Approximate Pattern Matching with Grey Scale Values

Tadao TAKAOKA
Department of Computer Science
Ibaraki University
Hitachi, Ibaraki 316, JAPAN
E-mail: takaoka@cis.ibaraki.ac.jp

Abstract

We design an efficient algorithm for approximate pattern matching with grey scale values. The grey scale values range over interval $[0, R]$ *where* R *is a positive real number and the error bound for matching is measured by* kR *where* k *is a non-negative integer. Then we show that our algorithm solves this problem for the one-dimensional problem with* $O(kn \log m/m)$ *expected time, which is sublinear, where* m *and* n *are the sizes of pattern and text. For the two-dimensional problem, the expected time is* $O(kn^2 \log m/m^2)$ *where* m^2 *and* n^2 *are the sizes of pattern and text. The results are useful for time series analysis and graphics where grey scale matching is more suitable than character matching.*

Keywords Analysis of algorithms, approximate pattern matching, sampling, parallel algorithms, two-dimensional pattern matching, expected time analysis.

1 Introduction

Approximate string matching problem is to find a pattern in a text with k differences, where a difference is a mismatch, a superfluous character or a missing character. If we allow only mismatches, it is called the k-mismatch problem. For the k-mismatch problem, Landau and Vishkin [6] gave an algorithm with $O(kn)$ time and $O(m \log m)$ preprocessing time for a pattern, where m and n are the lengths of pattern and text. For the k-difference problem, Ukkonen [10], Landau and Vishkin [7] and Galil and Park [3] gave algorithms with $O(kn)$ time and $O(m^2)$ preprocessing time. In the area of two-dimensional approximate pattern matching, Amir and Landau [1] gave algorithms with $O(m^2)$ preprocessing time and $O(kn^2)$ matching time for the k-mismatch problem and $O(k^2n^2)$ matching time for the k-difference problem where superfluous characters or missing characters are only given in rows or columns, that is, the effects do not propagate to the next rows or columns. This definition of k-difference does not seem as natural as the k-difference problem in the one-dimensional case, and the k-mismatch problem seems adequately useful in graphics, etc.

Recently Chang and Lawler [2] published an algorithm for approximate string matching with sublinear expected time of $O((kn/m) \log_r m)$, where k is the number of allowable differences and r is the size of the alphabet, under an independent uniform distribution of characters in pattern and text. They say this is sublinear because m is in the denominator part. They use the suffix tree of the pattern to find subpatterns in the text. Takaoka [8] simplified the algorithm with a position table, in stead of suffix tree, resulting in $O((kn/m) \log_{1/q} m)$ time where q is the probability of character match, satisfying $1 < 1/q \leq r$, that is, the result is more general in that we can allow a deviated distribution of characters. Furthermore the simple structure of the algorithm allows generalization to the two-dimensional pattern matching with $O((kn^2/m^2) \log_{1/q} m)$ expected time, where k is the number of allowable mismatches and m^2 and n^2 are the sizes of pattern and text respectively.

In the present paper, we generalize the result in [8] to the problem of approximate pattern matching with grey scale values. This problem is more suitable for pattern matching in time series analysis and graphics where more general distances are used. The array values are not characters, but real numbers ranging 0 to R in the grey scale problem. Let us call the conventional pattern matching the character problem. The central ideas in the character problem in [8] are sampling and

Proceedings of CATS'96 (Computing: The Australasian Theory Symposium), Melbourne, Australia, January 29–January 30 1996.

data compression. We take a small number of sample sequences of $c \log_r m$ length from text and compress them into numbers ranging 0 to $m^c - 1$ contained in single words where c is an appropriate constant. On the other hand we take samples from pattern at every point and convert them into numbers and organize the sample positions in the pattern into the position table which can be accessed by the converted numbers. By consulting the table with the numerical values from the text, we determine the sample matches and identify the candidate positions in the text for exhaustive check based on the primitive dynamic programming.

In the grey scale problem, a grey scale value occupies a single word and we can not take an approach of data compression directly. An easily hit idea is to quantize the pattern and text into binary images using a threshold value. But one threshold value does not work. If we take $R/2$ as the threshold value, a very close match between pattern and text may cause a big difference in the binary images if the grey scale values are close to $R/2$ and at opposite sides. To overcome this problem, we introduce two threshold values of $R/3$ and $2R/3$, and quantize pattern and text into ternary images. With these ternary images, we can use the method in [8] successfully, and establish $O((kn/m)\log m)$ expected time for the one-dimensional problem and $O((kn^2/m^2)\log m)$ expected time for the two-dimensional problem, where the allowable distance for a match is bounded by kR. This result is a natural extension of the character problem to the grey scale problem. Logarithm is taken with base 2 unless otherwise specified. We use the notions of arrays, strings, and tuples interchangeably.

2 Review of sampling and data compression

We give the pattern in array $pat[0 \,..\, m-1]$ and the text in array $text[0 \,..\, n-1]$. We say we have a match at j, if $pat[i] = text[i+j]$ for $i = 0, \cdots, m-1$, which is denoted by $pat[0 \,..\, m-1] = text[j \,..\, j+m-1]$.

We assume that the characters in *pat* and *text* are drawn independently from a fixed distribution with probability $p(c)$ for character c. Let the alphabet be Σ. Then we have $\sum_{c\in\Sigma} p(c) = 1$. The probability q that a character in *pat* matches one in *text* is given by $q = \sum_{c\in\Sigma} p^2(c)$. The value of q is minimum $1/r$ when all $p(c)$ are equal to $1/r$ where $|\Sigma| = r$. Therefore we have $1 < 1/q \le r$.

Now let *pat* be put at position j in *text*. That is, $pat[0]$ is aligned with $text[j]$. Then the probability that $pat[i] = text[j+i]$ is given by q and this event is independent from the event that $pat[i'] = text[j+i']$ for $i' \neq i$. We extend this fact to events on array portions. Let the event that $pat[i \,..\, i+\ell-1] = text[j \,..\, j+\ell-1]$ be denoted by $Eve(i,j)$. Then we have $\mathrm{prob}[Eve(i,j)] = q^\ell$.

Now we introduce the idea of sampling into *pat* and *text*. Samples are substrings of length ℓ of *pat* and *text*. We take samples densely from *pat* and sparsely from *text*. Specifically we define samples from *pat*, denoted by $p\_sam(i)$, by

$$p\_sam(i) = pat[i \,..\, i+\ell-1], \quad \text{for } i = 0, 1, \cdots, m-\ell,$$

and samples from text, denoted by $t\_sam(j)$, by

$$t\_sam(j) = text[j \,..\, j+\ell-1], \quad \text{for } j = 0, h, \cdots, (L-1)h,$$

where $L = \lceil n/h \rceil$ is the number of text samples and h is the sampling period. For our approximate pattern matching purpose, we determine ℓ and h as follows:

$$\ell = \lceil 4 \log_{1/q} m \rceil, \quad h = \left\lfloor \frac{m-k-\ell+1}{k+1} \right\rfloor,$$

where k is the number of differences allowed for matching. The former value of h in [8] was corrected by Tarhio [9]. As we require that there be no overlaps between samples, we need the conditions that $\ell \le h$, that is, $k \le O(m/\log_{1/q} \log m)$. The value of ℓ is determined so that the probability that samples of *pat* and *text* match is very small and hence many candidate positions for approximate matches can be discarded quickly. When *pat* is positioned at j, the probability that α samples of *pat* match those of *text* is given by $\binom{k+1}{\alpha} q^{\alpha\ell} \le \binom{k+1}{\alpha} m^{-4\alpha}$. The value of h is determined to detect unallowable differences efficiently as seen from the following lemma.

Lemma 1 *If there is an approximate match with up to k differences when pat is positioned at j, there is at least one sample match between pat and text.*

To handle the samples of *pat* and *text* easily, we convert them into integers over $0 \,..\, r-1$. Let an arbitrary array $a[0 \,..\, \ell-1]$ of integer type $0 \,..\, r-1$ be converted into integer value $num(a)$ by

$$num(a) = a[0]r^{\ell-1} + a[1]r^{\ell-2} + \cdots + a[\ell-1].$$

Samples $p\_sam(i)$ and $t\_sam(j)$ are converted
into $num(p\_sam(i))$ and $num(t\_sam(j))$ similarly, converting characters into integers over $0 \,..\, r-1$ by some one-to-one mapping and then applying the function num. Obviously function num can be computed in $O(\ell)$ time by Horner's method. We can assure that these values can be contained in single words since ℓ is small. The time for computing all $num(p\_sam(i))$ is $O(m)$ and that for all $num(t\_sam(j))$ is $O(kn \log_{1/q} m/m)$.

To detect sample matches efficiently, we organize the set of positions i in pat into position table t with $num(p\_sam(i))$ as its key value. The easiest way is to make the table t of size $M = r^\ell = O(m^{4 \log r / \log(1/q)})$. The b-th element of t, $t[b]$, is the list of positions i such that $b = num(p\_sam(i))$. The construction of t takes $O(M)$ time because we need initialization of setting $t[b]$ to nil. If we use a hash function we can construct t in $O(m)$ expected time. For simplicity we use the $O(M)$-time version of t in the following matching algorithm.

The idea of matching is to find possible positions on $text$ at which pat may have an approximate match by using $b = num(t\_sam(j))$ and positions in $t[b]$. More specifically, if i is in $t[b]$, the positions $j-i+s$ for $-k \le s \le k$ on $text$ are candidates for an approximate match and so we perform an exhaustive check at $j-i+s$ on $text$. By exhaustive check we mean we use a naive dynamic programming algorithm with $O(m^2)$ time to compute the edit distance. If the distance is not more than k, "success" is reported, "failure", otherwise. In the algorithm, we maintain another data structure, array $pos[0 \,..\, L-1]$, whose elements are lists of candidate positions. If $t\_sam(j)$ matches $p\_sam(i)$, we append $j-i+s$ to $pos[j/h]$. All components of pos are initialized to nil.

Algorithm 1
Approximate character matching

```
1 for j := 0 to (L − 1)h step h do
    pos[j/h] :=nil;
2 for j := 0 to (L − 1)h step h do begin
3   b := num(t_sam(j));
4   for i ∈ t[b] do for s := −k to k do
      append j − i + s to pos[j/h]
5 end;
6 for x := 0 to L − 1 do begin
7   for u ∈ pos[x] do
      exhaustive check at u;
8   if "success" is reported then
      output(u)
9 end;
10 if "success"is never reported then
    output("no match").
```

The expected time $T(j)$ for the exhaustive check at j over text is bounded by

$$\begin{aligned} T(j) &= (2k+1)\sum_{\alpha=1}^{k+1} \binom{k+1}{\alpha} m^{-4\alpha} m^2 \\ &\le O(k^2 m^{-2}) \le O(km^{-1}). \end{aligned}$$

The total expected time for exhaustive checks is given by $\sum T(j) = O(kn/m)$. Other times in this algorithm are absorbed in this complexity. The dominant part is the time for text sampling, and thus we summarize our analysis as follows:

Time for preprocessing the pattern:
$O(m)$
Expected time for approximate matching:
$O(kn \log_{1/q} m/m)$.

The result can be easily extended into two dimensions and we have an $O(kn^2 \log_{1/q} m/m^2)$ expected time algorithm with $O(m^2)$ preprocessing time. In this analysis and in later sections, the following lemmas are extensively used.

Lemma 2 *Let $E_1, \cdots, E_n$ be probabilistic events. Then we have*

$$\text{prob}[E_1 \vee \cdots \vee E_n] \le \text{prob}[E_1] + \cdots + \text{prob}[E_n].$$

Lemma 3 *Let pat be positioned at j. Then several text samples may match those in pat. Similar events may occur if pat is positioned at other positions on text. Conditioned on these events, we do some work on the text starting at j. Then we have*

$$\begin{aligned} &E[E[\textit{work at start position } j \\ &\qquad \textit{given any condition}]] \\ &= E[\textit{work at start position } j]\ , \end{aligned}$$

where $E[X]$ is the expected value of random variable X.

3 Metric space over grey scale values

In this section we consider arrays whose elements take grey scale values, not characters, taken from interval $[0, R]$, where R is a positive real number.

Definition 1 The complement $\overline{x}$ of $x \in [0, R]$ is defined by

$$\overline{x} = \begin{cases} 0, & \text{if } x \ge R/2 \\ R, & \text{otherwise.} \end{cases}$$

Definition 2 An alignment of arrays $a[0\,..\,m\text{–}1]$ and $b[0\,..\,n\text{–}1]$ over grey scale values is the pair $(a'[0\,..\,m'-1], b'[0\,..\,n'-1])$ with $\max\{m,n\} < m' \leq m+n-1$ such that $a[i]$ $(i=0,\cdots,m-1)$ and $b[j]$ $(j=0,\cdots,n-1)$ are scattered over a' and b' with index sets $I=\{i_0,\cdots,i_{m-1}\}$ and $J=\{j_0,\cdots,j_{n-1}\}$ satisfying $i_0 < \cdots < i_{m-1}$ and $j_0 < \cdots < j_{n-1}$ and $I \cup J = \{0,\cdots,m'-1\}$, that is,

$$a'[i_k] = a[k] \quad (k=0,\cdots,m-1)$$
$$b'[j_k] = b[k] \quad (k=0,\cdots,n-1).$$

We fill other elements of a' and b' as follows:

$$a'[k] = \overline{b'[k]}, \quad \text{for } k \notin I$$
$$b'[k] = \overline{a'[k]}, \quad \text{for } k \notin J.$$

These definitions for $k \notin I$ or $k \notin J$ are regarded as penalty for missing or superfluous elements in pattern matching. Note that the words of missing and superfluous are used when the situation is seen from the side of a. We say $a'[k]$ and $b'[k]$ are proper elements if they come from a and b respectively. The definitions of non-proper elements in the above are done using proper elements of partners. The alignment is abbreviated as (a', b').

Definition 3 The distance between $a[0\,..\,m-1]$ and $b[0\,..\,n-1]$ over grey scale values, $\delta(a,b)$, is defined by

$$\delta(a,b) = \min \sum_{k=0}^{m'-1} |a'[k] - b'[k]|,$$

where "min" is taken over all possible alignments (a', b'). An alignment that gives the above minimum is called an optimal alignment.

Lemma 4 *The function δ defines a metric space over arrays over $[0,R]$. That is, it satisfies the following three axioms of distance:* (1) $\delta(a,a)=0$, (2) $\delta(a,b)=\delta(b,a)$, *and* (3) $\delta(a,c) \leq \delta(a,b)+\delta(b,c)$.

See the appendix for the proof of this lemma. The distance $\delta(a,b)$ can be computed in $O(mn)$ time using an obvious dynamic programming technique. The algorithm is given below.

Algorithm for $\delta(a,b)$

$$d_{-1,-1} = 0$$
$$d_{i,-1} = \sum_{k=0}^{i} |a[k] - \overline{a[k]}|$$
$$(i=0,\cdots,m-1)$$
$$d_{-1,j} = \sum_{k=0}^{j} |b[k] - \overline{b[k]}|$$
$$(j=0,\cdots,n-1)$$
$$d_{i,j} = \min \begin{cases} d_{i-1,j} + |a[i] - \overline{a[i]}| \\ d_{i,j-1} + |b[j] - \overline{b[j]}| \\ d_{i-1,j-1} + |a[i] - b[j]| \end{cases}$$
$$(i=1,\cdots,m-1; j=1,\cdots,n-1)$$
$$\delta(a,b) = d_{m-1,n-1}.$$

Example Vectors a and b over $[0,7]$ are given by

$$a = (3,0,7,1,6,3)$$
$$b = (2,5,0,7,4,1).$$

The distance $\delta(a,b) = 16$ is given by the alignment.

$$a' = (3,\overline{5},0,7,1,6,3)$$
$$b' = (2,5,0,7,\overline{1},4,1),$$

where $\overline{5}=0$ and $\overline{1}=7$. This is seen from the following trace of $d_{i,j}$.

	j	-1	0	1	2	3	4	5
	b		2	5	0	7	4	1
i	a							
-1		*0*	5	10	17	24	28	34
0	3	4	*1*	*6*	13	20	24	30
1	0	11	6	6	*6*	13	17	30
2	7	18	13	8	13	*6*	10	22
3	1	24	19	14	9	*12*	9	15
4	6	30	25	20	20	10	*14*	15
5	3	34	29	24	23	14	11	*16*

For example, $d_{3,3}=12$ is computed as

$$\min\{d_{2,3} + |a[3] - \overline{a[3]}|,$$
$$d_{3,2} + |b[3] - \overline{b[3]}|,$$
$$d_{2,2} + |a[3] - b[3]|\},$$
$$= \min\{6+6, 9+7, 13+6\} = 12.$$

The computation path for $d_{5,5}$ is shown in italic.

4 Quantization of grey scale values

In this section we quantize grey scale values x into a ternary value $\tau(x)$ by using two threshold values $R/3$ and $2R/3$ as follows:

$$\tau(x) = \begin{cases} 0, & \text{if } 0 \leq x < R/3 \\ 1, & \text{if } R/3 \leq x \leq 2R/3 \\ 2, & \text{if } 2R/3 < x \leq R. \end{cases}$$

Lemma 5 *Let x and y be from $[0,R]$. Then*

$$|\tau(x) - \tau(y)| \geq 2 \Longrightarrow |x-y| > R/3.$$

We convert arrays over grey scale values into ternary arrays by applying function τ to array elements. Specifically array $a[0\ ..\ m-1]$ is converted to $\tau(a)[0\ ..\ m-1]$ by

$$\tau(a)[0\ ..\ m-1] = (\tau(a[0]), \cdots, \tau(a[m-1])).$$

Definition 4 Two arrays $a[0\ ..\ m-1]$ and $b[0\ ..\ m-1]$ over $[0, R]$ are said to be close if

$$|\tau(a[k]) - \tau(b[k])| \leq 1 \quad (k = 0, \cdots, m-1).$$

Ternary arrays $\tau(a)$ and $\tau(b)$ are said to be neighbours if this condition holds.

Lemma 6 *Let x and y be mutually independent random variables distributed uniformly over $[0, R]$. Then*

$$\text{prob}[|\tau(x) - \tau(y)| \leq 1] = 7/9.$$

Lemma 7 *Let $a[0\ ..\ m-1]$ and $b[0\ ..\ m-1]$ are arrays whose elements are mutually independent random variables distributed uniformly over $[0, R]$. Then*

$$\text{prob}[a \text{ and } b \text{ are close}] = (7/9)^m.$$

Now we apply our theory to *pat* and *text* which are arrays over $[0, R]$. A sampling mechanism is defined for *pat* and *text* similarly to that in Section 2. Let us correspond $text[j\ ..\ j-m]$ to $a[0\ ..\ m-1]$, $pat[0\ ..\ m-1]$ to $b[\ 0..\ m-1]$, t_sam to a_sam and p_sam to b_sam. The following theorem is central to our algorithm design.

Theorem 1 *Let $\delta(a, b) \leq KR/3$ for some non-negative integer K. Then there are integers k and k' such that $0 \leq k,\ k' \leq m-1$, $|k - k'| \leq \lceil 2K/3 \rceil$ and $a\_sam(k)$ and $b\_sam(k')$ are close. We use K for the error bound since we use k for sample positions.*

Proof. Let $(a'[0\ ..\ m'-1], b'[0\ ..\ m'-1])$ be an optimal alignment. Under this alignment we assume $a[k]$ is moved to $a'[i_k]$. Define interval I_k of integers by $[i_k, i_k + \ell - 1]$. Suppose that for all $k = 0, h, 2h, \cdots, (K-1)h$

$$^{\exists} i \in I_k\ |a'[i] - b'[i]| > R/3,$$

where $h = \lfloor (m - K - \ell + 1)/(K+1) \rfloor$ and $\ell \leq h$. Then we have

$$\begin{aligned} \delta(a,b) &= \sum_{i=0}^{m'-1} |a'[i] - b'[i]| \\ &\geq \sum_{j=0}^{K-1} \sum_{i \in I_{jh}} |a'[i] - b'[i]| \\ &> KR/3, \end{aligned}$$

which is a contradiction. Therefore we have for some k

$$^{\forall} i \in I_k\ |a'[i] - b'[i]| \leq R/3.$$

Assume that $b[k']$ is moved to $b'[k]$ under the alignment. If $a'[i]$ is a superfluous or missing element in I_k, we have $|a'[i] - b'[i]| \geq R/2$. Therefore there is no superfluous or missing element in I_k. From Lemma 5, we see that $|\tau(a'[i]) - \tau(b'[i])| \leq 1$. Since superfluous or missing elements cause at least $R/2$ penalty in $\delta(a, b)$, we can not have more than $\lfloor 2K/3 \rfloor$ such elements to the left of I_k under the alignment, which means k' is to the left or to the right of k by at most $\lfloor 2K/3 \rfloor$. Note that $\tau(b\_sam(k'))$ is a neighbour of $\tau(a\_sam(k))$.

5 Algorithm for grey scale matching

Based on the results in the previous sections, we now develop an efficient algorithm for approximate grey scale matching within $KR/3$ distance. We reasonably assume that $K \leq O(m)$. As we scan the text and make ternary arrays of samples of text, or ternary text samples for short, we want to know quickly whether there are neighbours of ternary pattern samples. These samples are converted into integer values by the method in Section 2 with $r = 3$. We call these, numerical samples. The $m-\ell$ numerical pattern samples and the numerical values of their neighbours are sorted into table T. The expected size of table T is $O(m(7/3)^\ell)$ and can be constructed in $O(m\ell(7/3)^\ell)$ expected time. Then we can decide whether a numerical text sample is a neighbour of any numerical pattern sample by binary search on T in $O(\ell)$ time. We call this binary search, which gives true if and only if a numerical text sample is in T, "check". Also all neighbours of a numerical text sample are generated systematically, each in $O(\ell)$ time. In the algorithm, this is expressed as exhausting the set of neighbours of $\tau(t\_sam(j))$, $neighbours(\tau(t\_sam(j)))$. The rest is much the same as Algorithm 1. Let $L = \lceil n/h \rceil$.

Algorithm 2

```
1 for j := 0 to (L − 1)h step h do
    pos[j/h] :=nil;
2 for j := 0 to (L − 1)h step h do begin
3   x := num(τ(t_sam(j)));
4   if check(x) then
5     for y ∈ neighbours(τ(t_sam(j)) do
6       for i ∈ t[y] do
7         for s := −⌊2K/3⌋ to ⌊2K/3⌋ do
8           append j − i + s to pos[j/h]
```

```
 9 end;
10 for x := 0 to L − 1 do begin
11    for u ∈ pos[x] do
        exhaustive check at u;
12    if "success" is reported then
        output(u);
13 end;
14 if "success" is never reported then
     output("no match").
```

6 Analysis

We assume that grey scale values in *pat* and *text* are uniformly distributed over $[0, R]$ and they are independent. From Lemma 7, we see that the probability that a text sample and a pattern sample of length ℓ are close, is given by $(7/9)^\ell$. We choose $\ell = \lceil 4 \log_{9/7} m \rceil$. Then this probability becomes m^{-4}. The probability that $check(x)$ is true for random x is bounded, using Lemma 2, by

$$\text{prob}[check(x) \text{ is true}] \leq m(7/9)^\ell = O(m^{-3}).$$

The expected time spent at lines 5 – 8 for each x is bounded by $O(3^\ell m(1/3)^\ell K) = O(Km)$, since the expected size of list $t[y]$ is $m(1/3)^\ell$ and the size of neighbours at line 5 is bounded by 3^ℓ. Therefore the expected time for lines 2 – 9 apart from line 3 is bounded by

$$\begin{aligned} O(m^{-3} \cdot Km \cdot n/h) &= O(K^2 n \cdot m^{-3}) \\ &\leq O(Kn/m^2). \end{aligned}$$

The probability that a position on text is a candidate for exhaustive check is bounded by

$$\begin{aligned} & (2\lceil 2K/3 \rceil + 1) \sum_\alpha \binom{K}{\alpha} (7/9)^{\alpha\ell} \\ = \;& O(K^2 (7/9)^\ell) = O(K^2 m^{-4}). \end{aligned}$$

Therefore the expected time for lines 10 – 13 is bounded by

$$\begin{aligned} O(K^2 m^{-4} \cdot n \cdot m^2) &= O(K^2 n/m^2) \\ &\leq O(Kn/m). \end{aligned}$$

In this analysis, we sum up expected times for all x and obtain correct estimation owing to Lemma 3.

The dominant part of our algorithm is the computing time for $num(t\_sam(j))$ $(j = 0, h, \cdots, (L-1)h)$ at line 3, which takes $O(Kn \log m/m)$ time and is not subject to probabilistic fluctuations.

The preprocessing of the pattern consists of two parts. One is to construct table t, which takes $O(\ell 3^\ell) = O(m^{4.37} \log m)$ time. The expected time for constructing T takes $O(m\ell(7/3)^\ell) = O(m^{4.37} \log m)$ time.

The result can be extended to two dimensions easily, and we get an algorithm with $O(Kn^2 \log m/m^2)$ expected time for matching to find an (m, m) pattern in the (n, n) text within $O(KR)$ distance where distance is defined without superfluous or missing elements. We can convert the algorithm for the two dimensional character problem in [8] to that for the grey scale problem.

7 Probabilistic dependency

In practical applications in time series analysis or graphics, very often grey scale values of neighbouring elements or pixels are not probabilistically independent. Naturally we can assume that the quantized values of neighbouring elements are probabilistically dependent. We model the dependency by a simple Markov chain for the one-dimensional character problem. The grey scale problem can be similarly treated by quantization.

Let *pat* and *text* be generated by a simple Markov chain whose transition matrix is given by $P = [p_{ij}]$, that is, the conditional probability is given by $\text{prob}[c_j | c_i] = p_{ij}$ $(i, j = 1, \cdots, r)$, where $\Sigma = \{c_1, \cdots, c_r\}$. Let $\boldsymbol{p} = (p_1, \cdots, p_r)$ be the stationary probability vector of characters $c_1, \cdots, c_r$. Then

$$\begin{aligned} & \text{prob}[pat[i \,..\, i+\ell-1] = text[j \,..\, j+\ell-1]] \\ &= \sum_{i_1=1}^{r} \sum_{i_2=1}^{r} \cdots \sum_{i_\ell=1}^{r} p_{i_1}^2 p_{i_1 i_2}^2 \cdots p_{i_{\ell-1} i_\ell}^2 \\ &= \boldsymbol{q} Q^\ell \boldsymbol{\xi}^T, \end{aligned}$$

where $\boldsymbol{q} = (p_1^2, \cdots, p_r^2)$, $\boldsymbol{\xi} = (1, \cdots, 1)$, $Q = [q_{ij}]$, $q_{ij} = p_{ij}^2$, and the sign "T" means transposition. The Frobenius root λ of Q, which is the greatest characteristic value of Q (see [4] for the Frobenius theory), satisfies

$$\lambda \leq \max_i \{ \sum_{j=1}^{r} q_{ij} \}.$$

Since we can assume that

$$\sum_{j=1}^{r} q_{ij} < 1 \quad (i = 1, \cdots, r),$$

we have $\lambda < 1$. The Frobenius theory further tells us that the characteristic vector $\boldsymbol{x} = (x_1, \cdots, x_r)$ corresponding to λ satisfies that $x_i > 0$ $(i = 1, \cdots, r)$. Let $\min\{x_i\} = x$. Then $\boldsymbol{\xi} \leq (1/x)\boldsymbol{x}$. Thus we have

$$\begin{aligned} & \text{prob}[pat[i \,..\, i+\ell-1] = text[j \,..\, j+\ell-1]] \\ &= \boldsymbol{q} Q^\ell \boldsymbol{\xi}^T \leq \boldsymbol{q} Q^\ell (1/x) \boldsymbol{x}^T \\ &= (1/x) \lambda^\ell \boldsymbol{q} \boldsymbol{x}^T = O(\lambda^\ell), \end{aligned}$$

where $\boldsymbol{q}\boldsymbol{x}^T$ is the inner product of $\boldsymbol{q}$ and $\boldsymbol{x}$. This value of λ can be used in place of q in Section 2. Since $q < \lambda$ in general, the performance of algorithm deteriorates as the dependency increases. To overcome this problem, we make samples using separate array elements, distance d apart. The Markov chain theory [5] says that the transition matrix P^d approaches the stationary probability matrix exponentially fast as d increases. That means that the process described by P^d is almost an independent process even for small d and that we increase the efficiency at the cost of reducing the allowable error bound as $k \leq O(m/(d \log m))$. The two-dimensional problem can be similarly treated.

8 Concluding remarks

An interesting aspect of our algorithms is that the dominant part of computing time is that of text sampling, which is not subject to probabilistic fluctuation. That is, the expected time is almost worst case time. The probability that the time of Algorithm 4 exceeds the time for text sampling is very small and will be evaluated by using its standard deviation in the future research.

If we take r quantization levels instead of 3, the main complexity of text sampling becomes $O(rKn \log_r m/m)$. Note, for example, that $R/3$ is replaced by R/r in Lemma 5. Thus the three level quantization is optimal in our frame work of algorithm, since we can minimize the above complexity at $r = e$ by differentiation where e is the base of natural logarithm. The size and computing time for tables t and T can be smaller with $r > 3$, however.

Acknowledgements

This research is partially supported by Grant-in-aid No. 07680332 by Monbusho Scientific Research Program and a research grant from Hitachi Engineering Co., Ltd.

References

[1] A. Amir and G.M. Landau, G. M. Fast parallel and serial multidimensional approximate array matching. *Theor. Comput. Sci.*, Volume 81, pages 97–115, 1991.

[2] W.I. Chang and E.L. Lawler. Approximate string matching in sublinear expected time. *FOCS*, Volume 31, pages 116–124, 1990.

[3] Z. Galil and K. Park. An improved algorithm for approximate string matching. ICALP'89, *LNCS*, Volume 372, pages 394–404, 1989.

[4] F.R. Gantmacher. *The Theory of Matrices*, Volume Two. Chelsea Pub. Co., 1964

[5] J.G. Kemeny, J.L. Snell. *Finite Markov Chains*. D. Van Nostrand Com., 1960.

[6] G.M. Landau and U. Vishkin. Efficient string matching with k mismatches. *Theor. Comput. Sci.*, pages 239–249, 1985.

[7] G.M. Landau and U. Vishkin. Fast string matching with k differences. *JCSS*, Volume 37, pages 63–78, 1988.

[8] T. Takaoka. Approximate pattern matching with samples. Proc. of the 5th Inter. Symp., ISAAC'94, Beijing, P.R. China, August 1994. *LNCS*, Volume 834, pages 234–242, 1994.

[9] J. Tarhio. *Private Communication, Feb. 29, 1995.*

[10] E. Ukkonen. Algorithms for approximate string matching. *Information and Control*, Volume 64, pages 100–118, 1985.

Appendix: Proof of Lemma 4.

We show the proof of (3), since (1) and (2) are straightforward. Let an optimal alignment for a and b be $(a'[0 \,..\, m'-1], b'[0 \,..\, m'-1])$ and that for b and c be $(b''[0 \,..\, m''-1], c'[0 \,..\, m''-1])$. To define an alignment $(a''[0 \,..\, m'''-1], c''[0 \,..\, m'''-1])$. we scan b' and b'' from left to right and align proper elements of b in b' and b''. To make the alignment compact, we shift pairs $(a'[i], b'[i])$ and $b''[j], c'[j])$ to the right by fewest places, and store $a'[i]$ and $c'[j]$ in the corresponding positions in a'' and c''. If $(a'[i], b'[i]) = (\overline{b_k}, b_k)$ and $(b''[j], c'[j]) = (b_k, \overline{b_k})$, we discord these pairs. For example,

$$\begin{aligned}
a' &= (a_1,\overline{b_2},a_2,a_3,a_4,a_5,\overline{b_5})\\
b' &= (b_1,b_2,b_3,\overline{a_3},b_4,\overline{a_5},b_5)\\
b'' &= (b_1,\overline{c_2},b_2,b_3,b_4,\overline{c_5},b_5)\\
c' &= (c_1,c_2,c_3,c_4,\overline{b_4},c_5,\overline{b_5})\\
a'' &= (a_1,\overline{c_2},\overline{b_2},a_2,a_3,a_4,a_5,\overline{c_5})\\
b''' &= (b_1,\overline{c_2},b_2,b_3,\overline{a_3},b_4,\overline{a_5},\overline{c_5})\\
c'' &= (c_1,c_2,c_3,c_4,\overline{a_3},\overline{b_4},\overline{a_5},c_5).
\end{aligned}$$

In the above, a_i, b_i and c_i are proper elements of a, b and c. We name the resulting array composed of b' and b'' by b'''. After all pairs in (a', b') and (b'', c') are processed, m''' is determined.

In this construction, there remain undefined elements in a'' and c''. If $(a''[k], b'''[k]) = (a_i, \overline{a_i})$ for some i, let $c''[k] = \overline{a_i}$. If $(b'''[k], c''[k]) = (\overline{c_j}, c_j)$ for some j, let $a''[k] = \overline{c_j}$. Letting

$$D(a,c) = \sum_{k=0}^{m'''-1} |a''[k] - c''[k]|,$$

we prove that $D(a,c) \leq \delta(a,b) + \delta(b,c)$. Observe that

$$D(a,c) \leq \sum_{k=0}^{m'''-1} \{|a''[k] - b'''[k]| + |b'''[k] - c''[k]|\}.$$

We have five patterns for triples $(a''[k], b'''[k], c''[k])$.

(1) (a_i, b_ℓ, c_j), that is, all are proper elements. In this case, $|a_i - b_\ell|$ and $|b_\ell - c_j|$ exist in the summations for $\delta(a,b)$ and $\delta(b,c)$.

(2) $(a_i, \overline{a_i}, \overline{a_i})$. Term $|a_i - \overline{a_i}|$ exists in $\delta(a,b)$ and $|\overline{a_i} - \overline{a_i}| = 0$.

(3) $(\overline{c_j}, \overline{c_j}, c_j)$. Similar to (2).

(4) $(a_i, b_\ell, \overline{b_\ell})$. Terms $|a_i - b_\ell|$ and $|b_\ell - \overline{b_\ell}|$ exist in $\delta(a,b)$ and $\delta(b,c)$.

(5) $(\overline{b_\ell}, b_\ell, c_j)$. Similar to (5).

From these we see that $D(a,c) \leq \delta(a,b) + \delta(b,c)$. Since $\delta(a,c) \leq D(a,c)$, we have proved $\delta(a,c) \leq \delta(a,b) + \delta(b,c)$.