see that when m is constant, the algorithm in the above theorem runs in polynomial time and computes a better than 3/2-approximate center. The question of approximating a center for constantly many ranks/permutations is particularly interesting because even for four inputs, it is known to be NP-complete with respect to *Kendall's tau distance* [18, 8], another often used dissimilarity measure for rankings [23, 44, 45].

[2] One may also consider one deletion and one insertion operation instead of a character move, and define the distance accordingly [17].

Nevertheless, we provide a polynomial-time algorithm to solve the (exact) Ulam center problem for three permutations (Theorem 12). It is worth noting that for Kendall's tau distance, it is unknown whether the center problem is in P or NP-complete for three permutations.

We show our result (Theorem 1) by establishing a surprising connection between the Ulam center and (a generalization of) the *closest string with wildcards* problem. We will explain this connection in the technical overview. In this paper, we further study the closest string with wildcards problem. In the *closest string* problem, given a set of n-length strings over some fixed alphabet Σ, the objective is to find a center (a string from Σ^n) under the Hamming distance. This problem is NP-complete [19], but a PTAS is known [29].

A variant of the closest string problem is the closest string with wildcards. In this variant, each input string may include any number of a wildcard character $*$. The wildcard character $*$ can be matched with all the characters of Σ. For two strings $s, s' \in (\Sigma \cup \{*\})^n$, the Hamming distance between them is defined as $d_H(s, s') := |\{i \in [n] \mid s[i] \neq s'[i] \text{ and } s[i] \neq *, s'[i] \neq *\}|$. Given a set of strings with wildcards, we are asked to find a center string (with no wildcard character) of length n with respect to the Hamming distance. (Note, if wildcards are allowed in the center string, the all-wildcard string will trivially become a center.) This problem is also NP-complete [22]. However, no better than a 2-factor (polynomial-time) algorithm is known. The parameterized complexity of this problem has also been considered [22, 25]. The Hamming distance with wildcard has been studied widely (e.g. [16, 28, 14]) due to its numerous applications in computational biology, large scale web searching, database systems.

As we mentioned earlier, for the simpler variant without any wildcard, there is a PTAS. Can we get a similar PTAS when wildcards are allowed? In this paper, we refute such a possibility. We show that attaining much better than a 2-approximation factor for the closest string with wildcards problem (even for a binary alphabet) is not possible unless P = NP.

▶ **Theorem 2.** *There is no deterministic polynomial-time $(2 - \epsilon)$-approximation algorithm (for any $\epsilon > 0$) for the closest string with wildcards problem, unless* P = NP.

The above hardness result holds even for a binary alphabet. The above theorem is in sharp contrast with the (typical) closest string problem for which a PTAS is known [29]. To the best of our knowledge, this is the first $(2 - \epsilon)$-factor inapproximability result for any center problem defined over a set of strings.

1.1 Technical overview

Approximating the Ulam center. One of our main contributions is a polynomial-time (better than) 3/2-factor approximation algorithm for the Ulam center problem for constantly many permutations. Our algorithm runs in $n^{O(m^2 \ln m)}$ time for m permutations, and achieves $\left(\frac{3}{2} - \frac{1}{3m}\right)$ approximation. For simplicity in exposition, we briefly describe our algorithm that achieves 3/2-approximation by assuming m is a constant. At the very high level, we first compute an exact n-length center (not necessarily a permutation) using dynamic programming and then convert that into a permutation that incurs approximation. The idea is similar to what was used for the Ulam median problem in [11]. However, the similarity ends here. The transformation algorithm that converts an n-length center into a permutation is more intricate, and the analysis is more involved. Another interesting aspect of our algorithm is that we establish a surprising connection between the approximate Ulam center and a generalization of the closest string problem. Let us now briefly explain our algorithm.

If the optimal center objective OPT is small (bounded by a constant), we can find a center permutation by performing an exhaustive search up to a small distance from any input. Thus as long as OPT is at most some constant, in polynomial time, we find an exact center permutation. So, from now, assume that OPT is at least some constant. Our main algorithm has two main steps. The first step constitutes a dynamic programming algorithm that returns an n-length string which maximizes the minimum LCS (Longest Common Substring) with the input permutations (see Subsection 4.1). This algorithm is essentially a generalization of [40, 27]. Let x_n^* be the string we get from this step. Note, the metric defined by $n - |\mathsf{LCS}|$ is essentially the Ulam distance over permutations. So our dynamic programming provides us a center string that minimizes the maximum $n - |\mathsf{LCS}|$. Let us denote this optimum value as OPT_n. Clearly, n-length center string is a relaxation of center permutation. Thus, $\mathsf{OPT}_n \leq \mathsf{OPT}$.

If x_n^* is a permutation, we are done, as we have found an optimal center permutation. Otherwise, we modify x_n^* to get a permutation. Let there be ℓ symbols $a_1, a_2, \dots, a_\ell$ that appear more than once in x_n^*. For any a_j, each occurrence might be part of a LCS with a subset of input permutations. Now, suppose we delete any one of the occurrences arbitrarily. In that case, we will increase the distances to the corresponding subset of inputs. Consequently, we may end up with a string far from a particular input permutation, causing a much worse objective value than OPT. Thus, we need to delete them in a "balanced" way such that distances to all the inputs increase in "a uniform manner". For that purpose, we introduce a generalization of the closest string problem, which we call *matrix bi-coloring*. We create a matrix having ℓ columns, each corresponding to a repeated symbol. Each row of the matrix corresponds to an input permutation. Thus the number of rows is equal to the number of inputs. Then for a column (corresponding to a_j), we color the entries as follows: If a_j of an input permutation s_i is aligned (with respect to some fixed optimal alignment) with the c-th occurrence of a_j in x_n^*, we color the corresponding entry ((i,j)-th entry) of the matrix by c. Essentially, for each symbol a_j, we have a set of color classes. Each color class c denotes the subset of inputs whose a_j aligns with c-th occurrence of x_n^*. There could be some uncolored entries as well. (See Figure 2 for an example.) Then, we select exactly one color per symbol/column (denoting which occurrence to keep in x_n^*) and cover (alternatively, mark as $\mathfrak{red}$) all the "un-matched" colored entries of that column. Next, we come up with a "coloring scheme" (i.e., a choice of colors per column) such that after covering (marking as $\mathfrak{red}$) the un-matched colored entries, the maximum covered ($\mathfrak{red}$) entries per row is minimized. In general (for an arbitrary colored matrix), there may not exist a coloring scheme leading to a bounded number of covered ($\mathfrak{red}$) entries per row. (This could happen for "tall" matrices, with significantly more rows than columns.) Fortunately, that is not the case for us. Since we have constantly many input permutations and the number of repeated symbols is large (follows from our large OPT assumption), there will always exist a "good" coloring scheme. If we keep the occurrences of repeated symbols in x_n^* according to an optimal coloring scheme, we end up not increasing the center objective value (maximum distances) by much. It is possible to find an optimal coloring scheme using another dynamic programming algorithm (Appendix B). Once we delete all the repeated occurrences, we insert the missing symbols into x_n^*, again in a balanced manner. In the end, we are left with a permutation z over $\mathcal{S}_n$.

The main point of removing repeated entries and the insertion of missing symbols in a balanced manner is to keep the distances to each input within a 3/2-factor of the initial distance. We argue that, in the end, the distance between an input permutation and the final permutation z will be at most 3/2 times the initial (maximum) distance to x_n^*. Hence, the center objective value of the output z is at most $\frac{3}{2}\mathsf{OPT}_n \leq \frac{3}{2}\mathsf{OPT}$. We refer the reader to Section 4 for the detailed analysis.

$$
\begin{aligned}
s_1 &= (\ 8\ 9\ 10\ \boxed{2}\ \boxed{5}\ \boxed{3}\ 11\ 12\ 13\ \boxed{1}\ \boxed{4}\ \boxed{7}\ \boxed{6}\ 14\ 15\ 16\ )\\
s_2 &= (\ 7\ 11\ 12\ 13\ \boxed{6}\ \boxed{4}\ \boxed{1}\ 8\ 9\ 10\ \boxed{3}\ \boxed{5}\ \boxed{2}\ 14\ 15\ 16\ )\\
s_3 &= (\ 14\ 15\ 16\ \boxed{5}\ \boxed{6}\ \boxed{4}\ 11\ 12\ 13\ \boxed{1}\ \boxed{2}\ \boxed{3}\ \boxed{7}\ 8\ 9\ 10\ )\\
s_4 &= (\ 11\ 12\ 13\ \boxed{3}\ \boxed{2}\ \boxed{1}\ 14\ 15\ 16\ \boxed{4}\ \boxed{5}\ \boxed{7}\ \boxed{6}\ 8\ 9\ 10\ )\\
s_5 &= (\ 7\ 8\ 9\ 10\ \boxed{3}\ \boxed{6}\ \boxed{2}\ 14\ 15\ 16\ \boxed{4}\ \boxed{1}\ \boxed{5}\ 11\ 12\ 13\ )\\
s_6 &= (\ 14\ 15\ 16\ \boxed{5}\ \boxed{1}\ \boxed{4}\ 8\ 9\ 10\ \boxed{2}\ \boxed{6}\ \boxed{3}\ \boxed{7}\ 11\ 12\ 13\ )
\end{aligned}
$$

$$
x = (\ \boxed{2}\ \boxed{5}\ \boxed{3}\ \boxed{6}\ \boxed{2}\ \boxed{4}\ \boxed{1}\ \boxed{4}\ \boxed{3}\ \boxed{5}\ \boxed{2}\ \boxed{1}\ \boxed{7}\ \boxed{6}\ \boxed{3}\ \boxed{7}\ )
$$

Figure 1 Let $n = 16$ and $m = 6$. Each of $s_1, s_2, \dots, s_6$ is a permutation in $\mathcal{S}_{16}$. x is a string (not a permutation) of length 16 over the alphabet $[16]$. Different occurrences of the same symbol are colored differently in x. The colored entries of s_i also denote an *alignment* with x (in this example, $\mathsf{LCS}(x, s_i)$). Note that $\min_{i\in[6]} |\mathsf{LCS}(x, s_i)| = 6$. Figure 2 shows the colored matrix for this example.

$$
\begin{array}{c|ccccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
s_1 & 1 & 1 & 1 & 2 & 1 & 2 & 1 \\
s_2 & 1 & 3 & 2 & 1 & 2 & 1 & \times \\
s_3 & 1 & 3 & 3 & 1 & 1 & 1 & 2 \\
s_4 & 1 & 2 & 1 & 2 & 2 & 2 & 1 \\
s_5 & 1 & 2 & 1 & 1 & 2 & 1 & \times \\
s_6 & 1 & 3 & 3 & 3 & 1 & 2 & 2 \\
f_j & 1 & 3 & 3 & 3 & 2 & 2 & 2
\end{array}
\qquad
\begin{array}{c|ccccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
s_1 & 1 & \text{red} & 1 & 2 & \text{red} & \text{red} & \text{red} \\
s_2 & 1 & 3 & \text{red} & \text{red} & 2 & 1 & \times \\
s_3 & 1 & 3 & \text{red} & \text{red} & \text{red} & 1 & 2 \\
s_4 & 1 & \text{red} & 1 & 2 & 2 & \text{red} & \text{red} \\
s_5 & 1 & \text{red} & 1 & \text{red} & 2 & 1 & \times \\
s_6 & 1 & 3 & \text{red} & \text{red} & \text{red} & \text{red} & 2 \\
f_j & 1 & 3 & 3 & 3 & 2 & 2 & 2
\end{array}
$$

Figure 2 (Left) A colored matrix M, corresponding to the example in Figure 1 (here, the number in each cell simply denotes the color of the cell). (Right) A matrix bi-coloring scheme $\mathcal{A}$ of M, where $\mathcal{A} = (1, 3, 1, 2, 2, 1, 2)$. As the maximum number of $\mathfrak{red}$ entries in a row is 4, we have $\mathsf{MRI}(\mathcal{A}(M)) = 4$.

Inapproximability of the closest string with wildcards. The matrix bi-coloring problem mentioned earlier is a generalization of the closest string with wildcards problem for a fixed alphabet Σ. To see this, restrict the number of colors per column for the matrix bi-coloring to the alphabet size. (In the matrix bi-coloring, the number of colors per column could be as large as the number of rows, and also, the number of colors in two different columns could be different.) In a simpler version where wildcards are not allowed, we know of a PTAS [29]. So it is quite natural to ask whether we can get a better than 2-factor approximation (ideally, a PTAS) for the closest string with wildcards problem in polynomial time. In this paper, we refute the possibility of having one, assuming $\mathsf{P} \neq \mathsf{NP}$. We show that there is no polynomial-time $(2 - \epsilon)$-approximation algorithm for this problem.

To show our result, we provide a reduction from a variant of the satisfiability (SAT) problem, namely $(1, k, 2k+1)$-SAT introduced by Austrin, Guruswami and Håstad [5]. In this problem, for any fixed integer $k \geq 1$, given a $(2k+1)$-CNF formula F, the objective is to distinguish whether there is a satisfying assignment that satisfies at least k literals per clause or F is unsatisfiable. For every fixed integer $k \geq 1$, this problem was shown to be NP-hard [5].

We provide a simple polynomial-time reduction from $(1,k,2k+1)$-SAT to the problem of approximating closest string with wildcards. For an $0<\epsilon<1$, fix $k=\lceil 1/\epsilon \rceil$. Given an instance ($(2k+1)$-CNF formula) F of $(1,k,2k+1)$-SAT with n variables and m clauses, for each clause we create an n-length binary string with wildcards ($\{0,1\}\cup\{*\}$). Each bit position of these strings corresponds to a variable. In a clause, if a variable appears as a positive literal, we set the corresponding bit position of the corresponding string to be 1; and if it appears as a negative literal, we set the corresponding bit position of the corresponding string to be 0. If a variable does not appear in a particular clause, we set the corresponding bit position to be a wildcard ($*$).

Thus in the reduced instance, each string contains at most $2k+1$ non-wildcard entries. It is quite straightforward to see that for a YES instance of the $(1,k,2k+1)$-SAT formula, there is a satisfying assignment that leads to a center string with objective value at most $k+1$. On the other hand, any center string with an objective value $<2k+1$ gives a satisfying assignment. (See Section 5 for the details.)

2 Preliminaries

Notations. Let $[n]$ denote the set $\{1,2,\dots,n\}$. We refer to the set of all permutations over $[n]$ by $\mathcal{S}_n$. Throughout this paper we consider any permutation x as a sequence of numbers $(a_1,a_2,\dots,a_n)$ such that $x(i)=a_i$.

The Ulam metric and the problem of finding a center. Given two permutations $x,y\in\mathcal{S}_n$, the *Ulam distance* between them, denoted by $d(x,y)$, is the minimum number of character move operations[3] that is needed to transform x into y. Alternatively, it can be defined as $n-|\mathsf{LCS}(x,y)|$, where $\mathsf{LCS}(x,y)$ denotes a *longest common subsequence* between x and y.

Given two strings (permutations) x and y of lengths n_x and n_y respectively, an *alignment* g is a function from $[n_x]$ to $[n_y]\cup\{\perp\}$ which satisfies:

- $\forall i\in[n_x]$, if $g(i)\neq\perp$, then $x(i)=y(g(i))$;
- Let $i\in[n_x], j\in[n_x]$ such that $i\neq j$, $g(i)\neq\perp$ and $g(j)\neq\perp$. Then $i<j \Leftrightarrow g(i)<g(j)$.

For an alignment g between two strings (permutations) x and y, we say g *aligns* a character $x(i)$ with some character $y(j)$ if and only if $j=g(i)$. Thus the alignment g is essentially a common subsequence between x and y (see Figure 1 for an example).

Given a set $S\subseteq\mathcal{S}_n$ and another permutation $y\in\mathcal{S}_n$, we refer to the quantity $\max_{x\in S} d(y,x)$ by the *center objective value* of S with respect to y, denoted by $\mathtt{Obj}(S,y)$.

Given a set $S\subseteq\mathcal{S}_n$, a *center* of S is a permutation $x_{\text{cen}}\in\mathcal{S}_n$ (not necessarily from S) such that $\mathtt{Obj}(S,x_{\text{cen}})$ is minimized, i.e., $x_{\text{cen}}=\arg\min_{y\in\mathcal{S}_n}\mathtt{Obj}(S,y)$. We denote $\mathtt{Obj}(S,x_{\text{cen}})$ by $\mathsf{OPT}(S)$. We call a permutation $\tilde{x}$ a *c-approximate center* (for some $c>0$) of S if and only if $\mathtt{Obj}(S,\tilde{x})\leq\mathsf{OPT}(S)\leq c\cdot\mathtt{Obj}(S,\tilde{x})$.

3 Matrix Bi-coloring

In this section, we introduce a problem called the *matrix bi-coloring* problem. Let us start by defining a colored matrix. We use positive integers to identify a color (except a special color $\mathfrak{red}$). An $m\times\ell$ dimensional matrix M is said to be *colored* if each entry of each column

[3] A single move operation in a permutation can be thought of as "picking up" a character from its position and then "inserting" that character in a different position.

$j \in [\ell]$ is either assigned a color from the set $[f_j]$ (for some positive integer f_j), or $\mathfrak{no\text{-}color}$ (which is to say it is uncolored). (Multiple entries in the same column may have the same color.) In other words, the number of *distinct* colors (other than $\mathfrak{no\text{-}color}$) that can be seen in column j is f_j. See Figure 2 for a visual depiction of matrix bi-coloring.

Given a colored matrix, our goal is to pick exactly one color c_j for each column $j \in [\ell]$ and recolor the matrix. All entries in column j which are colored c_j retain their color. We recolor the remaining entries (except the $\mathfrak{no\text{-}color}$ entries) of column j to $\mathfrak{red}$. This is called a *matrix bi-coloring scheme.* We now define this formally.

▶ **Definition 3** (Matrix Bi-coloring Scheme)**.** *Given an $m \times \ell$ colored matrix M, a* matrix bi-coloring scheme *$\mathcal{A}$ for M is an ℓ-tuple $(c_1, c_2, \ldots, c_\ell) \in [f_1] \times [f_2] \times \cdots \times [f_\ell]$.*

The ℓ-tuple $(c_1, c_2, \ldots, c_\ell)$ produced by the matrix bi-coloring scheme $\mathcal{A}$ is used to recolor the matrix M to produce a final $m \times \ell$ colored matrix $\mathcal{A}(M)$, computed as follows. For every $(i, j) \in [m] \times [\ell]$,

- if $M[i][j] = \mathfrak{no\text{-}color}$, then $\mathcal{A}(M)[i][j] = \mathfrak{no\text{-}color}$;
- else, if $M[i][j] = c_j$, then $\mathcal{A}(M)[i][j] = c_j$;
- else, $\mathcal{A}(M)[i][j] = \mathfrak{red}$.

For each row $i \in [m]$ of $\mathcal{A}(M)$, the *red index* of the row, denoted as $\mathsf{RI}_{\mathcal{A},M}(i)$, is the number of $\mathfrak{red}$ entries in the i-th row of $\mathcal{A}(M)$. (We will drop the subscript $\mathcal{A}, M$ when they are clear from the context.) The *maximum red index* of $\mathcal{A}(M)$ is defined as

$$\mathsf{MRI}(\mathcal{A}(M)) := \max_{i \in [m]} \mathsf{RI}(i).$$

Next, consider the following optimization problem.

▶ **Definition 4** (Matrix Bi-coloring Problem)**.** *Given an $m \times \ell$ colored matrix M, find a matrix bi-coloring scheme $\mathcal{A}$ of M with the minimum* MRI*. This minimum* MRI *is called the* bi-coloring number *of the matrix, denoted by* BCN*. Formally,*

$$\mathsf{BCN}(M) = \min_{\substack{\mathcal{A}:\ \mathcal{A} \text{ is a matrix bi-}\\ \text{coloring scheme for } M}} \mathsf{MRI}(\mathcal{A}(M)).$$

Thus, designing a matrix bi-coloring scheme essentially means coming up with a color for each column. We would like to emphasize that the above problem is a generalization of a certain variant of the center problem under the Hamming metric known as *closest string with wildcards* [22] (see Section 5).

We first show an upper bound on $\mathsf{BCN}(M)$. Then we provide a dynamic programming algorithm that, given a colored matrix M, finds $\mathsf{BCN}(M)$ exactly.

3.1 An upper bound on the bi-coloring number

We show that for every colored matrix M, there always exists a bi-coloring scheme $\mathcal{A}$ such that $\mathsf{MRI}(\mathcal{A}(M))$ is not "too large".

▶ **Theorem 5.** *Let M be an $m \times \ell$ colored matrix such that $m^4 \leq e^\mu$, where*

$$\mu = \sum_{j \in [\ell]} \left(1 - \frac{1}{f_j}\right).$$

Recall that f_j is the number of distinct colors in column $j \in [\ell]$ (not counting $\mathfrak{no\text{-}color}$). Then

$$\mathsf{BCN}(M) \leq \mu + 2\sqrt{\mu \ln m}.$$

An interesting fact about the above theorem is that μ depends only on the *number* of different colors in each column, regardless of how those colors are placed in M. We prove this theorem using the probabilistic method.

Proof of Theorem 5. Given a colored matrix M, we randomly pick an ℓ-tuple $(c_1, c_2, \ldots, c_\ell)$, by selecting each c_j independently uniformly at random from $[f_j]$. This leads to a random bi-coloring scheme $\mathcal{A}$. We then show that the expected MRI of $\mathcal{A}(M)$ is at most μ. Then by a simple Chernoff bound, we conclude that there exists a choice of the ℓ-tuple for which the MRI is at most $\mu + 2\sqrt{\mu \ln m}$, proving Theorem 5.

More precisely, for each column $j \in [\ell]$, we pick a color c_j independently uniformly at random from $[f_j]$. So,

$$\Pr_{c_j \sim [f_j]} [c_j = c] = \frac{1}{f_j} \qquad \forall\, j \in [\ell], c \in [f_j]. \tag{2}$$

Recall, in each column $j \in [\ell]$ of $\mathcal{A}(M)$, each entry $\mathcal{A}(M)[i][j]$ is either $\mathfrak{no\text{-}color}$, c_j or $\mathfrak{red}$. For all $i \in [m]$ and $j \in [\ell]$, let $X_{i,j}$ be an indicator random variable denoting whether $\mathcal{A}(M)[i][j] = \mathfrak{red}$ or not, i.e.,

$$X_{i,j} = \begin{cases} 1 & \text{if } \mathcal{A}(M)[i][j] = \mathfrak{red}; \\ 0 & \text{otherwise.} \end{cases}$$

Note, $X_{i,j} = 1$ if and only if $M[i][j] \in [f_j]$ and $c_j \neq M[i][j]$. For all $i \in [m]$, let

$$X_i = \sum_{j \in [\ell]} X_{i,j}.$$

Thus, the random variable X_i denotes the number of $\mathfrak{red}$-entries in row $i \in [m]$. The expected number of $\mathfrak{red}$-entries in row $i \in [m]$ is given by

$$\begin{aligned}
\mathbb{E}[X_i] &= \mathbb{E}\left[\sum_{j \in [\ell]} X_{i,j}\right] = \sum_{j \in [\ell]} \mathbb{E}[X_{i,j}] && \text{(Linearity of expectation)} \\
&= \sum_{j \in [\ell]} \Pr[X_{i,j} = 1] \\
&= \sum_{\substack{j \in [\ell]: \\ M[i][j] \in [f_j]}} \Pr[X_{i,j} = 1] + \sum_{\substack{j \in [\ell]: \\ M[i][j] = \mathfrak{no\text{-}color}}} \Pr[X_{i,j} = 1] \\
&= \sum_{\substack{j \in [\ell]: \\ M[i][j] \in [f_j]}} \Pr[X_{i,j} = 1] + 0 && \left(\begin{smallmatrix}\mathcal{A}(M)[i][j] = \mathfrak{no\text{-}color} \\ \iff M[i][j] = \mathfrak{no\text{-}color}\end{smallmatrix}\right) \\
&= \sum_{\substack{j \in [\ell]: \\ M[i][j] \in [f_j]}} \Pr\left[c_j \neq M[i][j]\right] && \text{(By definition)} \\
&= \sum_{\substack{j \in [\ell]: \\ M[i][j] \in [f_j]}} \left(1 - \frac{1}{f_j}\right) && \text{(By Equation 2)} \\
&\leq \sum_{j \in [\ell]} \left(1 - \frac{1}{f_j}\right) = \mu.
\end{aligned}$$

Thus for every row $i \in [m]$, the expected number of $\mathfrak{red}$-entries in row i is at most μ. We will now show that the event that all of the rows simultaneously have at most $\mu + 2\sqrt{\mu \ln m}$ many $\mathfrak{red}$-entries occurs with non-zero probability.

Note that for each fixed row $i \in [m]$, the indicator random variables $X_{i,1}, X_{i,2}, \ldots, X_{i,\ell}$ are independent. We set

$$\delta = 2\sqrt{(\ln m)/\mu}.$$

Since we are given that $m^4 \leq e^{\mu}$, taking ln on both sides and square rooting, we get $2\sqrt{(\ln m)/\mu} \leq 1$. Thus $0 < \delta \leq 1$. Then it follows from a standard application Chernoff bound that

$$\Pr[X_i \geq (1+\delta) \cdot \mu] \leq \exp\left(\frac{-\delta^2 \cdot \mu}{3}\right).$$

By a union bound,

$$\Pr[\exists\ i \in [m] : X_i \geq (1+\delta) \cdot \mu] \leq m \cdot \exp\left(\frac{-\delta^2 \cdot \mu}{3}\right).$$

Thus we get,

$$\begin{aligned}
\Pr[\forall\ i \in [m] : X_i < (1+\delta) \cdot \mu] &\geq 1 - \left(m \cdot \exp\left(\frac{-\delta^2 \cdot \mu}{3}\right)\right) \\
&= 1 - \left(m \cdot \exp\left(\frac{-\left(2\sqrt{(\ln m)/\mu}\right)^2 \cdot \mu}{3}\right)\right) \qquad \text{(Substituting } \delta\text{)} \\
&= 1 - \left(m \cdot \exp\left(\frac{-4 \ln m}{3}\right)\right) \\
&= 1 - m^{-1/3} > 0.
\end{aligned}$$

Thus, $\Pr[\forall\ i \in [m] : X_i < (1+\delta) \cdot \mu] > 0$. In other words, there is a non-zero probability that the number of $\mathfrak{red}$-entries in every row $i \in [m]$ is at most $(1+\delta) \cdot \mu$. Therefore, there exists a bi-coloring scheme $\mathcal{A}^*$ such that the red index $\mathsf{RI}(i)$ of every row $i \in [m]$ satisfies $\mathsf{RI}(i) \leq (1+\delta) \cdot \mu$. Hence,

$$\mathsf{MRI}(\mathcal{A}^*(M)) = \max_{i \in [m]} \mathsf{RI}(i) \leq (1+\delta) \cdot \mu = \mu + \delta\mu = \mu + 2\sqrt{\mu \ln m}.$$

Since $\mathsf{BCN}(M) \leq \mathsf{MRI}(\mathcal{A}^*(M))$, this completes the proof. ◀

We can also compute an optimal bi-coloring using a dynamic programming algorithm. We defer the algorithm to Appendix B.

▶ **Theorem 6.** *There is a deterministic algorithm* FindBiColoring *that, given an $m \times \ell$ colored matrix M, finds a matrix bi-coloring scheme $\mathcal{A}$ of M with the minimum* MRI*, in $O(m\ell^{m+1})$ time.*

4 Approximation Algorithm for the Ulam Center

In this section, we provide a 3/2-approximation algorithm for the Ulam center problem. In particular, we prove Theorem 1.

We are given a set of permutations $S = \{s_1, s_2, \cdots, s_m\} \subseteq \mathcal{S}_n$ as input. Our algorithm runs two procedures, each producing a permutation (candidate center), and returns the better of the two (that has smaller value). For any positive integer k and a permutation $s \in \mathcal{S}_n$, let us use the notation $\mathcal{B}_k(s)$ to denote the set of all the permutations at distance at most k from s, i.e.,

$$\mathcal{B}_k(s) := \{x \in \mathcal{S}_n \mid d(s,x) \leq k\}.$$

The first procedure BoundedSearch performs an exhaustive search up to distance k, for $k = 8m^2 \ln m$. More specifically, it considers an input permutation, say s_1, and enumerates over all $x \in \mathcal{B}_k(s_1)$, and finally returns a permutation $x \in \mathcal{B}_k(s_1)$ that minimizes $\max_{s_i \in S} d(x, s_i)$. Note, $|\mathcal{B}_k(s_1)| = O(n^{2k})$. Thus the running time of this procedure is $O(mn^{2k+1} \ln n)$. (Computing the Ulam distance between two permutations in $\mathcal{S}_n$ takes $O(n \ln n)$ time.) Clearly, if the optimum center objective $\mathsf{OPT}(S) \leq k$, the procedure BoundedSearch outputs an optimum center. So from now, we assume

$$\mathsf{OPT}(S) \geq 8m^2 \ln m. \tag{3}$$

The second procedure, referred to as ApproxCenter, has two main steps. Firstly, it computes the best center string (not necessarily a permutation) x_n^* of length at most n using a procedure FindStringCenter. And secondly, it converts x_n^* to a permutation $s^* \in \mathcal{S}_n$ using a procedure StringToPermutation.

We will show that assuming (3), s^* output by ApproxCenter is a 3/2-approximate center of S. Below we first describe each of the two main steps of ApproxCenter in detail.

4.1 Finding a length-restricted center string

This subsection provides a dynamic programming algorithm that given any set of n length strings computes a center string of length n. More specifically, we design an algorithm that computes a string (over the alphabet $[n]$) of length n, which maximizes the minimum longest common subsequence (LCS) with the input strings. We defer the algorithm to Appendix C.

▶ **Theorem 7.** *There is a deterministic algorithm* FindStringCenter *that, given m strings $s_1, s_2, \ldots, s_m$, each of length n, computes a string $x_n^* = \arg\max_{x \in [n]^n} (\min_i |\mathsf{LCS}(x, s_i)|)$, also of length n, in $O(n^{2m+1} 2^m)$ time.*

Now let us apply this algorithm to our problem. Recall that we are given a set of permutations $S = \{s_1, s_2, \ldots, s_m\} \subseteq \mathcal{S}_n$. We apply the procedure FindStringCenter on the input set S to get an n-length string x_n^*. Note that x_n^* need not be a permutation. In the next subsection, we describe how to transform x_n^* into a permutation.

4.2 Converting a length-restricted center string to a permutation

Let x_n^* be the n-length string obtained in the previous subsection. If x_n^* is a permutation, then we are done, as we have an *exact* solution to the Ulam center problem. Otherwise, let $\mathcal{R} = \{a_1, a_2, \ldots, a_\ell\}$ be the set of "repeated symbols", i.e., the symbols that appear at least twice in x_n^*. For each $a_j \in \mathcal{R}$, let freq_j denote the number of occurrences of a_j in x_n^*. Also, let $\mathcal{M}$ be the set of "missing symbols", i.e., the symbols that do not appear in x_n^*. To transform x_n^* into a permutation, we need to remove duplicate occurrences of the repeated symbols ($\mathcal{R}$), and insert all the missing symbols ($\mathcal{M}$).

Our transformation procedure StringToPermutation consists of following two steps (the pseudocodes for these can be found in Appendix D):

1. Use a procedure RemoveDuplicate (Algorithm 1) to remove all the duplicate occurrences of the symbols in $\mathcal{R}$. RemoveDuplicate first computes an (arbitrary) optimal alignment α_i between x_n^* and s_i, for each $i \in [m]$. Next, construct a colored matrix M of dimension $m \times \ell$ as follows: For each $i \in [m]$ and $j \in [\ell]$, if α_i aligns the r-th occurrence (for some $r \in [\mathsf{freq}_j]$) of the symbol a_j in x_n^*, set $M[i][j] = r$; else set $M[i][j] =$ no-color. (As s_i is a permutation, at most one occurrence of a_j in x_n^* can be aligned with the a_j in s_i.)
Then we use this colored matrix M as an instance of the matrix bi-coloring problem (defined in Section 3) and find an optimum bi-coloring scheme $\mathcal{A}$ for M (using Theorem 6). (For an illustration, see Figure 1 and Figure 2.) Let the scheme $\mathcal{A}$ be the tuple $(c_1, \dots, c_\ell)$. Then, for each symbol $a_j \in \mathcal{R}$, we keep the c_j-th occurrence of it in x_n^* and delete all the remaining occurrences of it. Let $\bar{x}$ denote the output string. (Note, no symbol in $\bar{x}$ appears more than once, and therefore the length of $\bar{x}$ might be less than n.)
2. Use a procedure InsertMissing (Algorithm 2) to insert all the symbols in $\mathcal{M}$ in $\bar{x}$ in a "balanced" manner. Compute an (arbitrary) optimal alignment β_i between $\bar{x}$ and s_i, for each $i \in [m]$. (Note, β_i's can easily be obtained by updating the α_i's computed before.) Consider s_1 and a symbol $b \in \mathcal{M}$. Suppose $s_1[p] = b$, for $p \in [n]$. Let $q \in [n]$ be the largest index $< p$ such that $s_1[q] = a$ is aligned by β_1.
Then place b just after a in $\bar{x}$, and also update β_1 (by aligning the symbol b). Then remove b from the set $\mathcal{M}$. Next, consider s_2 and another symbol from $\mathcal{M}$, and insert that symbol in $\bar{x}$ in a similar way. Loop through the input permutations one by one in a cyclic manner (after s_m, again take s_1) and perform the above process of inserting symbols in $\mathcal{M}$, until there is no symbol left in $\mathcal{M}$. Let us denote the final transformed string $\bar{x}$ by z.

It is straightforward to see that the final output string z is a permutation in $\mathcal{S}_n$. We claim that z is a 3/2-approximate center. Recall, we only need to argue for $\mathsf{OPT}(S) \geq 8m^2 \ln m$ (by Assumption 3).

▶ **Lemma 8.** *Assuming 3, the final string z output by the procedure* StringToPermutation *is a $\left(\frac{3}{2} - \frac{1}{3m}\right)$-approximate center of S.*

Before commencing the proof of Lemma 8, we need a simple observation on the size of $\mathcal{M}$, the set of missing symbols. Since x_n^* is of length n, the number of missing symbols is equal to the number of repeated occurrences of the symbols in $\mathcal{R}$. More specifically,

$$|\mathcal{M}| = \sum_{a_j \in \mathcal{R}} (\mathsf{freq}_j - 1). \tag{4}$$

Let $\mu := \sum_{j=1}^{\ell}(1 - 1/\mathsf{freq}_j)$. Note that

$$\begin{aligned} \mu &= \sum_{j=1}^{\ell} \big((\mathsf{freq}_j - 1)/\mathsf{freq}_j\big) \leq \frac{1}{2}\sum_j (\mathsf{freq}_j - 1) && \text{(Since } \min_j \mathsf{freq}_j \geq 2) \\ &\leq |\mathcal{M}|/2. && \text{(By Equation 4)} \qquad (5) \end{aligned}$$

▷ Claim 9. If $\mu \geq 4 \ln m$, then for all $i \in [m]$,

$$|\mathsf{LCS}(\bar{x}, s_i)| \geq |\mathsf{LCS}(x_n^*, s_i)| - \frac{|\mathcal{M}|}{2} - \sqrt{2|\mathcal{M}| \ln m}.$$

Proof. Recall, $\mathcal{A}$ is an optimum matrix bi-coloring scheme for M (constructed by the procedure RemoveDuplicate). $\bar{x}$ is obtained from x_n^* by removing all repeated occurrences of the symbols in $\mathcal{R}$ according to the tuple $(c_1, \dots, c_\ell)$, corresponding to $\mathcal{A}$. Note, $\mu \geq 4 \ln m$ implies $m^4 \leq e^\mu$. By Theorem 5, $\mathsf{MRI}(\mathcal{A}(M)) \leq \mu + 2\sqrt{\mu \ln m}$, where $\mu = \sum_{j=1}^{\ell}(1 - 1/\mathsf{freq}_j)$.

Consider any $s_i \in S$. Note, M was constructed using the alignment α_i between s_i and x^*_n. Observe, the number of symbols (initially) aligned by α_i that are deleted from x^*_n to obtain $\bar{x}$ is at most $\mathsf{RI}(i)$ (see Section 3 for the definition of RI). Thus, we get

$$\begin{aligned} |\mathsf{LCS}(\bar{x}, s_i)| &\geq |\mathsf{LCS}(x^*_n, s_i)| - (\mu + 2\sqrt{\mu \ln m}) \\ &\geq |\mathsf{LCS}(x^*_n, s_i)| - \frac{|\mathcal{M}|}{2} - \sqrt{2|\mathcal{M}| \ln m}. && \text{(By Equation 5)} \end{aligned}$$

This completes the proof of Claim 9. ◁

Proof of Lemma 8. We will argue that for all $s_i \in S$, $d(z, s_i) \leq \left(\frac{3}{2} - \frac{1}{3m}\right) \mathsf{OPT}(S)$. Let s^* be an optimum center of S under the Ulam metric. So, for all $s_i \in S$, $d(s^*, s_i) \leq \mathsf{OPT}(S)$. Recall, by definition, $d(s^*, s_i) = n - |\mathsf{LCS}(s^*, s_i)|$. Thus,

$$\forall s_i \in S, \quad |\mathsf{LCS}(s^*, s_i)| \geq n - \mathsf{OPT}(S). \tag{6}$$

Since x^*_n maximizes the minimum LCS between an n-length string and $s_i \in S$, by (6),

$$\forall s_i \in S, \quad |\mathsf{LCS}(x^*_n, s_i)| \geq n - \mathsf{OPT}(S). \tag{7}$$

Also, observe that

$$\begin{aligned} |\mathcal{M}| &\leq n - \min_{s_i \in S} (|\mathsf{LCS}(x^*_n, s_i)|) \\ &\leq \mathsf{OPT}(S). && (\min_{s_i \in S} (|\mathsf{LCS}(x^*_n, s_i)|) \geq n - \mathsf{OPT}(S) \text{ by Equation 7}) \end{aligned} \tag{8}$$

Consider an $s_i \in S$. By the procedure InsertMissing, among the inserted symbols at least $\lfloor |\mathcal{M}|/m \rfloor$ symbols will be aligned between the final string z and s_i by the alignment function β_i. In particular,

$$|\mathsf{LCS}(z, s_i)| \geq |\mathsf{LCS}(\bar{x}, s_i)| + \left\lfloor \frac{|\mathcal{M}|}{m} \right\rfloor. \tag{9}$$

Next, we proceed by considering the two cases depending on the value of μ separately. Let us first argue for $\mu < 4 \ln m$. In fact, in this case, we get a solution that is much closer to the optimum. More specifically, we claim that $d(z, s_i) \leq (1 + 1/m^2)\mathsf{OPT}(S)$. As $\bar{x}$ is obtained from x^*_n by deleting repeated occurrences of the symbols in $\mathcal{R}$ and $s_i \in \mathcal{S}_n$,

$$|\mathsf{LCS}(\bar{x}, s_i)| \geq |\mathsf{LCS}(x^*_n, s_i)| - \ell. \qquad \text{(Recall, } \ell = |\mathcal{R}|) \tag{10}$$

Note, the above inequality holds irrespective of the value of μ.

Since $\mathsf{freq}_j \geq 2$ for all $j \in [\ell]$, we have $\mu = \sum_{j=1}^{\ell} (1 - 1/\mathsf{freq}_j) \geq \ell/2$. This implies that

$$\ell \leq 2\mu < 8 \ln m \leq \frac{\mathsf{OPT}(S)}{m^2} \tag{11}$$

where the last inequality follows from Assumption (3). Thus,

$$\begin{aligned} d(z, s_i) &= n - |\mathsf{LCS}(z, s_i)| && \text{(By definition)} \\ &\leq n - |\mathsf{LCS}(\bar{x}, s_i)| && \text{(By Equation 9)} \\ &\leq n - |\mathsf{LCS}(x^*_n, s_i)| + \ell && \text{(By Equation 10)} \\ &\leq \mathsf{OPT}(S) + \frac{\mathsf{OPT}(S)}{m^2} && \text{(By Equation 7, 11)} \\ &\leq \left(1 + \frac{1}{m^2}\right) \mathsf{OPT}(S). \end{aligned}$$

So, for $\mu < 4\ln m$, we have $d(z,s_i) \le \left(1+\frac{1}{m^2}\right)\mathsf{OPT}(S)$, which is at most $\left(\frac{3}{2}-\frac{1}{3m}\right)\mathsf{OPT}(S)$ for $m \ge 2$.

Now, the only remaining case is $\mu \ge 4\ln m$. We will use an argument similar to the previous case. The only difference is that now to lower bound $|\mathsf{LCS}(\bar{x},s_i)|$, we will apply Claim 9 (instead of Equation 10).

$$\begin{aligned}
|\mathsf{LCS}(z,s_i)| &\ge |\mathsf{LCS}(\bar{x},s_i)| + \left\lfloor \frac{|\mathcal{M}|}{m} \right\rfloor && \text{(By Equation 9)}\\
&\ge |\mathsf{LCS}(x_n^*,s_i)| - \frac{|\mathcal{M}|}{2} - \sqrt{2|\mathcal{M}|\ln m} + \left\lfloor \frac{|\mathcal{M}|}{m} \right\rfloor && \text{(By Claim 9)}\\
&\ge |\mathsf{LCS}(x_n^*,s_i)| - |\mathcal{M}|\left(\frac{1}{2}-\frac{1}{m}\right) - \sqrt{2|\mathcal{M}|\ln m} - 1. && (12)
\end{aligned}$$

Hence,

$$\begin{aligned}
d(z,s_i) &= n - |\mathsf{LCS}(z,s_i)| && \text{(By definition)}\\
&\le n - |\mathsf{LCS}(x_n^*,s_i)| + |\mathcal{M}|\left(\frac{1}{2}-\frac{1}{m}\right) + \sqrt{2|\mathcal{M}|\ln m} + 1 && \text{(By Equation 12)}\\
&\le \mathsf{OPT}(S) + \left(\frac{1}{2}-\frac{1}{m}+\sqrt{\frac{2\ln m}{\mathsf{OPT}(S)}}\right)\mathsf{OPT}(S) + 1 && \text{(By Equation 7, 8)}\\
&\le \left(\frac{3}{2}-\frac{1}{m}+\sqrt{\frac{2\ln m}{8m^2\ln m}}+\frac{1}{\mathsf{OPT}(S)}\right)\mathsf{OPT}(S) && \text{(By Assumption (3))}\\
&\le \left(\frac{3}{2}-\frac{1}{3m}\right)\mathsf{OPT}(S). && \text{(By Assumption (3))}
\end{aligned}$$

This completes the proof of Lemma 8. ◀

Running time analysis. We now analyze the running time of the overall algorithm. The procedure BoundedSearch takes $O(mn^{2k+1}\ln n)$ time (to perform an exhaustive search up to distance k from an input permutation). Next, we analyze the running time of ApproxCenter. The first step of it uses Theorem 7 taking $O(2^m n^{2m+1})$ time. The second step consists of two procedures: RemoveDuplicate and InsertMissing. Constructing the colored matrix M in the procedure RemoveDuplicate takes $O(mn\ln n)$ time. (Note, it only involves m LCS computations. In each of them, one of the strings is a permutation and thus takes $O(n\ln n)$ time per string by using a standard LCS algorithm.) Next, RemoveDuplicate invokes the algorithm from Theorem 6 to find an optimal bi-coloring scheme, which requires $O(mn^{m+1})$ time (the number of missing symbols is at most n). Once we get the bi-coloring scheme, generating $\bar{x}$ takes only $O(n)$ time. The next procedure, InsertMissing, again involves m LCS computations and then updating those alignments according to the insertion of missing symbols. This takes $O(mn\ln n)$ time. So the overall running time is $O(mn^{2k+1}\ln n + 2^m n^{2m+1} + mn^{m+1}) = n^{O(m^2\ln m)}$ (by replacing $k = 8m^2\ln m$).

▶ Remark 10. Let us now comment on how to reduce the running time by increasing the approximation factor slightly. Recall, after performing the exhaustive search BoundedSearch up to distance k, we remain with the case when $\mathsf{OPT}(S) \ge k$. When we analyze the approximation factor of our algorithm (in particular, Lemma 8), to attain $\left(\frac{3}{2}-\frac{1}{3m}\right)$ we need to assume that $\mathsf{OPT}(S)$ is at least $\Omega(m^2\ln m)$. That is why we set $k = 8m^2\ln m$. Our analysis essentially shows that the approximation factor is $\max\left\{1+\frac{8\ln m}{k}, \frac{3}{2}-\frac{1}{m}+\sqrt{\frac{2\ln m}{k}}+\frac{1}{k}\right\}$ with the running time $O(mn^{2k+1}\ln n + 2^m n^{2m+1})$. So we get a trade-off between the approximation factor and the running time. For instance, if we set $k = 8m$, we get an approximation factor $\left(\frac{3}{2} + \sqrt{\frac{\ln m}{4m}} - \frac{7}{8m}\right)$ and running time $n^{O(m)}$. In fact, for any $0 < \epsilon < 1$, by setting $k = \frac{8\ln m}{\epsilon^2}$, we get a $\left(\frac{3}{2} + \epsilon - \frac{1}{m}\right)$-approximate center in $O(2^m n^{2m+1}) + n^{O\left(\frac{\ln m}{\epsilon^2}\right)}$ time.

▶ **Theorem 11.** *There is a deterministic algorithm that, given an $0 < \epsilon < 1$ and a set of m permutations $S \subseteq \mathcal{S}_n$, computes a $\left(\frac{3}{2} + \epsilon - \frac{1}{m}\right)$-approximate center of S in time $n^{3m} + n^{O\left(\frac{\ln m}{\epsilon^2}\right)}$.*

4.3 An exact algorithm for three permutations

When the number of input permutations is only three (i.e., $m = 3$), we can get an exact polynomial-time algorithm for the Ulam center problem.

▶ **Theorem 12.** *There is a deterministic polynomial-time algorithm that takes 3 permutations as input, and outputs their Ulam center.*

We will show that given three permutations s_1, s_2, s_3 from $\mathcal{S}_n$, it is possible to remove duplicate symbols and insert missing symbols in a simple and efficient way that converts an n-length center string for s_1, s_2, s_3 to an optimal center permutation for them.

If the x_n^* computed by our dynamic program is already a permutation, then we are done. Otherwise, first fix some optimal alignment between x_n^* and s_i for all $i \in [3]$. We incrementally update the sting x_n^* by processing the repeated symbols one by one. Take an arbitrary repeated symbol a in x_n^* and a missing symbol b. Note, a appears more than once. Consider the first occurrence of a in x_n^*, and do the following:

- If (that particular occurrence of) a does not align with any of s_1, s_2, s_3, then we delete it from x_n^*, and insert b at an arbitrary location in x_n^*. Remove b from $\mathcal{M}$. Clearly, this does not decrease $\mathsf{LCS}(x_n^*, s_i)$ for any of the three s_i's.
- If (that particular occurrence of) a aligns with the symbol a of only one input permutation (say s_1), then delete it from x_n^*, and insert b in x_n^* so that it aligns with the b in s_3. Remove b from $\mathcal{M}$. The length of $\mathsf{LCS}(x_n^*, s_1)$ is decreased by one for by the deletion of a, but it is then increased by one by the insertion of b. So $|\mathsf{LCS}(x_n^*, s_1)|$ remains the same.
- If (that particular occurrence of) a aligns with the symbol a in two of the permutations (say s_1, s_2), we do nothing.

Only except the last case, each time we process a repeated symbol a, we remove an occurrence of it from x_n^*. (Note that the number of times we need to process a symbol a is equal to its number of occurrences in x_n^*, minus one.) Since the number of input permutations is exactly three, at most one occurrence of a can be aligned with that of two input permutations. So for any repeated symbol in x_n^*, at most once we will be in the last case.

Once we are left with no repeated symbols, we stop. (Since x_n^* is always of length n, there will not be any missing symbols left after we are done with processing all the repeated symbols.) So, x_n^* will eventually become a permutation, and $|\mathsf{LCS}(x_n^*, s_i)|$ for none of the three s_i's will decrease. Let us denote the final string by z. By Equation 7, for each $i \in [3]$,

$$n - \mathsf{OPT}(S) \leq |\mathsf{LCS}(x_n^*, s_i)| = |\mathsf{LCS}(z, s_i)|.$$

Thus, $d(z, s_i) \leq \mathsf{OPT}(S)$. Hence, we conclude that z is an (exact) Ulam center for s_1, s_2, s_3. The running time of the algorithm is clearly polynomial in n, concluding Theorem 12.

5 Closest String with Wildcards

The matrix bi-coloring problem defined in Section 3 is a generalization of the well-known *closest string with wildcards* problem. In this problem, any given string may include wildcard characters which can be matched with any character of the other strings. Consider any alphabet Σ. For any two strings $s, s' \in (\Sigma \cup \{*\})^n$, the Hamming distance between them is defined as

$$d_H(s, s') := |\{i \in [n] \mid s[i] \neq s'[i] \text{ and } s[i] \neq * \text{ and } s'[i] \neq *\}|.$$

In the closest string with wildcards problem, given a set of m strings $s_1, s_2 \ldots, s_m \in (\Sigma \cup \{*\})^n$, the objective is to find a string $s \in \Sigma^n$ such that $\max_{i\in[m]} d_H(s, s_i)$ is minimized.

The above problem is a special case of the matrix bi-coloring problem, where the strings are the rows of the matrix and the wildcards $(*)$ are the $\mathfrak{no\text{-}color}$ entries in the matrix. If we restrict $f_j = |\Sigma|$ (for all $j \in [n]$) in the matrix bi-coloring problem, we get the closest string with wildcards problem.

So far we do not know of any polynomial-time algorithm for the closest string with wildcards problem that achieves a $(2-\epsilon)$-factor approximation (for some $0 < \epsilon < 1$). In this section, we refute the possibility of getting such an algorithm unless $\mathsf{P} = \mathsf{NP}$, even when the alphabet Σ is binary. In particular, we prove Theorem 2.

To show the inapproximability result, we start with defining a variant of the satisfiability (SAT) problem, namely $(1, k, 2k+1)$-SAT introduced by Austrin, Guruswami & Håstad [5].

▶ **Definition 13** ($(1, k, 2k+1)$-SAT). *Let $k \geq 1$ be a fixed integer constant. Given a $(2k+1)$-CNF formula F (i.e., each clause of F has exactly $2k+1$ literals), decide between the following two cases:*

- *YES: There is an assignment for the variables in F that satisfies at least k literals in each clause of F.*
- *NO: F is unsatisfiable.*

▶ **Theorem 14** ([5]). *For every fixed integer $k \geq 1$, $(1, k, 2k+1)$-SAT is NP-hard.*

For $k = 1$, $(1, k, 2k+1)$-SAT is simply 3-SAT. We now provide a polynomial-time reduction from $(1, k, 2k+1)$-SAT to (a gap version of) the closest string with wildcards problem.

▶ **Definition 15** (Approximate closest string with wildcards). *Consider any alphabet Σ and an $\epsilon > 0$. Given a set of m strings $s_1, s_2, \ldots, s_m \in (\Sigma \cup \{*\})^n$ (where $*$ is a wildcard) and a positive integer r, decide between the following two cases.*

- *YES: There is a string $s \in \Sigma^n$ such that for all $i \in [m]$, $d_H(s, s_i) \leq r$.*
- *NO: For all strings $s \in \Sigma^n$, there exists an $i \in [m]$ such that $d_H(s, s_i) > (2-\epsilon)r$.*

Proof of Theorem 2. Let $k = \lceil 1/\epsilon \rceil$. Consider an instance ($(2k+1)$-CNF formula) F of the $(1, k, 2k+1)$-SAT problem with n variables $x_1, x_2, \ldots, x_n$ and m clauses. We create m strings each of length n over the alphabet $\{0, 1\} \cup \{*\}$. For each clause C_i, we create a string s_i as follows: If the literal x_j appears in C_i, set $s_i[j] = 1$; else if the literal $\bar{x}_j$ (negation of x_j) appears in C_i, set $s_i[j] = 0$; else set $s_i[j] = *$. Set $r = k+1$.

Suppose F is a YES instance of $(1, k, 2k+1)$-SAT. Take the corresponding satisfying assignment σ (that satisfies at least k literals per clause). Create a string $s \in \{0, 1\}^n$ by setting $s[j] = 1$ if x_j is set to TRUE by σ, and $s[j] = 0$ if x_j is set to FALSE by σ, for all $j \in [n]$. Note that $d_H(s, s_i) \leq (2k+1) - k = k+1$ for all $i \in [m]$.

Now, suppose F is a NO instance of $(1, k, 2k+1)$-SAT. Assume to contrary that there exists a string $s \in \{0,1\}^n$ such that for all $i \in [m]$, $d_H(s, s_i) \leq (2-\epsilon)(k+1) < 2k+1$ (since $k \geq 1/\epsilon$). Then create an assignment σ' by setting x_j to TRUE if $s[j] = 1$, and FALSE if $s[j] = 0$, for all $j \in [n]$. Note that σ' satisfies F, contradicting the fact that F is unsatisfiable.

The proof follows from Theorem 14. ◀

6 Conclusion

In this paper, we study the problem of computing a center rank/permutation under the Ulam metric, which is known to be NP-complete. There is a folklore 2-approximation algorithm that works for every metric space. No better (polynomial-time) algorithm is known for the Ulam metric, even when the number of input permutations is constant. Our main result breaks below the 3/2-approximation for constantly many inputs. An exciting open direction is to beat the 2-approximation for arbitrarily many inputs (i.e., an algorithm whose running time is polynomial in both n and m).

In proving our result, we establish a connection between the Ulam center problem and the closest string with wildcards problem (the center problem under the Hamming metric in the presence of wildcards). We further show that the latter problem is $(2-\epsilon)$-inapproximable unless P = NP. This result is in sharp contrast with the PTAS known for the closest string problem without wildcards.

References

1 Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: Ranking and clustering. *J. ACM*, 55(5):23:1–23:27, 2008. doi:10.1145/1411509.1411513.

2 David Aldous and Persi Diaconis. Longest increasing subsequences: from patience sorting to the Baik-Deift-Johansson theorem. *Bulletin of the American Mathematical Society*, 36(4):413–432, 1999. doi:10.1090/S0273-0979-99-00796-X.

3 Alexandr Andoni and Robert Krauthgamer. The computational hardness of estimating edit distance. *SIAM J. Comput.*, 39(6):2398–2429, 2010.

4 Alexandr Andoni and Huy L. Nguyen. Near-optimal sublinear time algorithms for Ulam distance. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 76–86, 2010. doi:10.1137/1.9781611973075.8.

5 Per Austrin, Venkatesan Guruswami, and Johan Håstad. (2+ϵ)-sat is NP-hard. *SIAM J. Comput.*, 46(5):1554–1573, 2017.

6 Christian Bachmaier, Franz J. Brandenburg, Andreas Gleißner, and Andreas Hofmeier. On the hardness of maximum rank aggregation problems. *Journal of Discrete Algorithms*, 31:2–13, 2015. 24th International Workshop on Combinatorial Algorithms (IWOCA 2013).

7 Mihai Bădoiu, Sariel Har-Peled, and Piotr Indyk. Approximate clustering via core-sets. In *Proceedings of the thiry-fourth annual ACM Symposium on Theory of Computing*, pages 250–257, 2002.

8 Therese Biedl, Franz J Brandenburg, and Xiaotie Deng. On the complexity of crossings in permutations. *Discrete Mathematics*, 309(7):1813–1823, 2009.

9 Mahdi Boroujeni and Saeed Seddighin. Improved MPC algorithms for edit distance and Ulam distance. In *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019.*, pages 31–40, 2019.

10 Marc Bury and Chris Schwiegelshohn. On finding the jaccard center. In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

11 Diptarka Chakraborty, Debarati Das, and Robert Krauthgamer. Approximating the median under the ulam metric. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 761–775. SIAM, 2021.

12 Moses Charikar and Robert Krauthgamer. Embedding the Ulam metric into l_1. *Theory of Computing*, 2(11):207–224, 2006. doi:10.4086/toc.2006.v002a011.

13 Flavio Chierichetti, Ravi Kumar, Sandeep Pandey, and Sergei Vassilvitskii. Finding the Jaccard median. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 293–311. SIAM, 2010. doi:10.1137/1.9781611973075.25.

14 Raphaël Clifford, Klim Efremenko, Benny Porat, and Ely Porat. A black box for online approximate pattern matching. In *Annual Symposium on Combinatorial Pattern Matching*, pages 143–151. Springer, 2008.

15 Michael B Cohen, Yin Tat Lee, Gary Miller, Jakub Pachocki, and Aaron Sidford. Geometric median in nearly linear time. In *Proceedings of the forty-eighth annual ACM Symposium on Theory of Computing*, pages 9–21, 2016.

16 Richard Cole and Ramesh Hariharan. Verifying candidate matches in sparse and wildcard matching. In *Proceedings of the thiry-fourth annual ACM Symposium on Theory of Computing*, pages 592–601, 2002.

17 Graham Cormode, Shan Muthukrishnan, and Süleyman Cenk Sahinalp. Permutation editing and matching via embeddings. In *International Colloquium on Automata, Languages, and Programming*, pages 481–492. Springer, 2001. doi:10.1007/3-540-48224-5_40.

18 Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the web. In *Proceedings of the Tenth International World Wide Web Conference, WWW 10*, pages 613–622, 2001. doi:10.1145/371920.372165.

19 Moti Frances and Ami Litman. On covering problems of codes. *Theory of Computing Systems*, 30(2):113–119, 1997.

20 Nick Goldman, Paul Bertone, Siyuan Chen, Christophe Dessimoz, Emily M. LeProust, Botond Sipos, and Ewan Birney. Towards practical, high-capacity, low-maintenance information storage in synthesized DNA. *Nature*, 494(7435):77–80, 2013.

21 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997.

22 Danny Hermelin and Liat Rozenberg. Parameterized complexity analysis for the closest string with wildcards problem. In *Combinatorial Pattern Matching*, pages 140–149, Cham, 2014. Springer International Publishing.

23 John G Kemeny. Mathematics without numbers. *Daedalus*, 88(4):577–591, 1959.

24 Claire Kenyon-Mathieu and Warren Schudy. How to rank with few errors. In *Proceedings of the thirty-ninth annual ACM Symposium on Theory of Computing*, pages 95–103, 2007.

25 Tomohiro Koana, Vincent Froese, and Rolf Niedermeier. Parameterized algorithms for matrix completion with radius constraints. In *31st Annual Symposium on Combinatorial Pattern Matching (CPM 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

26 Teuvo Kohonen. Median strings. *Pattern Recognition Letters*, 3(5):309–313, 1985. doi:10.1016/0167-8655(85)90061-3.

27 Joseph B Kruskal. An overview of sequence comparison: Time warps, string edits, and macromolecules. *SIAM review*, 25(2):201–237, 1983. doi:10.1137/1025045.

28 Christina Leslie, Rui Kuang, and Kristin Bennett. Fast string kernels using inexact matching for protein sequences. *Journal of Machine Learning Research*, 5(9), 2004.

29 Ming Li, Bin Ma, and Lusheng Wang. On the closest string and substring problems. *Journal of the ACM (JACM)*, 49(2):157–171, March 2002.

30 Bin Ma and Xiaoming Sun. More efficient algorithms for closest string and substring problems. *SIAM Journal on Computing*, 39(4):1432–1443, 2010.

31 Carlos D. Martínez-Hinarejos, Alfons Juan, and Francisco Casacuberta. Use of median string for classification. In *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, volume 2, pages 903–906. IEEE, 2000. doi:10.1109/ICPR.2000.906220.

32 Daniel Marx. Closest substring problems with small distances. *SIAM J. Comput.*, 38:1382–1410, 2008.

33 Nimrod Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of the ACM (JACM)*, 31(1):114–127, 1984.

34 Stanislav Minsker. Geometric median and robust estimation in banach spaces. *Bernoulli*, 21(4):2308–2335, 2015.

35 Timothy Naumovitz, Michael Saks, and C. Seshadhri. Accurate and nearly optimal sublinear approximations to Ulam distance. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2012–2031, 2017. `doi:10.1137/1.9781611974782.131`.

36 François Nicolas and Eric Rivals. Complexities of the centre and median string problems. In *Combinatorial Pattern Matching, 14th Annual Symposium, CPM 2003, Morelia, Michocán, Mexico, June 25-27, 2003, Proceedings*, pages 315–327, 2003.

37 Pavel Pevzner. *Computational molecular biology: an algorithmic approach.* MIT press, 2000.

38 V Yu Popov. Multiple genome rearrangement by swaps and by element duplications. *Theoretical computer science*, 385(1-3):115–126, 2007.

39 Cyrus Rashtchian, Konstantin Makarychev, Miklós Z. Rácz, Siena Ang, Djordje Jevdjic, Sergey Yekhanin, Luis Ceze, and Karin Strauss. Clustering billions of reads for DNA data storage. In *Advances in Neural Information Processing Systems 30*, pages 3360–3371. Curran Associates, Inc., 2017.

40 David Sankoff. Minimal mutation trees of sequences. *SIAM Journal on Applied Mathematics*, 28(1):35–42, 1975. `doi:10.1137/0128004`.

41 Warren Schudy. Approximation schemes for inferring rankings and clusterings from pairwise data. *Ph.D. Thesis*, 2012.

42 James Joseph Sylvester. A question in the geometry of situation. *Quarterly Journal of Pure and Applied Mathematics*, 1(1):79–80, 1857.

43 E Alper Yildirim. Two algorithms for the minimum enclosing ball problem. *SIAM Journal on Optimization*, 19(3):1368–1391, 2008.

44 H Peyton Young. Condorcet's theory of voting. *American Political science review*, 82(4):1231–1244, 1988.

45 H Peyton Young and Arthur Levenglick. A consistent extension of condorcet's election principle. *SIAM Journal on Applied Mathematics*, 35(2):285–300, 1978.

A Ulam Median reduces to Ulam Center

The idea of the reduction is the same as [8, Theorem 6]. Here we discuss the idea for four permutations, which can be generalised to m permutations easily. Given a set $P = \{s_1, s_2, s_3, s_4\}$ of 4 permutations on $[n]$, we construct a new set $Q = (z_1, z_2, z_3, z_4)$ of 4 permutations on $[4n]$ by applying s_1, s_2, s_3, s_4 to four partitions of $[4n]$ as follows, such that the Ulam median for P can be obtained from the Ulam center for Q.

- $z_1 = (s_1([1, \dots, n]), s_2([n+1, \dots, 2n]), s_3([2n+1, 3n]), s_4([3n+1, 4n]))$
- $z_2 = (s_2([1, \dots, n]), s_3([n+1, \dots, 2n]), s_4([2n+1, 3n]), s_1([3n+1, 4n]))$
- $z_3 = (s_3([1, \dots, n]), s_4([n+1, \dots, 2n]), s_1([2n+1, 3n]), s_2([3n+1, 4n]))$
- $z_4 = (s_4([1, \dots, n]), s_1([n+1, \dots, 2n]), s_2([2n+1, 3n]), s_3([3n+1, 4n]))$

Let $c_Q \in \mathcal{S}_{4n}$ be an Ulam center of Q. The following are easy to see.

- $c_Q[1, \dots, n]$ does not contain any symbols from $[n+1, \dots, 4n]$;
- $c_Q[n+1, \dots, 2n]$ does not contain any symbols from $[1, \dots, n] \cup [2n+1, \dots, 4n]$;
- $c_Q[2n+1, \dots, 3n]$ does not contain any symbols from $[1, \dots, 2n] \cup [3n+1, \dots, 4n]$;
- $c_Q[3n+1, \dots, 4n]$ does not contain any symbols from $[1, \dots, 3n]$.

It is also easy to see that the four permutations are equidistant from c_Q. That is,

$$d(c_Q, z_1) = d(c_Q, z_2) = d(c_Q, z_3) = d(c_Q, z_4).$$

Finally, these facts are sufficient to claim that $c_Q[1, \dots, n]$ (or rather any one of the four partitions of $[4n]$) is an Ulam median for P. See [8, Theorem 6] for a comprehensive proof. (Although [8, Theorem 6] talks about the Kendall's tau distance, the argument could easily be extended for the Ulam metric.)

B Computing the Bi-coloring Number of a Colored Matrix

Here, we provide a dynamic programming algorithm that given any colored matrix M of dimension $m \times \ell$, computes $\mathsf{BCN}(M)$ in $O(m\ell^{m+1})$ time. More specifically, we prove Theorem 6.

For a clean description, we provide the dynamic program for $m = 4$, although it works for every positive integer m. We use C to denote our dynamic programming table. The cells of C store a Boolean value if the value is 1. C has 5 dimensions.

1. Subproblem: Let $C[i_1, i_2, i_3, i_4, k]$ denotes whether it is possible to leave at most i_1, i_2, i_3 and i_4 unpicked in rows 1 to 4 respectively by picking colors till column k. If it's not possible, the cell will contain a 0(False value). Otherwise it'll contain a 1(True value) along with the picked color. We denote the number of colors we can leave unpicked per row as the picking requirements for the cell in the dynamic program.
2. Computing C: Consider any column $k \geq 2$ and values $i_1 \geq 1, i_2 \geq 1, i_3 \geq 1$ and $i_4 \geq 1$. Picking any color in a row indicates, that the number of unpicked colors in all other rows increase by 1. Thus, (without loss of generality) M_{1j} could only be a feasibly choice if $C[i_1 - 1, i_2, i_3, i_4, k - 1]$ is true. If no choice of color in column k satisfies this property, then clearly we can't satisfy the picking requirements of the cell.
3. Recurrence:

$$\begin{aligned} C[i_1, i_2, i_3, i_4, k] &= C[i_1 - 1, i_2, i_3, i_4, k - 1] \vee D[i_1, i_2 - 1, i_3, i_4, k - 1] \\ &\quad \vee C[i_1, i_2, i_3 - 1, i_4, k - 1] \vee C[i_1, i_2, i_3, i_4 - 1, k - 1] \\ C[1, 0, 0, 0, 1] &= 1 \\ C[0, 1, 0, 0, 1] &= 1 \\ C[0, 0, 1, 0, 1] &= 1 \\ C[0, 0, 0, 1, 1] &= 1 \end{aligned}$$

4. Order of evaluation: We iterate over k one by one, and then evaluate over the first 4 indices lexicographically. Each cell queries lexicographically smaller cells.
5. Final Answer: Look at all the cells $C[i_1, i_2, i_3, i_4, \ell]$ for all $0 \leq i_1 \leq \ell, 0 \leq i_2 \leq \ell, 0 \leq i_3 \leq \ell, 0 \leq i_4 \leq \ell$ for which the cell contains a 1(True value). For each of these cells, let's denote the maximum of the quantity i_1, i_2, i_3 and i_4 as the i_{max} value for this cell. Output the cell with the minimum i_{max} value.

There are $(\ell + 1)^4 \times \ell$ sub-problems and for each cell, we look at 4 different sub-problems. Generalising to m strings: our dynamic program table has $(\ell + 1)^m \times \ell$ cells, and we look at m different cells to compute the answer for each cell. Thus our dynamic program runs in $m \times (\ell + 1)^m \times \ell$ time.

C Computing a Length-restricted Center String

For simplicity of exposition, we describe the dynamic programming algorithm FindString-Center only for three strings s_1, s_2, s_3. However, it can easily be extended to any number of strings in a natural way. We use D to denote our dynamic programming table. D stores a string and has 7 dimensions.

1. Subproblem: $D[i_1, i_2, i_3, k_1, k_2, k_3, \ell] = x_\ell$, where x_ℓ is an ℓ-length string with the following properties.
   a. $\mathsf{LCS}(s_1[1, 2, \ldots, i_1], x_\ell) \geq k_1$
   b. $\mathsf{LCS}(s_2[1, 2, \ldots, i_2], x_\ell) \geq k_2$
   c. $\mathsf{LCS}(s_3[1, 2, \ldots, i_3], x_\ell) \geq k_3$
   If such a string does not exist, then $D[i_1, i_2, i_3, k_1, k_2, k_3, \ell] = \emptyset$.
2. Computing D: Consider the substrings $s_1[1, 2, \ldots, i_1], s_2[1, 2, \ldots, i_2], s_3[1, 2, \ldots, i_3]$. The cases when there exists a string of length at most ℓ satisfying the above three conditions are listed below.
   a. At least one of the following is true.
      (i) $D[i_1, i_2, i_3, k_1, k_2, k_3, \ell - 1] \neq \emptyset$.
      (ii) $D[i_1 - 1, i_2, i_3, k_1 - 1, k_2, k_3, \ell] \neq \emptyset$.
      (iii) $D[i_1, i_2 - 1, i_3, k_1, k_2 - 1, k_3, \ell] \neq \emptyset$.
      (iv) $D[i_1, i_2, i_3 - 1, k_1, k_2, k_3 - 1, \ell] \neq \emptyset$.
      If the first of these four cases is true, then we can extend $x_{\ell-1}$ by putting an arbitrary symbol at the ℓ-th position in x_ℓ. In the other three cases, x_ℓ remains the same.
   b. At least one of the following is true.
      (v) If $D[i_1 - 1, i_2, i_3, k_1 - 1, k_2, k_3, \ell - 1] \neq \emptyset$, then $x_\ell \leftarrow x_{\ell-1} \circ s_1[i_1]$.
      (vi) If $D[i_1, i_2 - 1, i_3, k_1, k_2 - 1, k_3, \ell - 1] \neq \emptyset$, then $x_\ell \leftarrow x_{\ell-1} \circ s_2[i_2]$.
      (vii) If $D[i_1, i_2, i_3 - 1, k_1, k_2, k_3 - 1, \ell - 1] \neq \emptyset$, then $x_\ell \leftarrow x_{\ell-1} \circ s_3[i_3]$.
      (viii) If $D[i_1 - 1, i_2 - 1, i_3, k_1 - 1, k_2 - 1, k_3, \ell - 1] \neq \emptyset$ and $s_1[i_1] = s_2[i_2]$, then $x_\ell \leftarrow x_{\ell-1} \circ s_1[i_1]$.
      (ix) If $D[i_1, i_2 - 1, i_3 - 1, k_1, k_2 - 1, k_3 - 1, \ell - 1] \neq \emptyset$ and $s_2[i_2] = s_3[i_3]$, then $x_\ell \leftarrow x_{\ell-1} \circ s_2[i_2]$.
      (x) If $D[i_1 - 1, i_2, i_3 - 1, k_1 - 1, k_2, k_3 - 1, \ell - 1] \neq \emptyset$ and $s_1[i_1] = s_3[i_3]$, then $x_\ell \leftarrow x_{\ell-1} \circ s_1[i_1]$.
      (xi) If $D[i_1 - 1, i_2 - 1, i_3 - 1, k_1 - 1, k_2 - 1, k_3 - 1, \ell - 1] \neq \emptyset$ and $s_1[i_1] = s_2[i_2] = s_3[i_3]$, then $x_\ell \leftarrow x_{\ell-1} \circ s_1[i_1]$.
   c. None of the above are true.
      (xii) $D[i_1, i_2, i_3, k_1, k_2, k_3, \ell] = \emptyset$.
3. Recurrence: The cell $D[i_1, i_2, i_3, k_1, k_2, \ell]$ looks at all the possible 12 cases described above and does as mentioned in the points. The base case is $D[0, 0, 0, 0, 0, 0, 0] = \varepsilon$ where ε denotes the empty string.
4. Order of evaluation: We initialize by setting the cell $D[0, 0, 0, 0, 0, 0, 0] = \emptyset$ and proceed in lexicographic order. Note that each cell only queries lexicographically smaller cells.
5. Final Answer: Consider only those cells for which $D \neq \emptyset$. Let the string stored in each such cell σ be denoted by x_σ^*. Let $\mathsf{LCS}_\sigma^{\min}$ be one of the three strings $\{\mathsf{LCS}(x_\sigma^*, s_1), \mathsf{LCS}(x_\sigma^*, s_2), \mathsf{LCS}(x_\sigma^*, s_3)\}$, whichever has the minimum length. Compute $\mathsf{LCS}_\sigma^{\min}$ for the cell $D[n, n, n, k_1, k_2, k_3, n]$ for all $0 \leq k_1 \leq n, 0 \leq k_2 \leq n, 0 \leq k_3 \leq n$ (whenever $D[n, n, n, k_1, k_2, k_3, n] \neq \emptyset$). Among all these $\mathsf{LCS}_\sigma^{\min}$ strings, output the longest string as the final answer, denoted by x^*. (Note that x^* might not be of length n.)

If the final string x^* is of length less than n, then we fill in missing symbols from $[n]$ arbitrarily and make x^* an n-length string (denoted by x_n^*). Clearly adding more symbols to x^* cannot decrease $\mathsf{LCS}(s_i, x^*)$ for any of the s_i's.

D Pseudocodes from Section 4.2

Algorithm 1 RemoveDuplicate.

```
Result: Removes duplicate characters from $x_n^*$
Initialise $\alpha_i$ as an arbitrary occurrence of $\mathsf{LCS}(x_n^*, s_i)$ in $s_i$ $\forall i \in [m]$;
Initialise $M$ as an empty $m \times \ell$ matrix;
For $j \in [\ell]$, $a_j^k$ denotes the $k^{th}$ occurrence of $a_j$ in $x_n^*$ $\forall k \in \{1, 2, \ldots, \mathsf{freq}_j\}$;
for $(i, j) \in [m] \times [\ell]$ do
    if $a_j \in s_i$ then
        $M[i][j] \leftarrow k$, where $k$ is the unique index such that $a_j^k \in \alpha_i$
    end
    else
        $M[i][j] \leftarrow$ no-color
    end
end
$\mathcal{A} \leftarrow$ FindBiColoring($M$)
$\bar{x} \leftarrow \varepsilon$
for $i \in [n]$ do
    if $\exists j \in [\ell], k \in [\mathsf{freq}_j]$ such that $x_n^*[i] = a_j^k$ then
        if $k = \mathcal{A}[j]$ then
            $\bar{x} \leftarrow \bar{x} \circ x_n^*[i]$
        end
    end
    else
        $\bar{x} \leftarrow \bar{x} \circ x_n^*[i]$
    end
end
```

Algorithm 2 InsertMissing.

```
Result: Inserts missing characters into $\bar{x}$ so that $\bar{x} \in \mathcal{S}_n$
Initialise $\beta_i$ as an arbitrary occurrence of $\mathsf{LCS}(\bar{x}_n, s_i)$ in $s_i$ $\forall i \in [m]$;
$i \leftarrow 1$
while $\mathcal{M} \neq \emptyset$ do
    Pick any $b \in \mathcal{M}$
    $p \leftarrow s_i^{-1}(b)$
    if $\exists r \in [n]$ such that $r < p$, $s_1[r] \in \beta_i$ then
        $q \leftarrow \max\{r \in [n] \mid r < p, \ s_1[r] \in \beta_i\}$
        $j \leftarrow \bar{x}^{-1}(a)$
        $\bar{x} \leftarrow \bar{x}[1 \ldots j] \circ p \circ \bar{x}[j+1 \ldots \mathsf{len}(\bar{x})]$
    end
    else
        $\bar{x} \leftarrow b \circ \bar{x}$
    end
    $\mathcal{M} \leftarrow \mathcal{M} \setminus \{b\}$
    $i \leftarrow i \mod m + 1$
end
```