

Faster Algorithms for Finding Missing Patterns

Shuai Cheng Li

School of Computer Science
University of Waterloo
Waterloo ON N2L 3G1 Canada
Email: scl@cs.uwaterloo.ca

Abstract

The missing pattern pair problem, introduced in (Inenaga, Kivioja & Mäkinen 2004), was motivated by the need for optimization in Polymerase Chain Reaction, a technique commonly used in bioinformatics. The problem is to find a pair of patterns of the shortest total length within a string of length n , where the two patterns do not occur within a distance α anywhere in the string. Inenaga *et al.* (Inenaga et al. 2004) gave an algorithm with time complexity $O(\min\{\alpha n \log n, n^2\})$ to solve this problem. In this paper we propose an algorithm of time complexity $O(\min\{\alpha n \log n, n^{3/2}\})$, improving on the quadratic bound part of the earlier algorithm. We also design a simple algorithm of time complexity $O(\frac{n^2}{\alpha} \log^2 n)$, which is $O(n \log^2 n)$ if $\alpha = \Theta(n)$.

Keywords: pattern discovery, complexity, algorithm.

1 Introduction

Pattern discovery problems are among the most intensively studied problems in bioinformatics (Wang, Shapiro & Shasha 1999). An example of such problems is that of finding a pattern which does not appear in a given string — this is known as the *missing pattern problem*. This problem can be solved in time $O(n)$ where n is the length of the given string.

Inenaga *et al.* (Inenaga et al. 2004) introduced the problem called *missing pattern pair* (MPP) problem, where we are to find a pair of patterns of the shortest total length which do not appear in a given string S within a predefined distance α . An algorithm of time complexity $O(\min\{\alpha n \log n, n^2\})$ was given in (Inenaga et al. 2004). The problem has practical use in optimizing the sensitivity of Polymerase Chain Reaction methods — a standard technique for producing many copies of a region of DNA. In this paper we give an algorithm of $O(\min\{\alpha n \log n, n^{3/2}\})$ runtime, which should improve performance for the cases where α is large.

In the following subsection, a brief review of the biological motivation of this problem is presented, for a detailed version, please refer to (Inenaga et al. 2004).

1.1 Biological Motivations

Polymerase Chain Reaction (PCR) is used routinely to producing multiple copies of a sub sequence of

DNA. Primers in PCR refer a pair of short sequence. The two primers hybridize to their binding side of a target sequence, and this flanking the target sequence and makes the duplication of the flanked area possible. One of the problem is that the primers can bind to other sites rather than the targeting sites, and result in incorrect flanking. In order to overcome this problem, *Adapter primers* is designed. For specific primers, adapter primers bind short sequence to them. It is argued in (Inenaga et al. 2004) that the PCR process will be facilitated by identifying a *shortest missing pair* as the adapter primers.

This paper is organized as follows: in Section 2 we give the definitions and problem reviews. In Section 3 we propose a new method for solving the missing pattern problem in time $O(n)$, then we present how the method can be extended to solve the missing pattern pair problem in time $O(\min\{\alpha n \log n, n^{3/2}\})$ in Section 4. In Section 5, an approach with running time $O(\frac{n^2}{\alpha} \log^2 n)$ is proposed, which has a complexity of $O(n \log^2 n)$ when $\alpha = \Theta(n)$. Section 6 concludes this paper.

2 Preliminaries

We try to follow as much as possible notations from other literature. The symbol N denotes the set of natural numbers. The symbol N^+ denotes the set of positive natural numbers.

Let Σ be a finite alphabet of size σ (σ is assumed to be a constant in (Inenaga et al. 2004) and in this paper¹). A *word* or a *pattern* is a string of symbols over Σ , where the latter is more typically used to refer to a substring of some (longer) word. The length of a word S is denoted as $|S|$. The character at position i of a string S is written $S[i]$ (the starting position being 0); while the substring of S from position i to position j ($i \leq j$) is written as $S[i : j]$.

A pattern P of length k is said to *occur* at position j of a string S if and only if $j + k - 1 < n$ and $P = S[j : j + k - 1]$. The set of all the positions that a pattern P occurs in a string S is denoted $Occ(P, S)$. For example, $Occ(AA, "AAATGCTAA") = \{0, 1, 7\}$. A pattern P is a *single missing pattern* (SMP) w.r.t a string S over Σ if it does not occur at any position of S , that is $Occ(P, S) = \emptyset$.

2.1 Sequence of k -mers

A k -mer is a word of length k . For any $k \in N^+$, we let $\Sigma^{=k}$ denote the set of all the words over Σ of length k , that is, all the k -mers.

Given a string S and $k \in N^+$, we can construct a sequence of $|S| - k + 1$ k -mers by the positions which they occur, namely, $S[0 : k - 1], S[1 :$

Copyright copyright 2006, Australian Computer Society, Inc. This paper appeared at Computing: The Australasian Theory Symposium (CATS2006), Hobart, Australia. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 51. Barry Jay and Joachim Gudmundsson, Eds. Reproduction for academic, not-for profit purposes permitted provided this text is included.

¹ Σ is assumed to be $\{A, C, G, T\}$ in the original application of missing pattern.

$k], \dots, S[n - k, n - 1]$ (k -mers in the sequence may not be distinct). For any string S and $k \in N^+$, we let \mathcal{SEQ}_S^k denote this corresponding sequence of k -mers. The i -th k -mer in the sequence \mathcal{SEQ}_S^k is written $\mathcal{SEQ}_S^k[i]$ (the first k -mer being $\mathcal{SEQ}_S^k[0]$). The subsequence of \mathcal{SEQ}_S^k from position i to j is written as $\mathcal{SEQ}_S^k[i : j]$.

We let each character in Σ be represented by a number from 0 to $\sigma - 1$. Each k -mer, say s , can then be written as a string of numbers $i_0 i_1 \dots i_{k-1}$ where each number i_j in the string has a value from 0 to $\sigma - 1$ which represents the character at $s[j]$. For example, the 8-mer “AAATATGG” may be written as “00020211” (or simply 20211), where 0 represents A , 1 represents G , and 2 represents T . In this way, for any fixed k , there is a 1-1 mapping from $\Sigma^{=k}$ to the natural numbers from 0 to $\sigma^k - 1$. This representation of k -mers is useful as indices to arrays or lookup tables. This coding is simplified version of the hashing idea from (Karp & Rabin 1987), here we use a 1-1 mapping, as σ is assumed to be a constant.

Unless stated otherwise, for any $k \in N^+$ and string S , we assume members in $\Sigma^{=k}$ and \mathcal{SEQ}_S^k to be of this natural number form. Assuming that k is small ($k \leq \log_\sigma n + 1$, the same assumption is made in (Inenaga et al. 2004) and a suffix tree is used, in which each integer has $O(\log n)$ bits), each k -mer $v \in \Sigma^{=k}$ uses only $O(1)$ space. We shall now show that given any S and k , the computation of \mathcal{SEQ}_S^k in such a representation can be performed very efficiently.

Given any S and k , we first compute $\mathcal{SEQ}_S^k[0]$, then for any $i > 0$, $\mathcal{SEQ}_S^k[i]$ can be computed with the value of the k -mer at position $i - 1$ by the formula:

$$\begin{aligned} \mathcal{SEQ}_S^k[i] &= \sigma(\mathcal{SEQ}_S^k[i - 1]) - \lfloor \frac{\mathcal{SEQ}_S^k[i - 1]}{\sigma^{k-1}} \rfloor \sigma^{k-1} \\ &+ S[i + k - 1] \end{aligned}$$

Thus we have the following result.

Lemma 1 For any string S and $k \in N^+$, \mathcal{SEQ}_S^k can be computed in $O(|S|)$ time.

Similarly, \mathcal{SEQ}_S^{k-1} can be efficiently computed from \mathcal{SEQ}_S^k , by the following.

$$\mathcal{SEQ}_S^{k-1}[i - 1] = \lfloor \mathcal{SEQ}_S^k[i - 1] / \sigma \rfloor \quad (1)$$

We let $\text{content}(\mathcal{SEQ}_S^k[i : j])$ denote the set of all k -mers which appears in the subsequence of k -mers $\mathcal{SEQ}_S^k[i : j]$. For simplicity, $\text{content}(\mathcal{SEQ}_S^k[0 : |\mathcal{SEQ}_S^k| - 1])$ is written as $\text{content}(\mathcal{SEQ}_S^k)$. Let $\mathcal{SEQ}_S^k(i, d) = \mathcal{SEQ}_S^k[A : B]$ where $A = \max\{0, i - d + 1\}$ and $B = \min\{|S| - 1, i + d - 1\}$. Intuitively, $\mathcal{SEQ}_S^k(i, d)$ are the k -mers of up to a distance d from the position i . $\text{content}(\mathcal{SEQ}_S^k(i, d))$ are the distinct k -mers of up to a distance d from the position i .

Lemma 2 $v \in \text{content}(\mathcal{SEQ}_S^{k-1})$ if and only if there exists $v' \in \text{content}(\mathcal{SEQ}_S^k)$ such that $v = \lfloor v' / \delta \rfloor$, or $v = \mathcal{SEQ}_S^{k-1}[|S| - k + 1]$.

Lemma 2 and Equation 1 together show that $\text{content}(\mathcal{SEQ}_S^{k-1})$ can be computed completely from $\text{content}(\mathcal{SEQ}_S^k)$ in time $O(|\text{content}(\mathcal{SEQ}_S^k)|)$.

3 Shortest Missing Pattern Problem

The shortest missing pattern problem, proposed in (Inenaga et al. 2004), is to find the shortest single missing pattern (SMP) w.r.t. a given string S . (Inenaga et al. 2004) has proposed a solution based on suffix trees which runs in time $O(|S|)$ and space $O(|S|)$. It is well known that the suffix tree has a large overhead and is difficult to implement. We propose an alternative here which uses the same order of time and space, but finds all the shortest SMPs w.r.t. to a string S and easy to implement. Below we reproduce a Lemma from (Inenaga et al. 2004) which we need to show our result.

Lemma 3 For any string S , there exists an SMP of length $\lceil \log_\sigma(|S| + 1) \rceil$ w.r.t. S .

Proof: For any k , there are a total of σ^k possible k -mers (that is, $|\Sigma^{=k}| = \sigma^k$). A string S , on the other hand, has at most distinct $|S| - k + 1$ k -mers. If $\sigma^k > |S| - k + 1$, then there is a k -mer which does not occur in S . Hence for any $k \geq \lceil \log_\sigma(|S| + 1) \rceil$, there exists a k -mer which does not occur in S . ■

Below we list the algorithm which finds the shortest SMPs. The algorithm conducts an exhaustive search of all the k -mers, with decreasing values of k , beginning from $\lceil \log_\sigma(|S| + 1) \rceil$. The output is a number k of the shortest length of the missing patterns, and an array of bits \mathcal{B} where for each $v \in \Sigma^{=k}$, $\mathcal{B}[v] = 1$ if and only if v is a SMP.

Algorithm 1: Find all shortest SMPs

1. Let $l = \lceil \log_\sigma(|S| + 1) \rceil$.
2. Compute $\text{content}(\mathcal{SEQ}_S^l)$.
3. For $k = l$ to 1,
4. Allocate an array \mathcal{B} of σ^k bits, initializing each bit to 0.
5. For each $v \in \text{content}(\mathcal{SEQ}_S^k)$, set $\mathcal{B}[v]$ to 1.
6. (Note that elements in $\text{content}(\mathcal{SEQ}_S^k)$ range from 0 to $\sigma^k - 1$.)
7. If all the bits in \mathcal{B} are set to 1 (in which case there is no missing pattern of length k or below), output $k + 1$ and \mathcal{B}' (i.e. SMPs found in the previous iteration). (Note: by Lemma 3 there is at least one SMP at iteration $k = l$.)
9. Let $\mathcal{B}' = \mathcal{B}$, and compute $\text{content}(\mathcal{SEQ}_S^{k-1})$ from $\text{content}(\mathcal{SEQ}_S^k)$ (using Equation 1 and Lemma 2).
10. End

Since $\text{content}(\mathcal{SEQ}_S^{k-1})$, \mathcal{B} and \mathcal{B}' are of size σ^k for the k -th iteration, the space requirement is $O(\sigma^l) = O(|S|)$. By Lemma 1, line-2 can be computed in time $O(|S|)$. There are a total of at most l iterations for the loop at line-3, where each iteration takes time $O(\sigma^k)$. Thus the total time is $O(\sum_{k=1}^l \sigma^k)$. Since $\sum_{k=1}^l \sigma^k = \sigma^l + \sum_{k=1}^{l-1} \sigma^k = \sigma^l + \frac{\sigma(\sigma^{l-1} - 1)}{\sigma - 1} \leq 2\sigma^l$, the time complexity is $O(\sigma^l) = O(|S|)$.

Theorem 4 Given any string S , Algorithm 1 finds all the shortest SMP w.r.t S in $O(|S|)$ time, using $O(|S|)$ space.

4 Missing Pattern Pair Problem

A missing pattern pair (MPP) P_1 and P_2 with threshold α (written $\langle P_1, P_2 \rangle_\alpha$) w.r.t. a string S is a pair of patterns where:

1. either P_1 or P_2 is an SMP w.r.t S ; or
2. both P_1 and P_2 occur in S , and $\forall p_1 \in \text{Occ}(P_1, S)$, $\forall p_2 \in \text{Occ}(P_2, S)$, $|p_1 - p_2| > \alpha$. That is, no occurrences of P_1 occur within a distance of α from P_2 (and vice versa).

Our aim is to find a missing pattern pair with the shortest total length.

| | |
|---|--|
| MISSING PATTERN PAIR (MPP) PROBLEM (INENAGA ET AL. 2004) | |
| Input: | String S and $\alpha \in N^+$. |
| Output: | An MPP $\langle P_1, P_2 \rangle_\alpha$ w.r.t. S with minimal $ P_1 + P_2 $. |

An algorithm for the MMP Problem with time complexity $O(\min\{\alpha n \log n, n^2\})$ was given in (Inenaga et al. 2004).

4.1 Preliminaries

We first introduce the *Dynamic Perfect Hashing* data structure which will be used in our algorithm.

The *Dynamic Perfect Hashing* (Dietzfelbinger, Karlin, Mehlhorn & Der 1994) is a data structure which manages a dictionary (a set of key-data pair) with $O(1)$ amortized runtime cost in the following operations: *insert*(k), *delete*(k), and *getdata*(k), where k is the key used in the operation. Its space requirement is linearly proportional to the number of elements managed. We write $k \in \mathcal{H}$ just in case k is the key for a key-data pair in \mathcal{H} ; $\mathcal{H}[k]$ denotes the data part of the key-data pair in \mathcal{H} with key k .

4.2 Finding Missing Pattern Pair

This is our strategy for solving the MMP problem. We first run Algorithm 1 on the input string S . If it returns an SMP of length ℓ (note that by Lemma 3 Algorithm 1 must return some $\ell \in N^+$), we know:

1. any shortest MPP $\langle P_1, P_2 \rangle_\alpha$ must have $|P_1| + |P_2| \leq \ell$.
2. for any shortest MPP $\langle P_1, P_2 \rangle_\alpha$, if $|P_1| + |P_2| < \ell$, then both P_1 and P_2 occurs in S .

Based on the output of Algorithm 1, we then exhaustively search for all MPPs $\langle P_1, P_2 \rangle_\alpha$ of total length ℓ , $\ell - 1, \dots, 1$.

We first give a subroutine (Algorithm 2) that looks for MPPs $\langle P_1, P_2 \rangle$ where the lengths of P_1 and P_2 are fixed to, say, l_1 and l_2 respectively. Without loss of generality we let $l_1 \geq l_2$. By the argument above we assume that P_1 and P_2 both appear in the given string.

The subroutine first computes the sequences of k -mers $\mathcal{SEQ}_S^{l_1}$ and $\mathcal{SEQ}_S^{l_2}$. It then computes, as output, a $\sigma^{l_1} \times \sigma^{l_2}$ matrix \mathcal{BB}_{l_1, l_2} where for all $u \in \Sigma^{=l_1}$ and $v \in \Sigma^{=l_2}$, $\mathcal{BB}_{l_1, l_2}[u][v] = 1$ iff there exists some $i, j \in N$ such that $\mathcal{SEQ}_S^{l_1}[i] = u$ and $\mathcal{SEQ}_S^{l_2}[j] = v$ and $|i - j| \leq \alpha$. That is, $\langle u, v \rangle_\alpha$ is an MPP just in case $\mathcal{BB}_{l_1, l_2}[u][v] = 1$. For notation simplicity $\mathcal{BB}_{l_1, l_2}[u]$ refers to the array entries $\mathcal{BB}_{l_1, l_2}[u][v]$ with $0 \leq v \leq \sigma^{l_2} - 1$. When it is clear from the context, \mathcal{BB}_{l_1, l_2} is written as \mathcal{BB} .

Algorithm 2: Find all MPPs $\langle P_1, P_2 \rangle$

where $|P_1| = l_1$ and $|P_2| = l_2$

1. Compute $\mathcal{SEQ}_S^{l_1}$, $\mathcal{SEQ}_S^{l_2}$, $\text{content}(\mathcal{SEQ}_S^{l_2}(0, \alpha))$.
2. Allocate a $\sigma^{l_1} \times \sigma^{l_2}$ 1-bit matrix \mathcal{BB} , initialize each bit in \mathcal{BB} to 0.

3. Prepare dictionary \mathcal{H} .
4. For each $u \in \text{content}(\mathcal{SEQ}_S^{l_2}(0, \alpha))$
5. Let $\mathcal{H}[u]$ be the number of occurrences of u in $\mathcal{SEQ}_S^{l_2}(0, \alpha)$.
6. For $i = 0, 1, \dots, |\mathcal{SEQ}_S^{l_1}| - 1$,
7. Let $u = \mathcal{SEQ}_S^{l_1}[i]$,
8. For each key $v \in \mathcal{H}$
(Note: $v \in \mathcal{H} \Rightarrow v \in \text{content}(\mathcal{SEQ}_S^{l_2}(i, \alpha))$.)
9. Let $\mathcal{BB}[u][v] = 1$.
10. The following updates \mathcal{H} so that $v \in \mathcal{H} \Rightarrow v \in \text{content}(\mathcal{SEQ}_S^{l_2}(i + 1, \alpha))$.
11. If $i - \alpha \geq 0$
12. Let $v = \mathcal{SEQ}_S^{l_2}[i - \alpha]$.
13. Decrease $\mathcal{H}[v]$ by 1.
If $\mathcal{H}[v] = 0$ remove v' from \mathcal{H} .
14. If $i + \alpha + 1 < |\mathcal{SEQ}_S^{l_1}|$
15. Let $v = \mathcal{SEQ}_S^{l_2}[i + \alpha + 1]$.
16. If $v \notin \mathcal{H}$, let $\mathcal{H}[v] = 1$,
else increment $\mathcal{H}[v]$ by 1.

In Algorithm 2, line 1 runs in time $O(|S|)$; while line 2–3 takes $O(|\mathcal{BB}|)$ time, that is, $O(\sigma^{l_1} \cdot \sigma^{l_2}) = O(\sigma^\ell)$. Line 4–5 runs in $O(\alpha)$ time. There a total of $|S|$ iterations for the loop at line 6. At each iteration i , \mathcal{H} contains at most $\min\{2\alpha + 1, \sigma^{l_2}\}$ distinct entries. Thus line 8–9 runs in $O(\min\{2\alpha + 1, \sigma^{l_2}\})$ per iteration. Line 10–16 runs in constant time for each iteration. Algorithm 2 hence runs in time $O(|S| \min\{2\alpha + 1, \sigma^{l_2}\})$.

Lemma 5 Given string S , $l_1, l_2 \in N^+$ where $l_1 \geq l_2$, Algorithm 2 takes time $O(\min\{|S|\sigma^{l_2}, |S|\alpha\})$ to find all the MPPs $\langle P_1, P_2 \rangle_\alpha$ w.r.t. S where $|P_1| = l_1$ and $|P_2| = l_2$.

4.3 Identify Missing Pattern Pair with Minimum Length

Firstly the following lemma can be deduced:

Lemma 6 Given \mathcal{BB}_{l_1, l_2} , $\mathcal{BB}_{l_1, l_2 - 1}$ can be computed with time $O(\sigma^{l_1 + l_2} + \alpha)$.

Proof: There are two cases that $\mathcal{BB}_{l_1, l_2 - 1}[u][v] = 1$:

1. There exists v' such that $\mathcal{BB}_{l_1, l_2}[u][v'] = 1$ and with $v = \lfloor v'/\sigma \rfloor$
2. $u \in \text{content}(\mathcal{SEQ}_S^{l_1}(n - l_2 + 1, \alpha))$, and $v = \mathcal{SEQ}_S^{l_2 - 1}[n - l_2 + 1]$.

Totally there are $\sigma^{l_1 + l_2}$ entries in \mathcal{BB}_{l_1, l_2} and we just need to scan through all the entries of \mathcal{BB}_{l_1, l_2} (which takes time cost σ^l) and to obtain $\text{content}(\mathcal{SEQ}_S^k, n - l' + 1, \alpha)$ (which takes time $O(\alpha)$) to compute $\mathcal{BB}_{l_1, l_2 - 1}$. ■

With these, we are now ready to wrap everything to obtain a better algorithm. Firstly we use the algorithm for Section 3.1 to identify ℓ . Then we iterate through all the possible combinations of (l_1, l_2) ($l_1 + l_2 \leq \ell$) to obtain all the shortest MPPs.

As the two cases for (l_1, l_2) and (l_2, l_1) are symmetric, we just need to search the cases where $l_1 \geq l_2$. Denote $\delta(l) = \min\{l_1, \ell - l_1\}$. The pseudo code is presented in Algorithm 3. For each l_1 , the algorithm will search through all the possible values l_2 (while avoiding the symmetric cases) by using the result from Lemma 6 to avoid recomputation. Algorithm 2 will be employed for the combination $(l_1, \delta(l_1))$ (given l_1 , the largest possible value for l_2 is $\delta(l_1)$).

Algorithm 3: Find the Shortest MPPs

1. Identify the shortest missing pattern length ℓ with Algorithm 1.
2. Compute $\text{content}(\mathcal{SEQ}_S^\ell)$.
3. For $l_1 = \ell - 1$ to 1
4. Compute $\mathcal{BB}_{l_1, \delta(l_1)}$ with Algorithm 2.
5. For $l_2 = \delta(l_1) - 1$ to 1.
6. Compute \mathcal{BB}_{l_1, l_2} with Lemma 6
7. Record if there is a length $l_1 + l_2$ missing pattern is found.

The time cost is dominated by line 3-7 for Algorithm 3. For each value of l_1 , line 4 takes time $O(|S| \min\{\sigma^{\delta(l_1)}, \alpha\})$, and line 5-6 takes time $O(\sum_{l_2=1}^{\delta(l_1)-1} (\sigma^{l_2} + \alpha))$.

$$\begin{aligned} \sum_{l_2=1}^{\delta(l_1)-1} (\sigma^{l_2} + \alpha) &\leq \sigma^{\delta(l_2)} \frac{1}{\sigma - 1} + (\delta(l_1) - 1)\alpha \\ &= O(\sigma^{\delta(l_1)} + (\delta(l_1) - 1)\alpha) \end{aligned}$$

As $l_1 < \ell = O(\log n)$, which means $\sigma^{\delta(l_1)} = O(|S|)$, also we know that $\delta(l_1) = O(\sigma^{\delta(l_1)})$ and $\alpha \leq |S|$. Combine all these in, we have:

$$\begin{aligned} |S| \min\{\sigma^{\delta(l_1)}, \alpha\} + \sigma^{\delta(l_1)} + (\delta(l_1) - 1)\alpha \\ &= O(|S| \min\{\sigma^{\delta(l_1)}, \alpha\} + |S| + \delta(l_1)\alpha) \\ &= O(\min\{|S|\sigma^{\delta(l_1)} + \delta(l_1)\alpha, |S|\alpha + \delta(l_1)\alpha\}) \\ &= O(|S| \min\{\sigma^{\delta(l_1)}, \alpha\}) \end{aligned}$$

Thus for each iteration of the outer loop, line 4-6 takes time $O(|S| \min\{\sigma^{\delta(l_1)}, \alpha\})$. Lastly, sum up the terms over the possible l_1 values, we have:

$$\begin{aligned} \sum_{l_1 \leq \ell} |S| \min\{\sigma^{\delta(l_1)}, \alpha\} &\leq |S| \min\left\{\sum_{l_1 \leq \ell} \sigma^{\delta(l_1)}, \sum_{l_1 \leq \ell} \alpha\right\} \\ &\leq |S| \min\left\{\sum_{l_1 \leq \ell} \sigma^{\min\{l_1, \ell - l_1\}}, \ell\alpha\right\} \\ &\leq |S| \min\left\{\sum_{l_1 \leq \lceil \frac{\ell}{2} \rceil} 2\sigma^{l_1}, \ell\alpha\right\} \\ &= O(|S| \min\{\sigma^{\lceil \frac{\ell}{2} \rceil}, \ell\alpha\}) \\ &= O(|S| \min\{\sqrt{|S|}, \alpha \log n\}) \end{aligned}$$

For the space complexity, it is clear that it is $O(|S| + \Delta)$, where Δ represents the output size. If we just want to identify one shortest MPP, the space complexity is $O(|S|)$. Formally, the time and space complexity are concluded in Theorem 7.

Theorem 7 *The missing pattern pair problem with a given string S of length n and a threshold α can be solved with time complexity $O(\min\{n^{3/2}, n\alpha \log n\})$ and space complexity $O(n)$ with Algorithm 3.*

5 Faster Algorithm with Large α

It may be noticed if we can replace Algorithm 2 in Algorithm 3 with a faster subroutine, Algorithm 3 will result in less running time. In this section, a faster procedure for large α is presented. A simple data structure named *Range Union Tree* is defined to serve the purpose of this algorithm.

5.1 Range Union Tree

For k -mer sequence \mathcal{SEQ}_S^k , a $\text{report}(i, j)$ query reports the set of distinct elements of the subsequence $\mathcal{SEQ}_S^k[i : j]$, that is $\text{content}(\mathcal{SEQ}_S^k[i : j])$. At the first glance, an array representation of \mathcal{SEQ}_S^k will be good enough to handle the queries. However this approach will not be efficient when the report queries are numerous and there are high duplications for the elements of \mathcal{SEQ}_S^k . To serve the usage of this paper, a balanced binary tree is adopted to represent \mathcal{SEQ}_S^k . An example is illustrated in Figure 1. Each element of \mathcal{SEQ}_S^k is assigned to a leaf node. For each internal node v , we store an ordered list of the distinct integers contains in the leaf node of the subtree which is rooted at v . The smallest and largest indices in \mathcal{SEQ}_S^k under each subtree is also maintained at each internal node respectively. The space usage for this tree is $O(n \log n)$ since each level of the tree requires space $O(n)$. To construct the tree, time cost $O(n \log n)$ is enough by a bottom-up manner (which is similar as the merge sort). For a report query, it is easy to see that we just need to union $O(\log n)$ ordered lists, which can be accomplished with time complexity $O(\sigma^k \log n)$, as σ^k is the upper bound of result list size. This tree is referred as the *Range Union Tree* (RUT) in this paper.

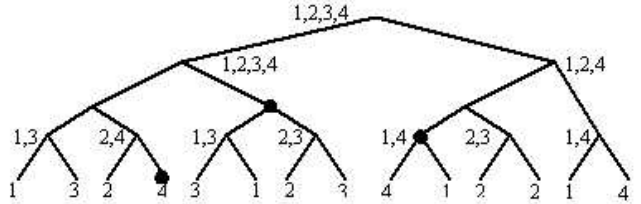


Figure 1: A RUT for sequence $\mathcal{SEQ}_S^k = \{1, 3, 2, 4, 3, 1, 2, 3, 4, 1, 3, 2, 1, 4\}$. To answer the query $\text{report}(3, 9)$ the element between $\mathcal{SEQ}_S^k[3]$ to $\mathcal{SEQ}_S^k[9]$, we just need to compute the union of these sets associated with the shaded nodes.

5.2 An Algorithm Based on RUT

Denote $\mathcal{SEQ}_S^l(Q, \alpha) = \bigcup_{i \in Q} \mathcal{SEQ}_S^l(i, \alpha)$. Let $Q_u^l = \text{Occ}(u, \mathcal{SEQ}_S^l)$. The task to compute $\mathcal{BB}[u]$ is essentially equivalent to compute $\text{content}(\mathcal{SEQ}_S^l(Q_u^l, \alpha))$. $\mathcal{SEQ}_S^l(Q_u^l, \alpha)$ consists a set of disjoint intervals (an interval here means a consecutive subsequence of indices eg. 1, 2, 3, 4, 7, 8, 9 is considered as two maximal disjointed intervals [1, 4], [7, 9]). The set of maximal disjointed intervals can be stored as an ordered list for each u and can be computed for all the u values simultaneously with time cost of $O(n)$ by scanning \mathcal{SEQ}_S^l once. An array \mathcal{A} with size σ^{l_1} with each entry indexed with u , $0 \leq u \leq \sigma^{l_1} - 1$ may be employed. The array entry at position u records the last interval's end index for u for the current scanning. While scanning the l_1 -mer with $u = \mathcal{SEQ}_S^l[i]$, by comparing $\max\{0, i - \alpha\}$ with $\mathcal{A}[i]$, we will know whether a new interval should be open, or the last interval should be just extended for u .

To compute $\text{content}(\mathcal{SEQ}_S^l(Q_u^l, \alpha))$ (we extend the content notation to a set of intervals), the set of intervals contained in $\mathcal{SEQ}_S^l(Q_u^l, \alpha)$ can be computed first, then with the RUT, the content of each interval can be obtained. Lastly the union of the content of the intervals can be identified. The number of

disjointed intervals for each u is bounded by $|S|/\alpha + c$ (from some constant c). For each u , we need query the RUT at most $|S|/\alpha + c$ times. Each query will cost time $O(\sigma^{l_2} \log n)$ and the time cost for each u , $0 \leq u \leq \sigma^{l_1}$ is $O(|S|/\alpha \sigma^{l_2} \log n)$. Thus this approach for computing \mathcal{BB} will result in an method with time complexity $O(\sigma^{l_1} |S|/\alpha \sigma^{l_2} \log n) = O(\frac{n^2}{\alpha} \log n)$.

Substitute it into Algorithm 3, we have:

Theorem 8 *Based on RUT, the missing pattern pair problem can be solved with time complexity of $O(\frac{n^2}{\alpha} \log^2 n)$*

6 Conclusion

In this paper, we proposed two deterministic algorithms for the missing pattern problem and have improved the bound for the MPP problem from $O(n^2)$ to $O(n^{3/2})$. Also we have demonstrated with a faster subroutine for a MPP for given lengths, a faster algorithm can be obtained under our framework.

References

- Dietzfelbinger, M., Karlin, A., Mehlhorn, K. & Der, F. M. (1994), 'Dynamic perfect hashing: Upper and lower bounds', *SIAM J. Comput.* **23**(4), 738–761.
- Inenaga, S., Kivioja, T. & Mäkinen, V. (2004), Finding missing patterns, *in* 'WABI', pp. 463–474.
- Karp, R. M. & Rabin, M. O. (1987), 'Efficient randomized pattern-matching algorithms', *IBM J. Res. Dev.* **31**(2), 249–260.
- Wang, J. T.-L., Shapiro, B. A. & Shasha, D., eds (1999), *Pattern Discovery in Biomolecular Data: Tools, Techniques and Applications*, Oxford University Press.