Bit-Parallel Approach to Approximate String Matching in Compressed Texts

Tetsuya Matsumoto Takuya Kida Masayuki Takeda Ayumi Shinohara
Setsuo Arikawa

Department of Informatics, Kyushu University 33, Fukuoka 812-8581, Japan
{tetsuya, kida, takeda, ayumi, arikawa}@i.kyushu-u.ac.jp

Abstract

In this paper, we address the problem of approximate string matching on compressed text. We consider this problem for a text string described in terms of collage system, *which is a formal system proposed by Kida et al. (1999) that captures various dictionary-based compression methods. We present an algorithm that exploits bit-parallelism, assuming that our problem fits in a single machine word, e.g.,* $(m-k)(k+1) \leq L$*, where* m *is the pattern length,* k *is the number of allowed errors, and* L *is the length in bits of the machine word. For a class of* simple *collage systems, the algorithm runs in* $O(k^2(\|\mathcal{D}\| + |S|) + km)$ *time using* $O(k^2\|\mathcal{D}\|)$ *space, where* $\|\mathcal{D}\|$ *is the size of dictionary* $\mathcal{D}$ *and* $|\mathcal{S}|$ *is the number of tokens in* $\mathcal{S}$*. The LZ78 and the LZW compression methods are covered by this class. Since we can regard* $n = \|\mathcal{D}\| + |\mathcal{S}|$ *as the compressed length, the time and the space complexities are* $O(k^2 n + km)$ *and* $O(k^2 n)$*, respectively. For general* k *and* m*, they become* $O(k^3 mn/L + km)$ *and* $O(k^3 mn/L)$*. Thus, our algorithm is competitive to the algorithm proposed by Kärkkäinen, et al. (2000) which runs in* $O(kmn)$ *time using* $O(kmn)$ *space, when* $k = O(\sqrt{L})$*.*

1 Introduction

The problem of approximate string matching is to find the locations of approximate occurrences P' of a pattern P in a text T such that the edit distance between P and P' is $\leq k$, where the edit distance between two strings is the minimum number of edit operations (insertions, deletions, and substitutions) required to convert one string into the other. This problem has been studied extensively. See an excellent survey paper by Navarro [7].

Baeza-Yates and Navarro [2] presented a very fast algorithm for on-line approximate string matching, which is based on the simulation of a nondeterministic finite automaton (NFA) built from the pattern P and the number k of allowed errors. Figure 1 shows the NFA for the pattern $P =$ POKEMON with $k = 4$. The simulation algorithm exploits bit-parallelism. The idea is essentially similar to the work of Wu and Manber [13], but the search time is improved by packing the automaton states differently. That is, the algorithm simulates the NFA using diagonals instead of rows. Since there are $m-k+1$ complete diagonals and the others are not really necessary, we need only $(m-k)(k+2)$ bits for a pattern of length m. Thus the automaton states can be packed into a machine word for a short pattern, and the algorithm achieves an $O(N)$ search time, where N is the text length. Recall that the Wu-Manber algorithm uses $k+1$ machine words to encode the automaton states, and requires $O(kN)$ time.

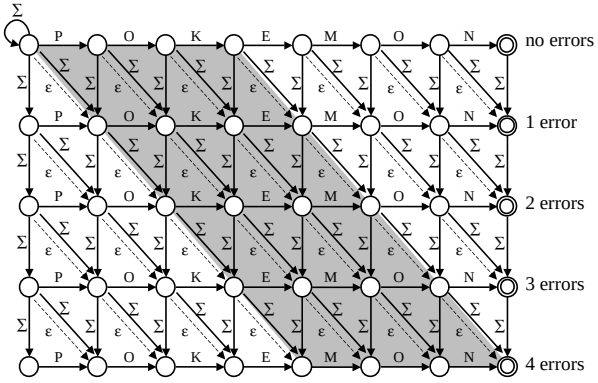

Figure 1. NFA for approximate string matching for the pattern POKEMON with $k = 4$.

On the other hand, the compressed pattern matching attracts recently special concern where the goal is to perform the exact string matching in a compressed text without decompressing it. The problem has been studied in 1990's by several researchers mainly for dictionary-based compression methods, such as the Ziv-Lempel family (e.g., LZ77 [14], LZ78 [15], LZW [12]). In [5], we introduced *collage systems*, a formal system to represent a text string, that cap-

tures various dictionary-based compressions. Within this framework, we generalized the existing compressed pattern matching algorithms and unified the concepts into a general algorithm. Thus *any* compression method covered by the framework has now a compressed pattern matching algorithm as an instance.

However, the studies on the compressed pattern matching have been undertaken only for the exact pattern matching. The approximate string matching on compressed text has been an open problem since advocated in [1]. The first solution to this problem was very recently given by Kärkkäinen et al. [4]. They addressed the LZ78 and the LZW compression methods. The proposed algorithm runs in $O(mkn+r)$ time, where n is the compressed text length, m is the pattern length, k is the number of allowed errors, and r is the number of pattern occurrences with k errors or less. For the existence problem it needs $O(mkn)$ time and space. The algorithm is based on the simulation of the classical dynamic programming algorithm for approximate string matching [10]. They also showed that the algorithm can be adapted to run in $O(k^2n + \min(kmn, m^2(m|\Sigma|)^k))$ time on the average. The experimental results reported in [4] show that, for small k and moderate m, the algorithm runs faster than the method of decompressing the text on the fly and searching over it using the modified dynamic programming method which runs in $O(kN)$ time on the average [11], where N is the original text length.

In this paper, we take the NFA simulation approach using bit-parallelism, instead of the dynamic programming approach. We present a new algorithm for approximate string matching to solve the existence problem. Like the work of Baeza-Yates and Navarro [2], we assume that our problem fits in a single machine word, i.e., $(m-k)(k+1) \leq L$, where L is the length in bits of the machine word. Our algorithm runs on a simple collage system $\langle \mathcal{D}, \mathcal{S} \rangle$ in $O(k^2(\|\mathcal{D}\| + |\mathcal{S}|) + km)$ time using $O(k^2\|\mathcal{D}\|)$ space. It should be emphasized that our algorithm works for *any* compression method that covered by the class of simple collage systems. For example, this class covers the LZ78 and the LZW compression methods. Since we can regard $n = \|\mathcal{D}\| + |\mathcal{S}|$, the time and the space complexities are $O(k^2n + km)$ and $O(k^2n)$, respectively. For general k and m, they become $O(k^3mn/L + km)$ and $O(k^3mn/L)$, respectively. Thus our algorithm is competitive to the one due to Kärkkäinen et al., when $k = O(\sqrt{L})$ which is quite reasonable.

2 Preliminaries

Let Σ be a finite alphabet. An element of Σ^* is called a *string*. Strings x, y, and z are said to be a *prefix*, *factor*, and *suffix* of the string $u = xyz$, respectively. The length of a string u is denoted by $|u|$. The empty string is denoted by ε, that is, $|\varepsilon| = 0$. The ith symbol of a string u is denoted by $u[i]$ for $1 \leq i \leq |u|$, and the factor of a string u that begins at position i and ends at position j is denoted by $u[i:j]$ for $1 \leq i \leq j \leq |u|$. For convenience, let $u[i:j] = \varepsilon$ for $j < i$. Denote by u^R the reversed string of a string u. For a string u and a non-negative integer i, the string obtained by removing the length i prefix (resp. suffix) from u is denoted by ${}^{[i]}u$ (resp. $u^{[i]}$). That is, ${}^{[i]}u = u[i+1:|u|]$ and $u^{[i]} = u[1:|u|-i]$. Denote by $D(x,y)$ the edit distance between two strings x and y.

3 Collage system

In a dictionary-based compression, a text string is described by a pair of a *dictionary* and a sequence of *tokens*, each of which represents a phrase defined in the dictionary. Kida et al. [5] introduced a unifying framework, named collage system, which abstracts various dictionary-based methods, such as the Lempel-Ziv family, the SEQUITUR [9], the RE-PAIR [6], and static dictionary methods. In [5] they presented a general compressed pattern matching algorithm for the framework, which is based on the simulation of the KMP automaton. This implies that *any* compression method covered by the framework has a compressed pattern matching algorithm as an instance.

A *collage system* is a pair $\langle \mathcal{D}, \mathcal{S} \rangle$ defined as follows: $\mathcal{D}$ is a sequence of assignments $X_1 = \mathit{expr}_1;\ X_2 = \mathit{expr}_2;\ \cdots;\ X_\ell = \mathit{expr}_\ell$, where each X_k is a token (or a variable) and expr_k is any of the form:

a	for $a \in \Sigma \cup \{\varepsilon\}$,	(*primitive assignment*)
X_iX_j	for $i, j < k$,	(*concatenation*)
${}^{[j]}X_i$	for $i < k$ and an integer j,	(*prefix truncation*)
$X_i^{[j]}$	for $i < k$ and an integer j,	(*suffix truncation*)
$(X_i)^j$	for $i < k$ and an integer j.	(j *times repetition*)

Each token represents a string obtained by evaluating the expression as it implies. The strings represented by tokens are called *phrases*. As we want to distinguish a token from the phrase it represents, we denote by $X.u$ the phrase represented by a token X. The *size* of $\mathcal{D}$ is the number ℓ of assignments and denoted by $\|\mathcal{D}\|$. Define the *height* of a token X to be the height of the syntax tree whose root is X. The *height* of $\mathcal{D}$ is defined by $\mathit{height}(\mathcal{D}) = \max\{\mathit{height}(X) \mid X \text{ in } \mathcal{D}\}$. It expresses the maximum dependency of the tokens in $\mathcal{D}$.

On the other hand, $\mathcal{S} = X_{i_1}, X_{i_2}, \ldots, X_{i_n}$ is a sequence of tokens defined in $\mathcal{D}$. We denote by $|\mathcal{S}|$ the number n of tokens in $\mathcal{S}$. The collage system represents a string obtained by concatenating the phrases represented by $X_{i_1}, X_{i_2}, \ldots, X_{i_n}$.

According to the framework, text strings compressed by LZW and RE-PAIR can be represented as follows.

LZW. $\mathcal{S} = X_{i_1}, X_{i_2}, \ldots, X_{i_n}$ and $\mathcal{D}$ is as follows:

$$X_1 = a_1; \quad X_2 = a_2; \quad \cdots; \quad X_q = a_q;$$
$$X_{q+1} = X_{i_1}X_{\sigma(i_2)}; \quad X_{q+2} = X_{i_2}X_{\sigma(i_3)}; \quad \cdots;$$
$$X_{q+n-1} = X_{i_{n-1}}X_{\sigma(i_n)},$$

where the alphabet is $\Sigma = \{a_1, \ldots, a_q\}$, $1 \leq i_1 \leq q$, and $\sigma(\ell)$ denotes the integer k, $1 \leq k \leq q$, such that a_k is the first symbol of the phrase $X_\ell.u$. $\mathcal{S}$ is encoded as a sequence of integers $i_1, i_2, \ldots, i_n$ in which an integer i_j is represented in $\lceil \log_2(q+j) \rceil$ bits, while $\mathcal{D}$ is not encoded since it can be obtained from $\mathcal{S}$.

RE-PAIR. $\mathcal{S} = X_{i_1}, X_{i_2}, \ldots, X_{i_n}$, and $\mathcal{D}$ is as follows:

$$X_1 = a_1; \quad X_2 = a_2; \quad \cdots; \quad X_q = a_q;$$
$$X_{q+1} = X_{\ell(1)}X_{r(1)}; \quad X_{q+2} = X_{\ell(2)}X_{r(2)}; \cdots;$$
$$X_{q+s} = X_{\ell(s)}X_{r(s)},$$

where $\Sigma = \{a_1, \cdots, a_q\}$. $\mathcal{D}$ and $\mathcal{S}$ are encoded using some appropriate encoding.

Theorem 1 (Kida et al. [5]) *The problem of pattern matching for a text $\langle \mathcal{D}, \mathcal{S} \rangle$ can be solved in $O(\mathit{height}(\mathcal{D})(\|\mathcal{D}\|+|\mathcal{S}|)+m^2+r)$ time using $O(\|\mathcal{D}\|+m^2)$ space, where r is the number of pattern occurrences. The factor $\mathit{height}(\mathcal{D})$ can be dropped if $\mathcal{D}$ contains no truncation.*

Since the factor $\|\mathcal{D}\| + |\mathcal{S}|$ is considered to be the compressed text length, the algorithm runs in linear time proportional to the compressed text length if $\mathcal{D}$ contains no truncation. The result coincides with the observation by Navarro and Raffinot [8] that LZ77 is not suitable for compressed pattern matching compared with LZ78 compression.

The following definitions will be needed for Sections 4 and 5.

Definition 1 *A collage system is said to be* regular *if it contains neither repetition nor truncation. A regular collage system is said to be* simple *if, for every assignment $X = YZ$, $|Y.u| = 1$ or $|Z.u| = 1$.*

Note that the collage systems for the SEQUITUR and the RE-PAIR are regular, and those for the LZW/LZ78 compressions are simple.

4 Basic idea

We give an overview of our algorithm for approximate string matching, which is based on the simulation of the NFA. First we give a formal description of the NFA for approximate string matching of pattern $P = P[1:m]$ with k errors. Denote by $\boldsymbol{N}$ the set of natural numbers $0, 1, \ldots$. For any $S \subseteq \boldsymbol{N} \times \boldsymbol{N}$ and any $\langle x, y \rangle \in \boldsymbol{N} \times \boldsymbol{N}$, let $S \oplus \langle x, y \rangle = \{\langle i+x, j+y \rangle \mid \langle i, j \rangle \in S\}$. Let

$$Q = \{\langle i, j \rangle \mid 0 \leq i \leq m, 0 \leq j \leq k\}.$$

Define the mapping $\delta^{\mathrm{ed}} : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$ by

$$\begin{aligned}
\delta^{\mathrm{ed}}(\langle i, j \rangle, \varepsilon) &= \{\langle i+1, j+1 \rangle \mid i+1 \leq m, j+1 \leq k\}, \\
\delta^{\mathrm{ed}}(\langle i, j \rangle, a) &= \{\langle i+1, j \rangle \mid i+1 \leq m, P[i+1] = a\} \\
&\quad \cup \{\langle i, j+1 \rangle \mid j+1 \leq k\} \\
&\quad \cup \{\langle i+1, j+1 \rangle \mid i+1 \leq m, j+1 \leq k\},
\end{aligned}$$

where $\langle i, j \rangle \in Q$ and $a \in \Sigma$. Define the mapping $\delta^{\mathrm{pm}} : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$ by

$$\begin{aligned}
\delta^{\mathrm{pm}}(\langle i, j \rangle, \varepsilon) &= \delta^{\mathrm{ed}}(\langle i, j \rangle, \varepsilon), \\
\delta^{\mathrm{pm}}(\langle i, j \rangle, a) &= \delta^{\mathrm{ed}}(\langle i, j \rangle, a) \cup \{\langle 0, 0 \rangle \mid i = j = 0\},
\end{aligned}$$

where $\langle i, j \rangle \in Q$ and $a \in \Sigma$. Let $F = \{\langle m, j \rangle \mid 0 \leq j \leq k\}$.

Proposition 1 *The NFA specified by the quintuple $(Q, \Sigma, \delta^{\mathrm{ed}}, \{\langle 0, 0 \rangle\}, F)$ accepts the language $\{w \in \Sigma^* \mid D(P, w) \leq k\}$.*

Proposition 2 *The NFA specified by the quintuple $(Q, \Sigma, \delta^{\mathrm{pm}}, \{\langle 0, 0 \rangle\}, F)$ accepts the language $\Sigma^*\{w \in \Sigma^* \mid D(P, w) \leq k\}$.*

Let $\hat{\delta}^{\mathrm{ed}}$ and $\hat{\delta}^{\mathrm{pm}}$, respectively, be the functions δ^{ed} and δ^{pm} extended over the domain $2^Q \times \Sigma^*$ in the standard way (see Section 2.4 of [3]).

Definition 2 *Define the function Jump $: 2^Q \times \mathcal{D} \to 2^Q$ by*

$$\mathit{Jump}(S, t) = \hat{\delta}^{\mathrm{pm}}(S, t.u).$$

Define the function Output $: 2^Q \times \mathcal{D} \to \{\mathit{true}, \mathit{false}\}$ by

$\mathit{Output}(S, t) = \mathit{true}$
$\Leftrightarrow$ *there exists a nonempty prefix w of $t.u$ such that $\hat{\delta}^{\mathrm{pm}}(S, w) \cap F \neq \emptyset$.*

The basic idea of our algorithm is to simulate the NFA in compressed text, by packing $S \in 2^Q$ in a machine word. We need only $(m-k+1)(k+1)$ states out of Q since there are $m - k + 1$ complete diagonals and the others are not necessary as pointed out by Baeza-Yates and Navarro [2]. That is, we need only the states in

$$Q_{\mathrm{core}} = \{\langle i, j \rangle \in Q \mid 0 \leq i \leq m-k+1,\ 0 \leq j \leq k\}.$$

Moreover, we know that the states on the first diagonal, i.e., the states $\langle i, j \rangle \in Q_{\mathrm{core}}$ with $i = j$ are always active, and therefore we do not have to store them. Thus we need only $(m-k)(k+1)$ bits to represent the NFA states.

Assume that the given pattern $P = P[1:m]$ is small so that $(m-k)(k+1)$ is not greater than the machine word length in bits. Then the union, the intersection, and the $\oplus$ operations on the subsets of $Q_{\rm core}$ can be performed in $O(1)$ time. As shown later, we can build the functions *Jump* and *Output* in $O(k^2\|\mathcal{D}\| + km)$ time using $O(k^2\|\mathcal{D}\|)$ space, so that they return their values in $O(k^2)$ time, for a simple collage system $\langle \mathcal{D}, \mathcal{S}\rangle$. Figure 2 gives an overview of the algorithm using the function *Jump* and *Output*.

We thus have the next result under the assumption that $(m-k)(k+1) \leq L$, where L is the machine word length in bits.

Theorem 2 *Our algorithm runs in $O(k^2(\|\mathcal{D}\|+|\mathcal{S}|)+km)$ time using $O(k^2\|\mathcal{D}\|)$ space for a simple collage system $\langle \mathcal{D}, \mathcal{S}\rangle$.*

5 Algorithm in detail

In this section, we discuss the construction of the two functions *Jump* and *Output*. Throughout this section, we assume that the union, the intersection, and the $\oplus$ operations for the subsets of Q can be performed in $O(1)$ time.

5.1 Construction of function *Jump*

Definition 3 *Let $I(w) = \hat{\delta}^{\rm pm}(\{\langle 0,0\rangle\}, w)$ for any string w in Σ^*.*

We can prove the following relation between the two transition functions $\hat{\delta}^{\rm pm}$ and $\hat{\delta}^{\rm ed}$.

Lemma 1 *For any $S \in 2^Q$ and any $w \in \Sigma^*$,*

$$\hat{\delta}^{\rm pm}(S,w) = \begin{cases} \hat{\delta}^{\rm ed}(S,w) \cup I(w), & \text{if } \langle 0,0\rangle \in S; \\ \hat{\delta}^{\rm ed}(S,w), & \text{otherwise.} \end{cases}$$

Note that $I(w)$ corresponds to the existence of the self-loop on the initial state $\langle 0,0\rangle$. Note also that the lemma is for general S in 2^Q. In fact, we do not have to care the sets S such that $\langle 0,0\rangle \notin S$, because the state $\langle 0,0\rangle$ of the NFA $(Q, \Sigma, \delta^{\rm pm}, \{\langle 0,0\rangle\}, F)$ is always active.

Definition 4 *For any $a \in \Sigma$, let*

$$G(a) = \{\langle i,j\rangle \in Q \mid i \neq 0, P[i] = a\},$$
$$H(a) = \{\langle i,j\rangle \in Q \mid P[i'] = a \text{ for some } i' \text{ with } 1 \leq i' \leq i\}.$$

Because we need only the states in $Q_{\rm core}$ and encode each of the sets $G(a) \cap Q_{\rm core}$ and $H(a) \cap Q_{\rm core}$ into a machine word, these tables can be built in $O(|\Sigma| + km)$ time using $O(|\Sigma|)$ space in a simple way.

Lemma 2 *For any $a \in \Sigma$ and any $u, v \in \Sigma^*$,*

$$I(a) = Q \cap \Big(E \cup (E \oplus \langle 0,1\rangle) \cup ((E \oplus \langle 1,0\rangle) \cap H(a)\Big)$$
$$I(uv) = \hat{\delta}^{\rm ed}(I(u), v) \cup I(v),$$

where $E = \{\langle i,j\rangle \in Q \mid i = j\}$.

Proof. Straightforward. ∎

The above lemma implies that, for a regular collage system, we can build the table which stores $I(t.u)$ for the tokens t in $\mathcal{D}$ by calling the function $\hat{\delta}^{\rm ed}$ $O(\|\mathcal{D}\|)$ times. Thus we concentrate on how to compute the function $\hat{\delta}^{\rm ed}$.

Definition 5 *For any $w \in \Sigma^*$ and any integers ℓ and d ($0 \leq \ell \leq m$ and $0 \leq d \leq k$), let*

$$V(w;\ell,d) = \left\{ i \;\middle|\; \begin{array}{l} 0 \leq i \leq m,\ 0 \leq i-\ell \leq m, \\ \langle i,d\rangle \in \hat{\delta}^{\rm ed}(\{\langle i-\ell, 0\rangle\}, w) \end{array} \right\},$$

and let

$$M(w;\ell,d) = \Big\{ \langle i,j\rangle \in Q \;\Big|\; i \in V(w;\ell,d) \Big\}.$$

Then we have the following lemma.

Lemma 3 *For any $S \in 2^Q$ and any $w \in \Sigma^*$,*

$$\hat{\delta}^{\rm ed}(S,w) = \bigcup_{\ell,d} (S \oplus \langle \ell, d\rangle) \cap M(w;\ell,d),$$

where ℓ and d satisfy $\max(0, |w|-k) \leq \ell \leq \min(|w|+k, m)$ and $0 \leq d \leq k$.

Proof. Let $\langle i,j\rangle \in \hat{\delta}^{\rm ed}(S,w)$. There exists $\langle i',j'\rangle \in S$ such that $\langle i,j\rangle \in \hat{\delta}^{\rm ed}(\{\langle i',j'\rangle, w)$. Let $\ell = i - i'$ and $d = j - j'$. It is easy to see that $\langle i,j\rangle \in S \oplus \langle \ell, d\rangle$, and that $0 \leq \ell \leq m$ and $0 \leq d \leq k$. By the definition of $\hat{\delta}^{\rm ed}$ we see that $\langle i,d\rangle \in \hat{\delta}^{\rm ed}(\{\langle i',0\rangle\}, w)$, and therefore $\langle i,j\rangle \in M(w;\ell,d)$. By Proposition 1, $\langle i,d\rangle \in \hat{\delta}^{\rm ed}(\{\langle i-\ell,0\rangle\}, w)$ implies that $D(P[i-\ell+1:i], w) \leq d$. Hence $|w| - d \leq |P[i-\ell+1:i]| \leq |w| + d$, and then $|w| - k \leq \ell \leq |w| + k$. Thus we have

$$\hat{\delta}^{\rm ed}(S,w) \subseteq \bigcup_{\ell,d} (S \oplus \langle \ell, d\rangle) \cap M(w;\ell,d).$$

On the other hand, let $\langle i,j\rangle \in (S \oplus \langle \ell, d\rangle) \cap M(w;\ell,d)$ for some ℓ and d such that $\max(0, |w|-k) \leq \ell \leq \min(|w|+k, m)$ and $0 \leq d \leq k$. We can prove in a similar manner that $\langle i,j\rangle \in \hat{\delta}^{\rm ed}(S,w)$. ∎

Let $\bar{M}$ be the table which stores $M(t.u;\ell,d)$ for the tokens t in $\mathcal{D}$, and the integers ℓ, d such that $\max(1, |t.u| - k) \leq \ell \leq \min(m, |t.u| + k)$ and $0 \leq d \leq k$. The size of $\bar{M}$ is $O(k^2\|\mathcal{D}\|)$. The above lemma implies that, for any $S \in 2^Q$ and any token t, we can obtain the value $\hat{\delta}^{\rm ed}(S, t.u)$ in $O(k^2)$ time using the table $\bar{M}$. We now consider the computation of the table $\bar{M}$.

Input. A text string represented as a pair of $\mathcal{D}$ and $\mathcal{S} = \mathcal{S}[1:n]$, and a pattern $P[1:m]$.
Output. A boolean value that indicates whether there is an approximate occurrence of P with k errors or less.
begin
/ Preprocessing */*
Build *Jump* and *Output* from the pattern P and the dictionary $\mathcal{D}$;

/ Text scanning */*
$R := I(\varepsilon)$; */* $I(w)$ is defined in Section 5 */*
for $\ell := 1$ **to** n **do begin**
if *Output*$(R, \mathcal{S}[\ell]) =$ *true* **then return** *true*;
$R :=$ *Jump*$(R, \mathcal{S}[\ell])$
end;
return *false*
end.

Figure 2. Overview of our algorithm.

Lemma 4 *For a simple collage system, we can build the table $\bar{M}$ in $O(k^2\|\mathcal{D}\| + km)$ time using $O(k^2\|\mathcal{D}\|)$ space.*

Proof. We can compute the table by the recurrence

$$M(\varepsilon;\ell,d) = \begin{cases} (Q \oplus \langle \ell, \ell \rangle) \cap Q, & \text{if } \ell = d \\ \emptyset, & \text{otherwise} \end{cases}$$

$$\begin{aligned} M(ua;\ell,d) = Q \cap \Big(& M(ua;\ell-1,d-1) \oplus \langle 1,0 \rangle \\ & \cup M(u;\ell-1,d-1) \oplus \langle 1,0 \rangle \\ & \cup M(u;\ell,d-1) \\ & \cup (M(u;\ell-1,d) \oplus \langle 1,0 \rangle) \cap G(a) \Big), \end{aligned}$$

$$\begin{aligned} M(au;\ell,d) \;=\; & M(au;\ell-1,d-1) \\ & \cup M(u;\ell-1,d-1) \\ & \cup M(u;\ell,d-1) \\ & \cup (M(u;\ell-1,d) \cap (G(a) \oplus \langle \ell-1,0 \rangle), \end{aligned}$$

where $u \in \Sigma^*$, $a \in \Sigma$, and $\ell \geq 0$ and $d \geq 0$. (We assume, for the sake of convenience, that $M(w;i,j) = \emptyset$ for all $w \in \Sigma^*$ if $i < 0$ or $j < 0$.) ∎

Unfortunately, for a regular collage system, we have not devised a way of computing the table $\bar{M}$. Thus our result is only for simple collage systems.

Lemma 5 *For a simple collage system, we can build the function Jump in $O(k^2\|\mathcal{D}\| + km)$ time using $O(k^2\|\mathcal{D}\|)$ space, so that it returns its value in $O(k^2)$ time.*

Proof. It follows from Lemmas 3 and 4. ∎

5.2 Construction of function *Output*

Definition 6 *For any $S \in 2^Q$ and any token t in $\mathcal{D}$, define Output$_1$ and Output$_2$ by*

Output$_1(t) = true$
$\Leftrightarrow$ *the string $t.u$ contains an approximate occurrence of P.*
Output$_2(S,t) = true$
$\Leftrightarrow$ *there exist integers ℓ and j with $\ell > 0,\ 0 \leq j \leq k$ and a non-empty prefix w of $t.u$ such that $\langle m-\ell, j \rangle \in S$ and $j + D(P[m-\ell+1:m], w) \leq k$.*

Definition 7 *For any $S \in 2^Q$ and any token X in $\mathcal{D}$, define Occ* by*

Occ$^\star(X) = true$
$\Leftrightarrow$ *there exist tokens Y, Z with $X = YZ \in \mathcal{D}$ and $u, v \in \Sigma^*$ such that u is a suffix of $Y.u$, v is a prefix of $Z.u$, and $D(uv, P) \leq k$.*

Let us consider the computation of *Occ**. We build the NFA for approximate pattern matching for the reversed pattern P, which plays a key role in the computation of *Occ**. Figure 3 shows the NFA for $P^R =$ NOMEKOP with $k = 4$. Note that the NFA in Fig. 3 is the same as the one obtained by reversing the direction of the arcs of the NFA in Fig. 1 except for the self-loop on the initial-state. The precise discussion is as follows. Define the mapping $\delta^{\mathrm{ed}}_{\mathrm{rev}} : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$ by

$$\begin{aligned} \delta^{\mathrm{ed}}_{\mathrm{rev}}(\langle i,j \rangle, \varepsilon) \;=\; & \{\langle i-1, j-1 \rangle \mid 0 \leq i-1, 0 \leq j-1\}, \\ \delta^{\mathrm{ed}}_{\mathrm{rev}}(\langle i,j \rangle, a) \;=\; & \{\langle i-1, j \rangle \mid 0 \leq i-1, P[i] = a\} \\ & \cup \{\langle i, j-1 \rangle \mid 0 \leq j-1\} \\ & \cup \{\langle i-1, j-1 \rangle \mid 0 \leq i-1, 0 \leq j-1\}, \end{aligned}$$

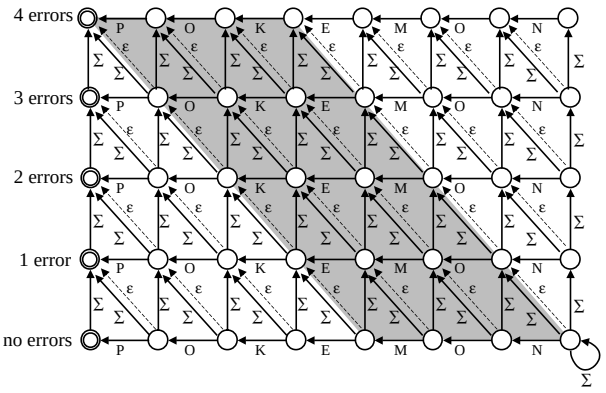

Figure 3. NFA for approximate string matching of the reversed pattern P^R = NOMEKOP with $k = 4$.

where $\langle i, j\rangle \in Q$ and $a \in \Sigma$. Define the mapping $\delta^{\rm pm}_{\rm rev} : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$ by

$$\begin{aligned}\delta^{\rm pm}_{\rm rev}(\langle i, j\rangle, \varepsilon) &= \delta^{\rm ed}_{\rm rev}(\langle i, j\rangle, \varepsilon),\\ \delta^{\rm pm}_{\rm rev}(\langle i, j\rangle, a) &= \delta^{\rm ed}_{\rm rev}(\langle i, j\rangle, a) \cup \{\langle m, k\rangle \mid i = m, j = k\},\end{aligned}$$

where $\langle i, j\rangle \in Q$ and $a \in \Sigma$. Let $F_{\rm rev} = \{\langle 0, j\rangle \mid 0 \le j \le k\}$. Obviously we have:

Proposition 3 *The NFA specified by the quintuple $(Q, \Sigma, \delta^{\rm ed}_{\rm rev}, \{\langle m, k\rangle\}, F_{\rm rev})$ accepts the language $\{w \in \Sigma^* \mid D(P^R, w) \le k\}$.*

Proposition 4 *The NFA specified by the quintuple $(Q, \Sigma, \delta^{\rm pm}_{\rm rev}, \{\langle m, k\rangle\}, F_{\rm rev})$ accepts the language $\Sigma^*\{w \in \Sigma^* \mid D(P^R, w) \le k\}$.*

Lemma 6 *For any strings $u, v \in \Sigma^*$, the string uv contains an approximate occurrence of P with at most k errors if and only if*

$$I(u) \cap I_{\rm rev}(v^R) \neq \emptyset,$$

where $I_{\rm rev}(w) = \hat{\delta}^{\rm pm}_{\rm rev}(\langle m, k\rangle, w)$ for $w \in \Sigma^*$.

Lemma 7 *For a simple collage system, we can build in $O(k^2\|\mathcal{D}\| + km)$ time using $O(k^2\|\mathcal{D}\|)$ space the table which stores the values Occ$^\star(t)$ for the tokens t in $\mathcal{D}$.*

Proof. All tokens X are defined in $\mathcal{D}$ as either $X = a$ or $X = YZ$, where $a \in \Sigma$ and Y, Z are tokens. When $X = a$, *Occ*$^\star(X) = \emptyset$ by the definition of *Occ**. When $X = YZ$, *Occ*$^\star(X) = true$ if and only if

$$I(Y.u) \cap I_{\rm rev}((Z.u)^R) \neq \emptyset,$$

by Lemma 6. The proof is complete. ∎

Lemma 8 *For a simple collage system, we can build in $O(k^2\|\mathcal{D}\| + km)$ time using $O(k^2\|\mathcal{D}\|)$ space the table which stores the values Output$_1(t)$ for the tokens t in $\mathcal{D}$.*

Proof. When $X = a$, *Output*$_1(X) = true$ if and only if $D(P, a) \le k$. When $X = YZ$,

$$\textit{Output}_1(X) = \textit{Output}_1(Y) \text{ OR } \textit{Output}_1(Z) \text{ OR } \textit{Occ}^\star(X),$$

where OR denotes the logical OR. The proof is complete. ∎

Now we turn into the computation of *Output*$_2$. The computation can be done with the same idea used for that of *Output*$_1$. By the definition of *Output*$_2$, it is not hard to see that *Output*$_2(S, t) = true$ if and only if

$$S \cap I_{\rm rev}((t.u)^R) \cap \{\langle i, j\rangle \in Q \mid i \neq m\} \neq \emptyset.$$

Thus we can prove the next lemma.

Lemma 9 *For a simple collage system, the function Output$_2$ can be built in $O(k^2\|\mathcal{D}\| + km)$ time using $O(k^2\|\mathcal{D}\|)$ space, so that it returns its value in $O(k^2)$ time.*

Since *Output*$(S, t) =$ *Output*$_1(t)$ OR *Output*$_2(S, t)$, we have the next result.

Lemma 10 *For a simple collage system, the function Output can be built in $O(k^2\|\mathcal{D}\| + km)$ time using $O(k^2\|\mathcal{D}\|)$ space, so that it returns the value in $O(k^2)$ time.*

Theorem 2 follows from Lemmas 5 and 10.

5.3 Bit-parallel implementation

In our implementation, we do not encode all of the automaton states in Q. We only encode the states in $Q_{\rm core}$ into a machine word. We thus substitute $Q_{\rm core}$ for Q. The discussion in Sections 5.1 and 5.2 is still true except for the computation of the table $\bar{M}$ mentioned in the proof of Lemma 4. For a correct computation of the table $\bar{M}$, we need some modification based on the following fact.

Proposition 5 *Let J be a subset of $\{0, \ldots, m\}$, and let $S = \{\langle i, j\rangle \in Q \mid i \in J\}$. Let $S_{\rm core} = S \cap Q_{\rm core}$. Then,*

$$Q_{\rm core} \cap (S \oplus \langle 1, 0\rangle) = Q_{\rm core} \cap \Big(S_{\rm core} \oplus \langle 1, 0\rangle \cup S_{\rm core} \oplus \langle 1, 1\rangle\Big).$$

Now, we explain the details of our bit-parallel implementation. We use the $(k(i - j - 1) + i)$th bit of a machine word to indicate whether or not the state $\langle i, j\rangle \in Q_{\rm core}$ is active (see Fig. 4). The union and the intersection operations on the subsets of $Q_{\rm core}$ can be evidently computed in $O(1)$ time by using the bitwise OR and AND, respectively. Next we discuss the computation of the operation $\oplus$. For

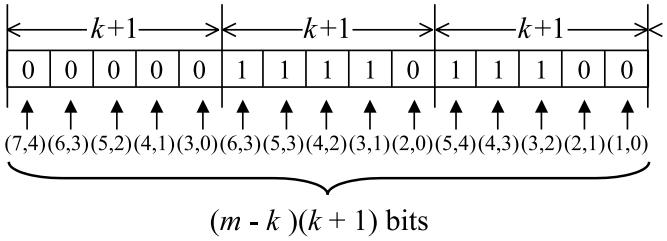

Figure 4. Encoding of the example NFA, where m=7 and k=4.

.

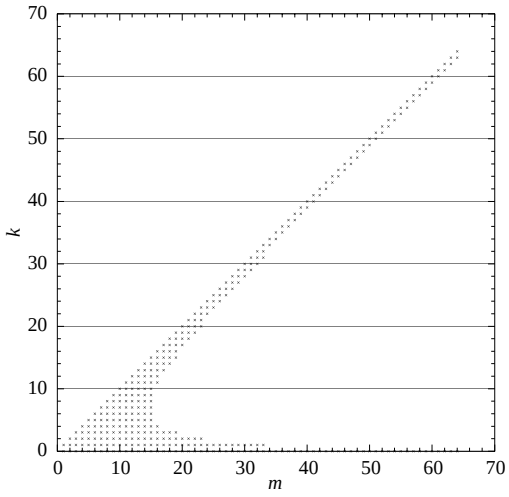

Figure 5. Combinations of m and k satisfying $(m-k)(k+1) \leq 64$.

.

a computation of $S \oplus \langle \ell, d \rangle$, it suffices to move all of the points $\langle i, j \rangle \in S$ to the right by $\ell - d$ (to the left by $d - \ell$, if $\ell < d$) and then move them down by d along the diagonal. The translation seems to be done using bit-shift by $(k+1)(\ell - d) + d$. However, this is not enough because the bits in the result which correspond to the points $\langle i, j \rangle$ with $0 \leq j < d$ should be $\emptyset$. To make these bits 0, we use mask bits which depend on d and can be in advance built in $O(k)$ time and space. Therefore we can execute the operation $\oplus$ in $O(1)$ time.

Figure5 shows the combinations of m and k satisfying $(m-k)(k+1) \leq L$, where $L = 64$.

6 Experimental results

To estimate the performance of our algorithm, we tested the following two programs for LZW compressed texts.

- *Decompression followed by a search based on the dynamic programing.*
  We embedded the approximate search routine based on the modified dynamic programing[11], which runs in $O(kN)$ time on the average, into the decompression program. This method is abbreviated as **uncompress+DP**.

- *Our algorithm.*
  This is abbreviated as **Bit-parallel on LZW**.

Our experiment was carried out on an AlphaStation XP1000 with an Alpha21264 processor at 667MHz running Tru64 UNIX operating system V4.0F. We used a subset of the GenBank database as a text file, which is an annotated collection of all publicly available DNA sequences. However, all fields other than accession number and nucleotide sequence were removed. The file size is about 17.1 Mbyte originally, and the compression ratio by **compress** is 26.80%. The patterns are randomly selected text substrings which we modified so that they have no approximate occurrences in the texts.

Table 1 shows the running times (in CPU time) for $k = 1$ and $m = 8 \sim 32$, where the preprocessing times were included. On the other hand, Table 2 shows the running times for $k = 1 \sim 5$ and $m = 14$.

Unfortunately, the results show that our algorithm is slower than the decompression followed by the dynamic programming. Although our algorithm runs in $O(k^2 n)$ time after preprocessing, the constant factor hidden is rather large.

7 Conclusion

We have presented the first algorithm using bit-parallelism for the problem of approximate string matching on compressed text. The algorithm is intended to solve the existence problem. Assuming that our problem fits in a single machine word, the algorithm runs in $O(k^2(\|\mathcal{D}\|+|\mathcal{S}|)+km)$ time using $O(k^2\|\mathcal{D}\|)$ space for a simple collage system. For the LZW or the LZ78 compression, the time and the space complexities are $O(k^2 n + km)$ and $O(k^2 n)$, respectively, where n is the compressed text length. This is a desirable property compared with the $O(mkn)$ time and space complexities of the algorithm due to Kärkkäinen et al. [4], although ours cannot cope with the all occurrence problem.

Unfortunately, the result of our experiments on LZW compressed text shows that the algorithm is in practice slower than a decompression followed by a search with the modified dynamic programming [11]. If we specialize the implementation to the LZW compression, the performance might be improved.

Table 1. CPU time comparison for $k = 1$.

m	8	12	16	20	24	28	32
uncompress+DP	2.235	2.243	2.241	2.241	2.250	2.240	2.235
Bit-parallel on LZW	5.364	6.233	6.475	6.633	6.829	7.021	7.269

Table 2. CPU time comparison for $m = 14$.

k	1	2	3	4	5
uncompress+DP	2.240	2.987	3.684	4.369	4.953
Bit-parallel on LZW	6.327	12.909	21.509	31.500	43.033

A practical fast algorithm for approximate string matching uses some filtering technique [7]. Although we have not yet devised a filtering technique suitable for compressed text searching, it will improve the performance of our algorithm in practice.

Acknowledgement

We would like to thank Professor Apostolico for fruitful discussions on this work. We would also like to thank the anonymous referees for suggestions which led to improvements in the presentation of the results.

References

[1] A. Amir and G. Benson. Efficient two-dimensional compressed matching. In *Proc. Data Compression Conference*, page 279, 1992.

[2] R. Baeza-Yates and G. Navarro. Faster approximate string matching. *Algorithmica*, 23:127–158, 1999.

[3] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[4] J. Kärkkäinen, G. Navarro, and E. Ukkonen. Approximate string matching over Ziv-Lempel compressed text. In *Proc. 11th Ann. Symp. on Combinatorial Pattern Matching*, pages 195–209. Springer-Verlag, 2000.

[5] T. Kida, Y. Shibata, M. Takeda, A. Shinohara, and S. Arikawa. A unifying framework for compressed pattern matching. In *Proc. 6th International Symp. on String Processing and Information Retrieval*, pages 89–96. IEEE Computer Society, 1999.

[6] N. J. Larsson and A. Moffat. Offline dictionary-based compression. In *Proc. Data Compression Conference '99*, pages 296–305. IEEE Computer Society, 1999.

[7] G. Navarro. A guided tour to approximate string matching. Technical Report TR/DCC-99-5, Dept. of Computer Science, Univ. of Chile, 1999.

[8] G. Navarro and M. Raffinot. A general practical approach to pattern matching over Ziv-Lempel compressed text. In *Proc. 10th Ann. Symp. on Combinatorial Pattern Matching*, pages 14–36. Springer-Verlag, 1999.

[9] C. G. Nevill-Manning, I. H. Witten, and D. L. Maulsby. Compression by induction of hierarchical grammars. In *DCC94*, pages 244–253. IEEE Press, 1994.

[10] P. Sellers. The theory and computation of evolutionary distances: pattern recognition. *J. of Algorithms*, 1:359–373, 1980.

[11] E. Ukkonen. Finding approximate patterns in strings. *J. Algorithms*, 6, 1985.

[12] T. A. Welch. A technique for high performance data compression. *IEEE Comput.*, 17:8–19, June 1984.

[13] S. Wu and U. Manber. Fast text searching allowing errors. *Comm. ACM*, 35(10):83–91, October 1992.

[14] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Trans. on Inform. Theory*, IT-23(3):337–349, May 1977.

[15] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Trans. on Inform. Theory*, 24(5):530–536, Sep 1978.