Listing Center Strings under the Edit Distance Metric*

Hiromitsu Maji[1] and Taisuke IZUMI[1]

Graduate School of Engineering, Nagoya Institute of Technology,
Nagoya, 466-8555, Japan.
cke17607@stn.nitech.ac.jp, t-izumi@nitech.ac.jp

Abstract. Given a set W of k strings of length n over an alphabet Σ, the center string of W is defined as the string w such that the maximum distance to w of all strings in W is minimized under some specified metric. We present a new algorithm for the decision version of this problem under the edit distance metric. Given a threshold parameter d, the algorithm lists all the strings such that the distance from any input string is bounded by d in $O((3^d(d+2))^k dk|\Sigma|n + Mn)$ time, where M is the number of the output strings. To the best of our knowledge, this is the first FPT algorithm for the center string under the edit distance metric (even as a finding algorithm). By a slight modification, we also obtain an algorithm listing length-l common subsequences of W, which runs in $O((n-l)^{k+1}k|\Sigma|l + Ml)$ time.

1 Introduction

Finding a common structure from a given set of strings is recognized as one of the important problems in computational biology. A *center string* (or equivalently, *closest string*) is the one that minimizes the maximum distance of all strings in a input set W under some specific metric. The decision version of the center string problem under metric δ, which is the primary problem considered in this paper, is formalized as follows:

> **Input**: A set W of k strings of length n over an alphabet Σ, and a threshold value $d \in \mathbb{N}$.
> **Output**: A string w such that $\delta(w, w') \leq d$ holds for any $w' \in W$ if it exists. Otherwise the value of "FALSE".

In the definition above, the distance metric is not concretely defined. Usually it is chosen according to applications. Popular metrics useful in many applications are *Hamming distance* and *edit distance*. Unfortunately, for both metrics, the center string problem is NP-complete [4], and thus we need some sort of relaxation for attacking this problem. In this paper, we consider fixed-parameter algorithms for the center string problem under the edit distance metric. However, unfortunately again, that problem is W[1]-hard with respect to the number of

* This work is supported in part by KAKENHI No. 15H00852 and 25289227.

input strings k [16], which implies that the problem is unlikely to have an algorithm with a running time such as $O(f(k)\cdot poly(n))$. The currently best algorithm is the one by Nicolas et al. [16], which achieves $O(|\Sigma|n^k)$ time bound.

To circumvent the hardness results above, we focus on the fixed-parameter tractability for parameters both d and k. That is, we explore the algorithms having a running time with the form of $O(f(d,k)\cdot poly(n))$. The primary contribution of this paper is that such an algorithm actually exists. Our new algorithm finds a center string in $O((3^d(d+2))^k dk|\Sigma|n)$ time . To the best of our knowledge, this is the first FPT algorithm for finding center string problem under the edit distance metric. By a simple extension of this finding algorithm , we propose an algorithm to list all the solutions in the output-sensitive manner: The algorithm lists all the center strings for the edit distance metric in $O((3^d(d+2))^k dk|\Sigma|n + Mn)$ time, where M is the number of the output strings. Since in typical scenarios the center string problem is considered with a small d, our algorithms are practically more useful than the previous one.

The algorithms are constructed with a new dynamic-programming strategy, where the DP table records the distance information around diagonal vertices in alignment graphs. Interestingly, we can utilize the same strategy to solve another problem. Our second result is an algorithm listing length-l common subsequences of all input strings. The time complexity of this algorithm is $O((n-l)^{k+1}|\Sigma|l + Ml)$. Note that the longest common subsequence (LCS) problem, which is the optimization version of the length-l common subsequence problem, is known to be NP-complete [14], and W[1]-hard for parameter k [17]. On the other hand, finding length-l common subsequences trivially allows an $O(|\Sigma|^l poly(n,k))$-time algorithm by checking all strings of length l. That is, it is fixed-parameter tractable for parameter l in the case of constant-size alphabets. Furthermore, Irving and Fraser shows two algorithms of finding a LCS of length at least l in $O((n-l)^{k-1}kn)$ and $O((n-l)^{k-1}kl + k|\Sigma|n)$ times respectively [9]. That is, finding a common subsequence with length near to n is also fixed-parameter tractable (for $n-l$). Our algorithm can be seen as a listing version of the two algorithms by Irving and Fraser.

The paper is organized as follows: In Section 2, we present the prior work related to the topics of this paper. Section 4 provides our algorithm for the center string problem. Its extension to the LCS problem is considered in Section 5 Finally, we conclude this paper with a future direction in Section 6.

2 Related Work

The center string problem for the Hamming distance metric (often called *closest string problem*) is extensively studied. In general, that problem is NP-complete [6, 11], but allows a fixed-parameter algorithm with respect to d. Following the first FPT-algorithm by Gramm et al. [7], a number of papers improved the time complexity [3, 13, 18]. The closest substring problem, which is a generalized version of the closest string problem, is also well studied. Interestingly the closest substring problem is W[1]-hard with respect to both d and k even if the alpha-

bet is binary [15]. Marx shows an efficient algorithm for computing the closest substring for small d and/or k (but it is not an FPT algorithm) [15].

Compared to the Hamming distance metric, the center string problem under the edit distance metric is less studied. As we stated in the introduction, the paper by Nicolas and Rivals is only the one explicitly considering that setting [16]. The case of other metrics is considered in [5].

The longest common subsequence (LCS) problem for multiple strings is regarded as a special case of the center string for the edit distance metrics. It is equivalent to the center string problem for the edit distance metric with substitution cost two. About exact solutions for the LCS problem, a few papers propose several algorithms with different characteristics [8, 9]. The LCS problem for some restricted instances is considered in [1, 2].

Another variant of the center string problem is the median string problem, which requires to find the string minimizing the sum of the distance to each input string. While the median string under the Hamming distance metric is easily solvable in polynomial time, the case for edit distance is known to be NP-complete [4], and W[1]-hard for parameter k [16].

Approximated solutions for the problems introduced above are also investigated [11, 12]. PTASs are allowed for the closest (sub)string problem [12], but the longest common subsequence problem has no polynomial-time algorithm with any approximation ratio better than n^c for some constant $c > 0$ unless P = NP [10]. No polynomial-time approximated solution for the center string problem under the edit distance metric is known so far.

3 Preliminaries

3.1 Edit Distance

We denote the alphabet by Σ. An element in Σ^* is called *string*. The length of a string w is denoted by $|w|$, and the i-th character of w is denoted by $w[i]$ ($1 \leq i \leq |w|$). The operator $\circ$ means the concatenation of two strings (or characters). For $w \in \Sigma^*$, let $ta(w)$ be the string obtained by removing the first character of w. That is, $w = w[1] \circ ta(w)$. Letting W be a set of strings, we define $W \circ x = \{w \circ x | w \in W\}$.

The *edit distance* $ED(w_1, w_2)$ between two strings w_1 and w_2 is defined as follows:

$$ED(w_1, w_2) = \begin{cases} \max\{|w_1|, |w_2|\} \\ \qquad\qquad\qquad\qquad (\text{if } |w_1| = 0 \vee |w_2| = 0) \\ \min\{ED(ta(w_2), ta(w_1)) + c(w_1[1], w_2[1]), \\ \ ED(ta(w_1), w_2) + 1, ED(w_1, w_2) + 1\} \\ \qquad\qquad\qquad\qquad (\text{otherwise}), \end{cases}$$

where $c(a, b)$ is the function returning zero if $a = b$ or one otherwise. Note that while we assume that $c(a, b)$ is uniform (i.e.,the substitution cost does not depend on target characters), our algorithm can be applied to the case of non-uniform cost functions (as long as it returns an integer value).

It is well-known that the computation of the edit distance between two strings w_1 and w_2 can be reduced to the shortest path problem for some directed acyclic graph $G(w_1, w_2) = (V, E, f)$, called *alignment graph*, which defined as follows (Fig. 1):

- $V = \{v_{i,j} | i, j \in [0, n]\}$.
- For any $i, j \in [0, n]$, $v_{i,j}$ has a directed edge to each vertex $v_{i+1,j}$, $v_{i,j+1}$, and $v_{i+1,j+1}$ if it exists.
- $f(e) = \begin{cases} 0 \text{ if } e = (v_{i,j}, v_{i+1,j+1}) \text{ for some } i, j \in [0, n] \text{ and } w_1[i] = w_2[j] \\ 1 \text{ otherwise.} \end{cases}$

An edge $e = (v_{i,j}, v_{i',j'}) \in E$ is called a *horizontal*, *vertical*, or *diagonal* edge if $i = i'$, $j = j'$, or $(i \neq i' \wedge j \neq j')$ holds respectively. The set of vertices $\{v_{i,j} | j \in [0, n]\}$ and $\{v_{i,j} | i \in [0, n]\}$ are called *i-th row* and *j-th column* respectively. The distance between two vertices u and v is denoted by $dist(u, v)$. In particular, if $u = v_{0,0}$, we omit the first argument and describe $dist(u)$ for short. The following theorem is a classical fact.

Theorem 1. $dist(v_{n,n}) = ED(w_1, w_2)$.

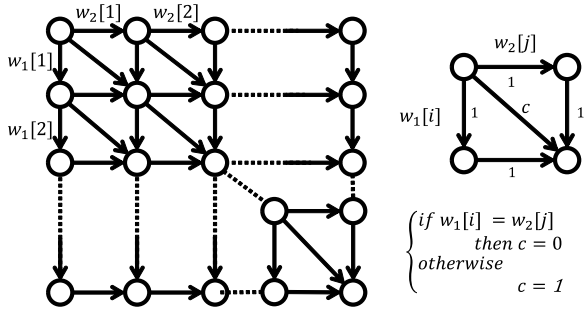


Fig. 1. Alignment graph $G(w_1, w_2)$

The *band* of alignment graphs is defined as the set of vertices $\{v_{i,j} | |i - j| \leq d/2\}$[1]. We also define $h(j)$ as the intersection size of the j-th column and the band. That is, let $h(j) = \min\{j + d/2, d, (n - j) + d/2\}$. For ease of explanation, we give aliases to each vertex in the band: The vertices of the j-th column in the band are called $u_{0,j}, u_{1,j}, \ldots, u_{h(j),j}$ from the upper side. The notations above are illustrated in Fig. 2.

We have the following lemma:

[1] The definition of the band depends on the value of d. Hence it may be more precise to include that dependency in the notation (e.g., calling d-band). However, to avoid the complication of notations, we treat the value of d as a certain kind of "global constant." Actually, in the following argument, we introduce several definitions dependent on d with no explicit description of the dependency.

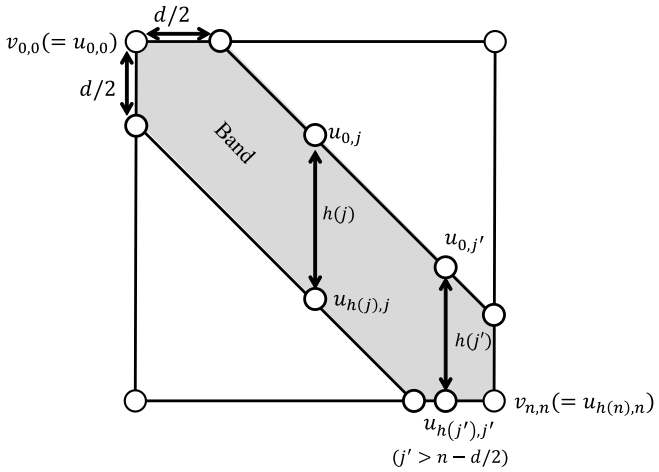


Fig. 2. Band and vertex aliasing

Lemma 1. *Given $w_1, w_2 \in \Sigma^n$ satisfying $ED(w_1, w_2) \leq d$, all the vertices constituting the shortest path from $v_{0,0}$ to $v_{n,n}$ in $G(w_1, w_2)$ are contained in the band.*

Proof. The shortest path from $v_{i,j}$ to $v_{n,n}$ must contain at least $|i - j|$ non-diagonal edges, all of which have weight one. Thus its length is more than or equal to $|i - j|$. Similarly, the length of the shortest path from $v_{0,0}$ to $v_{i,j}$ is also more than or equal to $|i - j|$. If $|i - j| > d/2$ (i.e., $v_{i,j}$ is out of the band), the length of any path from $v_{0,0}$ to $v_{n,n}$ via $v_{i,j}$ is more than d. It follows that $v_{i,j}$ cannot be contained in the shortest path because $dist(v_{n,n}) = ED(w_1, w_2) \leq d$. □

The lemma above implies that it suffices to consider the subgraph of $G(w_1, w_2)$ induced by the band because we only care about the paths from $v_{0,0}$ to $v_{n,n}$ of length at most d. We denote that induced subgraph by $B(w_1, w_2)$. The terminologies and notations introduced for $G(w_1, w_2)$ are also used for $B(w_1, w_2)$. In the following argument, we sometimes treat $B(w_1, w_2)$ for some string w_2 of length less than n. So we extend the definition of $B(w_1, w_2)$: For strings $w_1 \in \Sigma^n$ and $w_2 \in \Sigma^m$ such that $m < n$, we define $B(w_1, w_2)$ as the one in which edge $e = (v_{i,j}, v_{i',j'})$ for $j < m$ has the weight according to the original function f, and all other edges have weight one.

4 Listing Center Strings

In this section, we propose an algorithm for the center string problem, called $\mathsf{ListCenter}(W)$. The core idea of $\mathsf{ListCenter}(W)$ is to compute the intersection of

the k balls of radius d centered at each input string in $W = \{w_1, w_2, \cdots, w_k\}$. Thus, before explaining the main algorithm, we first introduce a preliminary algorithm called $\mathsf{ListBall}(w)$, which lists all the strings whose edit distance from w is at most d. The main algorithm is obtained by a simple extension of $\mathsf{ListBall}(w)$.

4.1 Algorithm ListBall: Listing Strings within Distance d

Letting $\lceil x \rceil_{d+1} = \min\{d+1, x\}$ for short, we define the (w_1, j)-*profile* of string w_2 as the vector $(\lceil dist(u_{0,j}) \rceil_{d+1}, \lceil dist(u_{1,j}) \rceil_{d+1}, \ldots, \lceil dist(u_{h(j),j}) \rceil_{d+1})$, where $dist(u_{i,j})$ is the distance in $B(w_1, w_2)$. Intuitively, the (w_1, j)-profile of w_2 is the distance vector to the vertices of the j-th column in the band, but the information about distances exceeding d are omitted. Without ambiguity, we often omit w_1 and simply call j-profile.

It should be noted that any $(h(j)+1)$-dimensional vector cannot become a j-profile. we say that $P \in [0, d+1]^{h(j)+1}$ is *possible* if there exists $w_1, w_2 \in \Sigma^n$ such that P becomes the (w_1, j)-profile of w_2. We can show a necessary condition for the possibility of P:

Lemma 2. *If $P = (p_0, p_1, \ldots, p_{h(j)}) \in [0, d+1]^{h(j)+1}$ is a possible j-profile, $|p_i - p_{i+1}| \leq 1$ holds for any $i \in [0, h(j)-1]$.*

Proof. Since $dist(u_{i+1,j}) - dist(u_{i,j}) \leq 1$ obviously holds because edge $(u_{i,j}, u_{i+1,j})$ has weight one, it suffices to show $dist(u_{i,j}) - dist(u_{i+1,j}) \leq 1$. Let $u_{0,0} = x_0, \cdots, x_r = u_{i+1,j}$ be the shortest path from $u_{0,0}$ to $u_{i+1,j}$ in $B(w_1, w_2)$, x_c be the last vertex contained in the i-th row, and $dist(x_c) = l$ (see Fig. 3). Since x_c is reachable to $u_{i,j}$ only traversing horizontal edges, $dist(u_{i,j}) \leq l + (r - c)$ holds. The shortest path from x_c to $u_{i+1,j}$ can contain at most one diagonal edge, its length is at least $r - c - 1$, and thus $dist(u_{i+1,j}) \geq l + (r - c - 1)$ holds. Consequently, we have $dist(u_{i,j}) - dist(u_{i+1,j}) \leq 1$. □

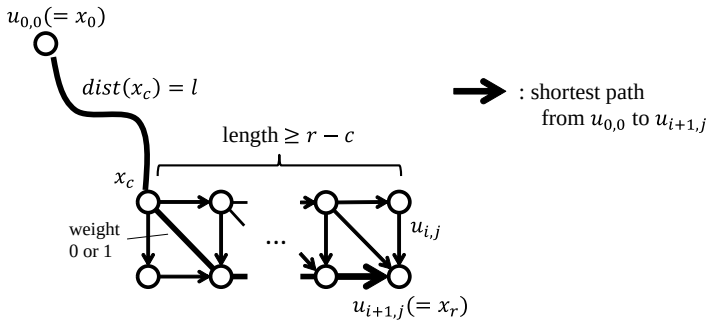


Fig. 3. Proof of Lemma 2

Let $\mathcal{P}_j$ be the set of the sequences in $[0, d+1]^{h(j)+1}$ satisfying the condition of Lemma 2. Clearly $\mathcal{P}_j$ contains all the possible profiles. From Lemma 2, we have the following corollary:

Corollary 1. *For any $j \in [0, n]$, $|\mathcal{P}_j| \leq 3^d(d+2)$.*

We also have the following corollary from the definition of profiles.

Corollary 2. *Any string w has 0-profile $P_{init} = (0, 1, 2, \ldots, h(0))$.*

For $P = (p_0, p_1, \ldots, p_{h(j)}) \in \mathcal{P}_j$ and $w \in \Sigma^n$, we define $S^w_{P,j}$ as the set of length-j strings having P as its (w, j)-profile. Let $\mathcal{P}_{term} = \{(p_0, p_1, \ldots, p_{h(n)}) \in [0, d+1]^{h(n)+1} | p_{h(n)} \leq d\}$. Then the union $\cup_{P \in \mathcal{P}_{term}} S^w_{P,n}$ is the set of the strings to be listed as the computation result. The core idea of $\mathsf{ListBall}(w)$ is to compute $S^w_{P,j}$ for any P and j via dynamic programming. To lead the DP recurrence formula, we introduce one more notation defined as follows: Let $j \in [0, n-1]$ and $w \in \Sigma^n$. For two vectors $P = (p_0, p_1, \ldots, p_{h(j)}) \in [0, d+1]^{h(j)+1}$ and $Q = (q_0, q_1, \ldots, q_{h(j+1)}) \in [0, d+1]^{h(j+1)+1}$, we say that P *is connected to* Q *with* $(x, w) \in \Sigma \times \Sigma^*$, denoted by $P \overset{w,x,j}{\longmapsto} Q$, if there exists a string w' such that $w'[j+1] = x$ and P and Q are respectively the (w, j)-profile and $(w, j+1)$-profile of w'. The key fact to obtain the recurrence formula is the next lemma:

Lemma 3. *Fixing $w \in \Sigma^n$, the $(j+1)$-profile Q satisfying $P \overset{w,x,j}{\longmapsto} Q$ is uniquely determined from P and x in $O(d)$ time.*

Proof. The uniqueness of Q is obvious because any shortest path to a vertex in the $(j+1)$-th column must pass a vertex in the j-th column in $B(w, *)$. Thus we prove that Q can be computed in $O(d)$ time. In graph $B(w, *)$, we can determine the weights of all the edges between jth and $(j+1)$-th columns by x. Thus we can compute Q by calculating the distances up to d from $v_{0,0}$ to the vertices in the $(j+1)$-th column, provided distances up to d to the vertices in the j-th column. For any $i' \in [0, n]$, the predecessor of $v_{i',j+1}$ in the shortest path from $v_{0,0}$ to $v_{i',j+1}$ is either $v_{i'-1,j}$, $v_{i',j}$, or $v_{i'-1,j+1}$. So if the distances (up to d) to those vertices are already known, the shortest path to $v_{i',j+1}$ (with length up to d) can be computed in a constant time. This implies that the values of the $(j+1)$-profile can be fixed from the upper side sequentially (i.e., in the order of $u_{j+1,0}, u_{j+1,1}, \ldots, u_{j+1,h(j+1)}$). Since $h(j+1) = O(d)$ holds, we can compute Q in $O(d)$ time. The lemma is proved. □

This lemma implies that if we know the j-profile of a string w, we can know the $(j+1)$-profile of $w \circ x$ for any $x \in \Sigma$. Conversely, if we want to know some (unknown) string $w = w' \circ x$ having some $(j+1)$-profile, it suffices to idenfity w' and its j-profile. This fact induces the following recurrence formula.

$$S^w_{Q,j+1} = \bigcup_{\substack{x \in \Sigma \\ P : P \overset{w,x,j}{\longmapsto} Q}} S^w_{P,j} \circ x. \tag{1}$$

Now we are ready to explain the algorithm, which consists of the following two steps:

– The first step of the algorithm is to construct the edge-labeled DAG $\Gamma = (V_\Gamma, E_\Gamma, f_\Gamma)$ defined as follows:
  - $V_\Gamma = (\cup_{j=0}^{n}\{(P,j)|P \in \mathcal{P}_j\} \cup \{t\}$, where t is the special sink vertex. We also give alias s to the vertex $(P_{init}, 0)$.
  - A vertex (P,j) is connected to $(Q, j+1)$ by an edge with label x if $P \overset{w,x,j}{\longmapsto} Q$. Note that if two or more characters x satisfy $P \overset{w,x,j}{\longmapsto} Q$, (P,j) and $(Q,j+1)$ are connected by multiedges. Finally, we add edges from all the vertices (P,n) satisfying $P \in \mathcal{P}_{term}$ to t with the null-character label.
– For any s-t path $X = e_0, e_1, \ldots e_n$ in Γ, we define $\gamma(X)$ as the string formed by traversing X (i.e., $\gamma(X) = f_\Gamma(e_0) \circ f_\Gamma(e_1) \circ \cdots \circ f_\Gamma(e_n)$). The second step of the algorithm is to output $\gamma(X)$ for each s-t path X in Γ.

The correctness of this algorithm relies on the following lemma:

Lemma 4. *For any $(P,j) \in V_\Gamma$ and a string $w_2 \in \Sigma^j$, $w_2 \in S^{w_1}_{P,j}$ holds if and only if there exists a path X from s to (P,j) such that $w_2 = \gamma(X)$ holds.*

Proof. The proof is done by induction on j. (**Basis**): It is obvious from Corollary 2. (**Induction step**): Suppose as the induction hypothesis that the lemma holds for $j = j' - 1$ and consider the case of $j = j'$. We prove only the direction of $\Rightarrow$ because the opposite direction ($\Leftarrow$) is easily proved by following backward the argument. Let $w_2 = w_2' \circ x$. The recurrence formula (Eq. 1) implies that if $w_2 \in S^{w_1}_{P,j'}$, there exists a $(j'-1)$-profile P' such that $w_2' \in S^{w_1}_{P',j'-1}$ and $P' \overset{w,x,j'-1}{\longmapsto} P$ hold. Then, by the induction hypothesis, there exists a path X' from s to (P', j') and $w_2' = \gamma(X')$. So there exists a path X to (P,j') from s and $\gamma(X) = w_2' \circ x$. The lemma is proved. □

We consider the running time of the algorithm. In the first step, for each vertex (P,i), the algorithm needs to compute all the pairs (x,Q) such that $P \overset{w,x,i}{\longmapsto} Q$ holds. From Lemma 3, it can be computed in $O(|\Sigma|d)$ time. From Corollary 1, we also have $|V_\Gamma| = O(3^d dn)$. Thus the total running time of the first step is $O(3^d(d^2|\Sigma|n)$. For the second step, all s-t paths can be enumerated by the naive recursion after pruning the vertices from which t is unreachable. Its running time is $O(Mn + |E_\Gamma|)$, where M is the number of paths enumerated. Finally, we have the following theorem.

Theorem 2. *Given $w \in \Sigma^n$, algorithm ListBall(w) lists all the strings whose distance from w is less than or equal to d in $O(3^d d^2|\Sigma|n + Mn)$ time.*

4.2 Listing Center Strings

By extending algorithm ListBall(w), we construct the main algorithm ListCenter(W). The primary idea is that given $W = \{w_1, w_2, \ldots, w^k\}$, ListCenter$(W)$ concurrently runs ListBall(w_i) for each input $w_i \in W$. We give the detailed explanation below:

A k-tuple of profiles $\mathbf{P} = (P_1, P_2, \ldots, P_k) \in \mathcal{P}_j^k$ is called the (W, j)-*profile* of a string w if P_i is w's (w_i, j)-profile for any $i \in [1, k]$. The notation $\mathbf{P} \overset{W,x,j}{\longmapsto} \mathbf{Q}$ for $\mathbf{P} = (P_1, P_2, \ldots, P_k) \in \mathcal{P}_j^k$ and $\mathbf{Q} = (Q_1, Q_2, \ldots, Q_k) \in \mathcal{P}_j^k$ means that there exists $w \in \Sigma^n$ and $j \in [0, n]$ such that $\mathbf{P}$ and $\mathbf{Q}$ are w's (W, j)-profile and $(W, j+1)$-profile respectively and $w[j] = x$ holds.

The remaining structure of $\mathsf{ListCenter}(W)$ is almost the same as $\mathsf{ListBall}(w)$, but the definition of profiles and its connectivity relationship are replaced by the ones above. More precisely, the algorithm utilizes the graph Γ^k defined as follows, instead of Γ:

- $V_\Gamma = (\cup_{i=0}^{n} \{(\mathbf{P}, i) | \mathbf{P} \in \mathcal{P}_i^k\} \cup \{t\}$, where t is the special sink vertex. We also give alias s to $((P_{init}, P_{init}, \ldots, P_{init}), 0)$.
- A vertex $(\mathbf{P}, i)$ is connected to $(\mathbf{Q}, i+1)$ by an edge with label x if $\mathbf{P} \overset{W,x,i}{\longmapsto} \mathbf{Q}$. Note that if two or more characters x satisfy $\mathbf{P} \overset{W,x,i}{\longmapsto} \mathbf{Q}$, $(\mathbf{P}, i)$ and $(\mathbf{Q}, i+1)$ are connected by multiedges. Finally, we add the edges from all the vertices $(\mathbf{P}, n)$ satisfying $\mathbf{P} \in \mathcal{P}_{term}^k$ to t with the null-character label.

It is not difficult to prove that the string $\gamma(X)$ corresponding to a s-t path X in Γ^k has a distance at most d to each string $w_i \in W$. Thus by enumerating all s-t paths we can list all center strings. We bound the running time of this algorithm. The analysis of the second step completely follows that for $\mathsf{ListBall}(w)$. For the first step, the size of Γ^k is larger than Γ. The number of vertices in Γ^k is $O((3^d(d+2))^k n)$. In addition, the computation of outgoing edges for each vertex takes obviously k times of the case for $\mathsf{ListBall}(w)$, i.e., $O(k|\Sigma|d)$ time. Hence the total running time of the first step is $O((3^d(d+2))^k dk|\Sigma|n)$. Consequently we have the following main theorem.

Theorem 3. *Algorithm $\mathsf{ListCenter}(W)$ lists all center strings for W under the edit distance metric in $O((3^d(d+2))^k dk|\Sigma|n + Mn)$ time, where M is the number of output strings.*

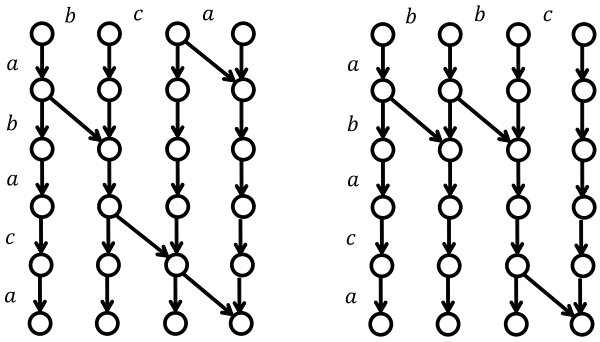
5 Listing Common Subsequences

A subsequence of a string $w \in \Sigma^n$ is any string obtained from w by deleting several characters. We denote by $Sub(w)$ the set of all subsequences of w. The decision version of the *longest common subsequence problem* (LCS) is defined as follows:

> **Input**: A set $W = \{w_1, w_2, \cdots, w_k\}$ of k strings over Σ of length n, and a threshold value $l \in \mathbb{N}$.
> **Output**: A string $w \in \cap_{i=1}^{k} Sub(w_i)$ such that $|w| \geq l$ if it exists. Otherwise the value of "FALSE".

Let $\bar{l} = n - l$ for short. In this section we show an algorithm called $\mathsf{ListLCS}_l(W)$, which is an algorithm listing common subsequences of length l. This algorithm is obtained by a refinement of $\mathsf{ListCenter}(W)$. For $w_1 \in \Sigma^n$ and $w_2 \in \Sigma^l$, we construct the *LCS alignment graph* $G_{LCS}(w_1, w_2) = (V_{LCS}, E_{LCS})$ as follows:

– $V_{LCS} = \{v_{i,j} | i \in [0, n], j \in [0, l]\}$.
– For any $i \in [0, n-1]$ and $j \in [0, l]$, add $e = (v_{i,j}, v_{i,j+1})$. In addition, add $e = (v_{i,j}, v_{i+1,j+1})$ if $w_1[i] = w_2[j]$.



(a) reachable (b) unreachable

Fig. 4. Two examples of LCS alignment graphs.

Note that LCS alignment graphs are unweighted. It is not difficult to prove the following lemma:

Lemma 5. *A string w_2 is a subsequence of a string w_1 if and only if $v_{0,0}$ is reachable to $v_{n,l}$ in $G_{LCS}(w_1, w_2)$.*

Two examples of LCS alignment graphs, which correspond to reachable and unreachable cases respectively, are shown in Fig. 4. We define the band of LCS alignment graphs as the set of vertices $\{v_{i,j} | j \leq i \leq j + \bar{l}, j \in [0, n]\}$ (see Fig. 5). The following lemma is analogous to Lemma 1 in the center-string case.

Lemma 6. *Any vertex $v_{i,j}$ out of the band is either unreachable to $v_{0,0}$ or $v_{n,l}$.*

Similarly as the center string case, let $B_{LCS}(w_1, w_2)$ be the subgraph of $G_{LCS}(w_1, w_2)$ induced by the band. Now we introduce the refined definition of profiles: The (w_1, j)-profile of w_2 is a binary $(\bar{l}+1)$-dimensional vector representing the reachability from $v_{0,0}$ to each vertex in the j-th column That is, a vertex $v_{i+j,j}$ is reachable from $v_{0,0}$ in $B_{LCS}(w_1, w_2)$ if and only if the (w_1, j)-profile $P \in [0, 1]^{\bar{l}+1}$ of w_2 satisfies $P[i] = 1$. Since $v_{i,j'}$ for $j' > j$ is reachable from $v_{0,0}$ when $v_{i,j}$ is reachable from $v_{0,0}$, any possible j-profile can be represented as the concatenation of an all-zero sequence followed by an all-one sequence. Furthermore, we do not have to consider the all-zero vector as a profile. Therefore, the total number of possible j-profiles is at most $\bar{l}$. We set $\mathcal{P}_j$ to all possible j-profiles.

The remaining part of algorithm $\mathsf{ListLCS}_l(W)$ is almost the same as $\mathsf{ListCenter}(W)$. Following the definition of profiles above, we construct the graph Γ^k and enumerate all s-t paths in Γ^k. Only the difference is the design of the source and the edges incoming to the sink in Γ^k. In the context of listing common subsequences, the $(\bar{l}+1)$-dimensional all-one vector is the unique possible 0-profile, and thus s is set to it. The vertices in $\{(P,l)|P[\bar{l}]=1\}$ is adjacent to t.

By the analysis similar with Section 4, we can bound the running time of this algorithm as follows:

Theorem 4. *For any set of k strings $W=\{w^1,w^2,\dots w^k\}$, algorithm $\mathsf{ListLCS}_l(W)$ enumerates all length-l common sequences in $O(\bar{l}^{k+1}k|\Sigma|l+Ml)$ time, where M is the number of output strings.*

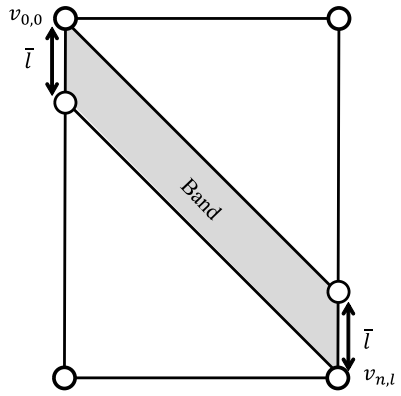


Fig. 5. Band of LCS alignment graphs.

6 Concluding Remarks

In this paper, we presented two algorithms called $\mathsf{ListCenter}(W)$ and $\mathsf{ListLCS}_l(W)$. Algorithm $\mathsf{ListCenter}$ enumerates all the center strings for given k strings and a threshold distance d in $O((3^d(d+2))^k dk|\Sigma|n+Mn)$ time. In addition, this algorithm finds one solution in $O((3^d(d+2))^k dk|\Sigma|n)$ time, which is the first FPT algorithm for the center string problem under the edit distance metric. Algorithm $\mathsf{ListLCS}_l$ is designed with the same framework as $\mathsf{ListCenter}$, which enumerates length-l common subsequences for k strings in $O(\bar{l}^{k+1}k|\Sigma|l+Ml)$ time.

On the parameterized complexity of the center string problem under the edit distance metric is surprisingly less studied. An important open problem is to show the fixed-parameter (in)tractability with respect to d only. While the

authors conjecture W[1]-hardness of that setting, the proof is still missing. Even if it is actually W[1]-hard, the exploration of faster algorithms (for example, running in $O(d^k \cdot poly(n)$ time) is also an interesting open problem.

References

1. G. Blin, P. Bonizzoni, R. Dondi, and F. Sikora. On the parameterized complexity of the repetition free longest common subsequence problem. *Inf. Process. Lett.*, 112(7):272–276, 2012.
2. G. Blin, L. Bulteau, M. Jiang, P. Tejada, and S. Vialette. Hardness of longest common subsequence for sequences with bounded run-lengths. In *Proc. of 23rd Annual Symposium on Combinatorial Pattern Matching*, pages 138–148. 2012.
3. Z.-Z. Chen and L. Wang. Fast exact algorithms for the closest string and substring problems with application to the planted (l,d)-motif model. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 8(5):1400–1410, 2011.
4. C. de la Higuera and F. Casacuberta. Topology of strings: Median string is np-complete. *Theoretical Computer Science*, 230(1-2):39 – 48, 2000.
5. L. P. Dinu and A. Popa. On the closest string via rank distance. In *In Proc. of 23rd Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 7354 of *Lecture Notes in Computer Science*, pages 413–426. 2012.
6. M. Frances and A. Litman. On covering problems of codes. *Theory of Computing Systems*, 30(2):113–119, 1997.
7. J. Gramm, R. Niedermeier, P. Rossmanith, et al. Fixed-parameter algorithms for closest string and related problems. *Algorithmica*, 37(1):25–42, 2003.
8. K. Hakata and H. Imai. The longest common subsequence problem for small alphabet size between many strings. In *Proc. of the 3rd International Symposium on Algorithms and Computation (ISAAC)*, pages 469–478, 1992.
9. R. W. Irving and C. B. Fraser. Two algorithms for the longest common subsequence of three (or more) strings. In *In Proc. of 3rd Annual Symposium on Combinatorial Pattern Matching*, volume 644, pages 214–229. 1992.
10. T. Jiang and M. Li. On the approximation of shortest common supersequencesand longest common subsequences. *SIAM J. Comput.*, 24(5):1122–1139, oct 1995.
11. J. K. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang. Distinguishing string selection problems. *Information and Computation*, 185(1):41 – 55, 2003.
12. M. Li, B. Ma, and L. Wang. On the closest string and substring problems. *J. ACM*, 49(2):157–171, 2002.
13. B. Ma and X. Sun. More efficient algorithms for closest string and substring problems. *SIAM J. Comput.*, 39(4):1432–1443, 2009.
14. D. Maier. The complexity of some problems on subsequences and supersequences. *J. ACM*, 25(2):322–336, 1978.
15. D. Marx. Closest substring problems with small distances. *SIAM J. Comput.*, 38(4):1382–1410, 2008.
16. F. Nicolas and E. Rivals. Hardness results for the center and median string problems under the weighted and unweighted edit distances. *Journal of Discrete Algorithms*, 3(2-4):390 – 415, 2005.
17. K. Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *Journal of Computer and System Sciences*, 67(4):757–771, 2003.
18. L. Wang and B. Zhu. Efficient algorithms for the closest string and distinguishing string selection problems. In *Frontiers in Algorithmics*, pages 261–270. 2009.