アルゴリズム 35－4
（1993．10．1）

Don't Care記号つき文字列に対する近似マッチング・アルゴリズム

阿久津 達也
機械技術研究所

文字列の近似マッチング・アルゴリズムとしてLandau と Vishkin によるアルゴリズムが良く知られている。本稿では彼らのアルゴリズムを基に開発した、Don't Care 記号付き文字列に対する逐次および並列の近似マッチング・アルゴリズムを示す。なお、n をテキスト文字列の長さ、m をパターン文字列の長さ、Σ を文字集合、k を許容誤差とした場合に、逐次アルゴリズムは $O(\sqrt{km}\,n \log|\Sigma| \log^2 \frac{m}{k} \log\log \frac{m}{k})$ 時間で動作し、並列アルゴリズムはCRCW-PRAM 上で $O(\sqrt{\frac{m}{k}}\,n \log|\Sigma| \log \frac{m}{k} \log\log \frac{m}{k})$ 個のプロセッサを用いて $O(k \log m)$ 時間で動作する。さらに、より拡張した文字列パターンに対する近似マッチング・アルゴリズムも示す。

APPROXIMATE STRING MATCHING WITH DON'T CARE CHARACTERS

Tatsuya AKUTSU

Mechanical Engineering Laboratory,
1-2 Namiki, Tsukuba, Ibaraki, 305 Japan.
e-mail: akutsu@mel.go.jp

This paper presents parallel and serial approximate matching algorithms for strings with don't care characters. They are based on the approximate string matching algorithm developed by Landau and Vishkin. The serial algorithm works in $O(\sqrt{km}\,n \log|\Sigma| \log^2 \frac{m}{k} \log\log \frac{m}{k})$ time and the parallel algorithm works in $O(k \log m)$ time using $O(\sqrt{\frac{m}{k}}\,n \log|\Sigma| \log \frac{m}{k} \log\log \frac{m}{k})$ processors on a CRCW-PRAM, where n denotes the length of a text string, m denotes the length of a pattern string, k denotes the maximum number of differences and Σ denotes the alphabet (i.e. the set of characters). Several extensions of algorithms are described, too.

1 Introduction

Approximate string matching is important not only from a theoretical viewpoint but also from a practical viewpoint. In particular, it is important for molecular biology since the exact matching is not sufficient [4]. While several variants are considered in approximate string matching, the *string matching with k differences* is the most important one. It is defined as follows [7, 8]. Let $T = t_1 \cdots t_n$ be a text string and $P = p_1 \cdots p_m$ be a pattern string over an alphabet Σ. Note that $|\Sigma| \leq m$ can be assumed without loss of generality. A *difference* is one of the following:

(A) A character of the pattern corresponds to a different character of the text.

(B) A character of the pattern corresponds to no character in the text.

(C) A character of the text corresponds to no character in the pattern.

If $t_i \cdots t_j$ can correspond to P with at most k differences, we say that P occurs at position j of T with at most k differences. Then, the problem is defined as follows: given a text string T, a pattern string P and an positive integer k $(1 \leq k \leq m)$, find all positions of T where P occurs with at most k differences.

[Example 1] P ="bcdefgh" occurs at position 8 of T ="abxdyeghij" with differences 3 by the following correspondence:

```
T     a  b  x  d  y  e     g  h  i  j
P        b  c  d     e  f  g  h
            (A)    (C)    (B)
```

Landau and Vishkin developed an $O((k + \log m)n)$ time algorithm for the string matching with k differences [8]. Galil and Park developed theoretically and practically improved algorithms [7].

Although efficient algorithms are developed, the string matching with k differences seems insufficient for such applications as motif search. In motif search, *don't care* characters are frequently used [4], where the don't care character is a character which matches any character. In this paper, '$*$' denotes the don't care character. Concerning with don't care characters, the exact string matching with don't cares was studied about 20 years ago. Fisher and Paterson developed an $O(n \log |\Sigma| \log^2 m \log\log m)$ time algorithm based on the fast calculation method of convolutions [5]. Their technique was applied to various pattern matching problems [1, 2]. Abrahamson generalized their algorithm and developed algorithms for the *generalized string matching* [1].

Combining Landau and Vishkin's algorithm with Fisher and Paterson's algorithm, we developed serial and parallel algorithms for the string matching with k differences in which don't cares may appear both in a text string and in a pattern string. The serial algorithm works in $O(\sqrt{km}\, n \log |\Sigma| \log^2 \frac{m}{k} \log\log \frac{m}{k})$ time and the parallel algorithm works in $O(k \log m)$ time using $O(\sqrt{\frac{m}{k}}\, n \log |\Sigma| \log \frac{m}{k} \log\log \frac{m}{k})$ processors, where a CRCW-PRAM is assumed as a model of a parallel computer. While suffix trees are used in Landau and Vishkin's algorithm, tables which are constructed by Fisher and Paterson's algorithm are used in our algorithms. This paper describes these algorithms. Extensions for more general patterns of strings are described, too.

[Example 2] P ="bc$*$eghi" occurs at position 8 of T ="a$*$cdefgij" with differences 2 by the following correspondence:

```
T    a  *  c  d  e  f  g     i  j
P       b  c  *  e     g  h  i
                  (C)    (B)
```

2 String matching with k differences

In this section, we overview Landau and Vishkin's approximate string matching algorithm [8].

2.1 Simple algorithm based on dynamic programming

In this subsection, we describe an $O(mn)$ time algorithm for the string matching with k differences. It was developed by a lot of persons independently. It is based on the dynamic programming technique.

Let $D[i,j]$ $(0 \leq i \leq m$ and $0 \leq j \leq n)$ be the minimum number of differences between $p_1 \cdots p_i$ and any substring of T ending at t_j. Then, it is easy to see that $D[i,j]$ is determined by

$$D[i,j] \;=\; \min(\; D[i-1,j]+1,\; D[i,j-1]+1,\; D[i-1,j-1]+h\;)$$

where $h = 0$ if $t_j = p_i$ and $h = 1$ otherwise. Thus, the following procedure solves the string matching with k differences in $O(mn)$ time. Note that all occurrences can be enumerated by outputting all j's such that $D[m,j] \leq k$.

Procedure *SimpleDynamic*(P,T)
begin
  for all j such that $0 \leq j \leq n$ **do** $D[0,j] \leftarrow 0$;
  for all i such that $0 \leq i \leq m$ **do** $D[i,0] \leftarrow i$;
  for $i = 1$ **to** m **do**
    for $j = 1$ **to** n **do**
      begin
        if $p_i = t_j$ **then** $h \leftarrow 0$ **else** $h \leftarrow 1$;
        $D[i,j] \leftarrow \min(\; D[i-1,j]+1,\; D[i,j-1]+1,\; D[i-1,j-1]+h\;)$
      end
end

[Example 3] Let P ="CAAG" and T ="CCCAGAT". Then, the following table shows the values of $D[i,j]$'s.

		C	C	C	A	G	A	T
	0	0	0	0	0	0	0	0
C	1	0	0	0	1	1	1	1
A	2	1	1	1	0	1	1	2
A	3	2	2	2	1	1	1	2
G	4	3	3	3	2	1	2	2

2.2 Landau and Vishkin's algorithm

While the simple dynamic programming algorithm takes $O(mn)$ time, Landau and Vishkin developed an $O((k + \log m)n)$ time algorithm [8]. Their algorithm computes the same in-

formation as in the matrix $D[i,j]$ of the simple dynamic programming algorithm, using the *diagonals* of the matrix. A diagonal d of the matrix consists of all $D[i,j]$'s such that $j-i=d$.

For a number of differences e and a diagonal d, $L[d,e]$ denotes the largest row i such that $D[i,j]=e$ and $j-i=d$. For example, $L[3,0]=0$, $L[3,1]=3$ and $L[3,2]=4$ in the case of Example 3. Note that the value of $D[i,j]$ such that $j-i=d$ grows monotonically as i grows. Thus, for the string matching with k differences, we need only compute the values of $L[d,e]$'s such that $e \leq k$. Since the number of diagonals is $O(n)$, the number of $L[d,e]$'s which are required to be computed is $O(kn)$. Moreover, Landau and Vishkin showed that $L[d,e]$'s could be computed by the following procedure.

```
Procedure LandauVishkin(P,T)
begin
  for all d such that 0 ≤ d ≤ n do L[d,-1] ← -1;
  for all d such that -(k+1) ≤ d ≤ -1 do
    begin
      L[d,|d|-1] ← |d|-1;
      L[d,|d|-2] ← |d|-2
    end
  for all e such that -1 ≤ e ≤ k do L[n+1,e] ← -1;
  for e = 0 to k do
    for all d such that -e ≤ d ≤ n do
      begin
        row ← max(L[d,e-1]+1, L[d-1,e-1], L[d+1,e-1]+1);
        row ← min(row, m);
        while row < m and row + d < n and p_{row+1} = t_{row+1+d} do row ← row+1;  -($)
        L[d,e] ← row;
        if L[d,e] = m then
          Print "There is an occurrence ending at t_{d+m}"
      end
end
```

If procedure $LandauVishkin(P,T)$ is implemeted as it is, it takes $O(mn)$ time. However, Landau and Vishkin shows that the part ($) can be computed in $O(1)$ time if the suffix tree associated with $T \cdot P$ is already constructed, where $T \cdot P$ denotes the concatenation of T and P. Note that a suffix tree associated with a string of length n is constructed in $O(n)$ time for an alphabet of fixed size and in $O(n \log n)$ time for a general alphabet [3, 9]. Using this technique, $LandauVishkin(P,T)$ works in $O(kn)$ time for an alphabet of fixed size and in $O((k+\log m)n)$ time for a general alphabet where the time for the construction of a suffix tree is included. Refer [8] for details.

3 Approximate matching with don't cares

In this section, we describe serial and parallel algorithms for the k differences problem with don't cares. For two characters p and q, we write $p \sim q$ if $p = q$, $p =$'$*$', or $q =$'$*$' holds, For two strings $s = s_1 \cdots s_j$ and $u = u_1 \cdots u_k$, we write $s \sim u$ if $(\forall i)(s_i \sim u_i)$ and $j = k$ hold.

First, note that procedure $LandauVishkin(P,T)$ works correctly for strings with don't cares if the part $p_{row+1} = t_{row+1+d}$ in ($) is replaced by $p_{row+1} \sim t_{row+1+d}$. However, suffix trees can not be used to compute this modified part efficiently. Thus, we use a table $W[r,j]$ instead of a suffix tree.

3.1 Utilization of a table

Let M be an integer where the value of M is to be determined later. We assume without loss of generality that M divides m. Let P^r denotes the substring $p_{(r-1)M+1}p_{(r-1)M+2}\cdots p_{rM}$ of P. Let T^j denotes the substring $t_jt_{j+1}\cdots t_{j+M-1}$ of T. If $j+M-1>n$, T^j denotes the empty string. Then, $W[r,j]$ denotes the maximum number h such that $P^r \cdot P^{r+1}\ldots P^{(r+h-1)} \sim T^j \cdot T^{j+M}\ldots T^{j+(h-1)M}$ holds. If such h does not exist, $W[r,j]=0$.

If $W[r,j]$'s are already computed for all $1 \le r \le \frac{m}{M}$ and $1 \le j \le n$, the part corresponding to (\$) can be computed efficiently by the following:

```
while M does not divide row + 1 and row < m
and row + d < n and p_{row+1} ~ t_{row+d+1} do
  row ← row + 1;
if row < m and row + d < n then row ← row + M × W[(row+1)/M, row + 1 + d];
while row < m and row + d < n and p_{row+1} ~ t_{row+d+1} do row ← row + 1;
```

Let (#) denote the above part and let $ModifiedLV(P,T)$ denote the modified procedure of $LandauVishkin(P,T)$ where (\$) is replaced by (#). Then, it is easy to see that the following proposition holds (see Figure 1).

[**Proposition 1**] Assume that a table $W[r,j]$ is already constructed. Then, $ModifiedLV(P,T)$ solves k differences problem with don't care characters in $O(knM)$ time.

3.2 Construction of the table

The table $W[r,j]$ can be constructed using the convolution based algorithm by Fisher and Paterson. The following procedure constructs the table.

```
Procedure MakeTable(P,T)
begin
  for all r such that 1 ≤ r ≤ m/M do                                    -(a)
    begin
      FisherPaterson(P^r, T);
      for all j (1 ≤ j ≤ n) do
        if P^r ~ T^j then W[r,j] ← 1 else W[r,j] ← 0
    end;
  for all j such that 1 ≤ j ≤ n − M + 1 do                              -(b)
    Using the list ranking technique, compute W[r, M(r − 1) + j]'s for all r;
  for all r such that 2 ≤ r ≤ m/M do                                    -(c)
    for all j such that 1 ≤ j ≤ M do
      Using the list ranking technique, compute W[h, M(h − 1) + j]'s for all h ≥ r
end
```

[**Proposition 2**] $MakeTable(P,T)$ computes the table $W[r,j]$ in $O(\frac{mn}{M}\log|\Sigma|\log^2 M\log\log M)$ time.

(Proof) Since it is easy to see that the procedure computes the table correctly, we consider the time complexity.

Fisher and Paterson's convolution based algorithm computes all occurrences of a pattern of length q in a text of length p with don't cares in $O(p\log|\Sigma|\log^2 q\log\log q)$ time [1, 5]. Thus,

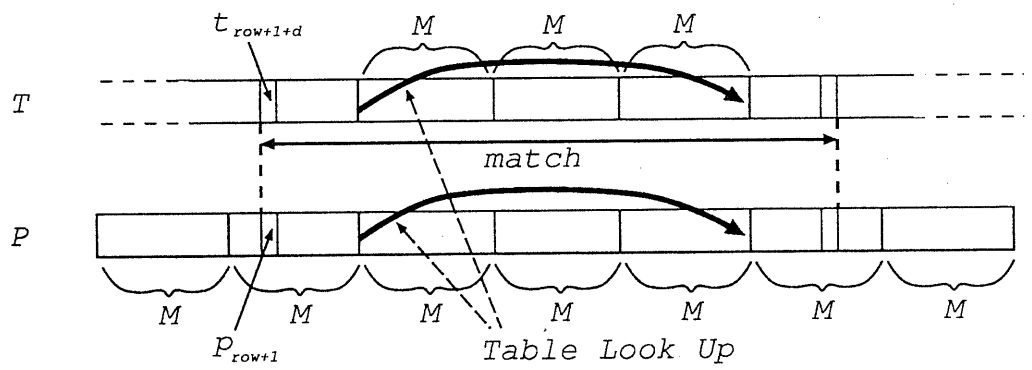

Figure 1: Utilization of the table $W[r,j]$.

$O(n \log|\Sigma| \log^2 M \log\log M)$ time is required per $FisherPaterson(P^r,T)$. Since 'for loop' of (a) is repeated $\frac{m}{M}$ times, the total time required for (a) is $O(\frac{mn}{M} \log|\Sigma| \log^2 M \log\log M)$.

Since list ranking can be done in linear time, part (b) takes $O(n \times \frac{m}{M}) = O(\frac{mn}{M})$ time and part (c) takes $O(\frac{m}{M} \times M \times \frac{m}{M}) = O(\frac{m^2}{M})$ time. Thus, the total time required for $MakeTable(P,T)$ is $O(\frac{mn}{M} \log|\Sigma| \log^2 M \log\log M)$ time. □

Letting $M = \sqrt{\frac{m}{k}}$ and combining propositions 1 and 2, we get the following theorem.

[Theorem 3] The k differences problem with don't care characters can be solved in $O(\sqrt{km}\, n \log|\Sigma| \log^2 \frac{m}{k} \log\log \frac{m}{k})$ time.

3.3 Parallel algorithm

In [8], a parallel version of procedure $LandauVishkin(P,T)$ is described. It works in $O(k)$ time using $O(n)$ processors except the construction of the suffix tree. Here, we consider a parallel version of our algorithm.

First, we consider procedure $ModifiedLV(P,T)$. Since the other parts are not modified from $LandauVishkin(P,T)$, We may consider part (#) only. It is easy to see that part (#) can be done in $O(\log M)$ time using $O(M)$ processors per execution. Thus, $ModifiedLV(P,T)$ works in $O(k \log M)$ time using $O(nM)$ processors.

Next, we consider the construction of the table. For a text string of length p and a pattern string of length q, the exact matching with don't cares can be done in $O(\log q)$ time using $O(p \log|\Sigma| \log q \log\log q)$ processors [6]. Thus, part (a) can be done in $O(\log M)$ time using $O(\frac{mn}{M} \log|\Sigma| \log M \log\log M)$ processors. It is easy to see that $O(\frac{mn}{M})$ processors and $O(\log m)$ time are sufficient for parts (b) and (c) even if a simple parallel list ranking algorithm is used. Thus, the table $W[r,j]$ can be constructed in $O(\log m)$ time using $O(\frac{mn}{M} \log|\Sigma| \log M \log\log M)$ processors. Letting $M = \sqrt{\frac{m}{k}}$, $ModifiedLV(P,T)$ can be done in $O(k \log m)$ time using $O(\sqrt{\frac{m}{k}}\, n)$ processors and the construction of the table can be done in $O(k \log m)$ time using $O(\sqrt{\frac{m}{k}}\, n \log|\Sigma| \log \frac{m}{k} \log\log \frac{m}{k})$ processors. Therefore, the following theorem holds.

[**Theorem 4**] The k differences problem with don't care characters can be solved in $O(k \log m)$ time using $O(\sqrt{\frac{m}{k}}\, n \log|\Sigma| \log\frac{m}{k} \log\log\frac{m}{k})$ processors on a CRCW-PRAM.

4 Extensions

In motif search, more complex patterns are used [4]. For example, the number of consecutive don't care characters is sometimes specified instead of repeating don't care characters. For an another example, such a description as ⟨A C⟩ is sometimes used, which denotes a character which matches a character 'A' or 'C'. In this section, we show that the k differences problem can be solved in $o(mn)$ time for small k even if such extented patterns are used.

4.1 Approximate matching with integer characters

In this subsection, we consider the k *differences problem with integer characters* in which any positive integer number may appear as a character in a pattern string. For example, 'ab1db3c' denotes 'ab*db***c'. We assume without loss of generality that two consecutive characters do not appear in a pattern string. Of course, the k differences problem with integer characters can be treated by expanding an input pattern string to a string in which each integer is replaced by consecutive don't care characters. However, the expanded string may become much longer than the input pattern string. Thus, we describe an algorithm for the k differences problem with integer characters by modifying the serial algorithm described in Section 3.

First, we describe how to modify the part of the construction of the table. The table $W[r,j]$ is defined in the same way except P^r is replaced by $(P^r)'$, where s' denotes the expanded string of s. Since the length of $(P^r)'$ may be $O(n)$, the time complexity of part (a) increases to $O(\frac{mn}{M} \log|\Sigma| \log^2 n \log\log n)$. Since it is sufficient for parts (b) and (c), the total time required for the construction of the table is $O(\frac{mn}{M} \log|\Sigma| \log^2 n \log\log n)$.

Next, we describe how to modify procedure $ModifiedLV(P,T)$. In the modified procedure, p_{row} and row are considered not for P but for P'. Note that the size of the table $L[d,e]$ remains $O(n)$. Since it is enough to modify part (#) only, we consider part (#) only. If p_{row+1} is the first character of $(P^r)'$ for some r, the table is looked up. If p_{row+1} is not '*', it is processed in the same way as (#). If p_{row+1} is '*', row jumps to the next position of P' such that $p_{row+1} \neq$'*'. It can be done in $O(1)$ time if appropriate preprocessing is done. It is easy to see that this preprocessing can be done in $O(n)$ time. Thus, the total time required for the modified procedure is $O(knM)$. Therefore, letting $M = \sqrt{\frac{m}{k}}$, the k difference problem with integer characters can be solved in in $O(\sqrt{km}\, n \log|\Sigma| \log^2 n \log\log n)$ time.

4.2 Approximate matching with generalized string patterns

Abrahamson considered the *generalized string matching* problem [1], which was a generalization of the string matching with don't cares. In the generalized string matching, such expressions as $\langle x_1\, x_2\, \cdots\rangle$ and $[x_1\, x_2\, \cdots]$ may appear in a pattern string. $\langle x_1\, x_2\, \cdots\rangle$ denotes a character which matches any character of $x_1, x_2, \cdots$. $[x_1\, x_2\, \cdots]$ denotes a character which matches any character except $x_1, x_2, \cdots$. Similar expressions sometimes appear in motif search. While Abrahamson considered the exact matching problem, we consider the *generalized string matching problem with k differences.*

The algorithm is almost the same as the serial algorithm described in Section 3. Only the difference is that Abrahamson's generalized string matching algorithms are used in place of Fisher and Patterson's algorithm. Here, m denotes the length of the expression of the pattern

and we assume that $m \leq n$. Abrahamson described an $O(\sqrt{m}\, n\, polylog(m))$ time algorithm for a general alphabet. Using a similar discussion as in Section 3 and letting $M = (\frac{m}{k})^{\frac{2}{3}}$, we can obtain an $O(k^{\frac{1}{3}}\, m^{\frac{2}{3}}\, n\, polylog(m))$ time algorithm for the generalized string matching with k differences. For a fixed size alphabet, Abrahamson described an $O(n\, polylog(m))$ time algorithm. Using a similar discussion as in Section 3, we can obtain an $O(\sqrt{km}\, n\, polylog(m))$ time algorithm for the generalized string matching with k differences for a fixed size alphabet.

5 Concluding remarks

This paper have presented algorithms for generalized approximate string matching problems. Although the algorithms work in $o(nm)$ time for small k, they are not practical since the convolution based exact matching algorithms are not practical. However, for a small size patterns, Abrahamson describes a practical algorithm for the generalized string matching [1]. It can also be used for the string matching with don't cares. Using this algorithm, practical algorithms for generalized approximate string matching might be developed.

Another problems are remained for the presented algorithms. The most important one is that the space complexities of algorithms are at least $O(\sqrt{\frac{m}{k}}\, m)$. Thus, more space economical algorithms should be developed. Of course, more efficient algorithms should be developed since the presented algorithms do not seem to be optimal.

References

[1] K. Abrahamson. "Genaralized string matching". *SIAM Journal on Computing*, Vol. 16, pp. 1039–1051, 1987.

[2] A. Amir and G. Landau. "Fast parallel and serial multidimensional approximate array matching". *Theoretical Computer Science*, Vol. 81, pp. 97–115, 1991.

[3] A. Apostolico, C. Iliopoulos, G. M. Landau, B. Schieber, and U. Vishkin. "Parallel construction of a suffix tree with applications". *Algorithmica*, Vol. 3, pp. 347–365, 1988.

[4] C. Branden and J. Tooze. *"Introduction to Protein Structure"*. Garland Publishing Inc., New York, 1991.

[5] M. Fisher and M. Paterson. "String matching and other products". In *Complexity of Computation (SIAM-AMS Proceedings)*, volume 7, pp. 113–125, 1974.

[6] Z. Galil and R. Giancarlo. "Data structures and algorithms for approximate string matching". *Journal of Complxity*, Vol. 4, pp. 33–72, 1988.

[7] Z. Galil and K. Park. "An improved algorithm for approximate string matching". *SIAM Journal on Computing*, Vol. 19, pp. 989–999, 1990.

[8] G. M. Landau and U. Vishkin. "Fast parallel and serial approximate string matching". *Journal of Algorithms*, Vol. 10, pp. 157–169, 1989.

[9] P. Weiner. "Linear pattern matching algorithms". In *Proceedings of IEEE Symposium on Switching and Automata Theory*, pp. 1–11, 1973.