アルゴリズム 43－5
（1995．1．23）

可変長ギャップ付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
群馬大学 工学部 情報工学科

DNA 配列やアミノ酸配列においては、あるパターンの後に数十文字のギャップがあってから別なパターンが現れる場合がある。そのような文字列を検索するためには指定された範囲長の任意文字列とマッチすることのできる記号を含んだパターンに対する近似マッチングが必要となる。本稿では、そのための動的計画法に基づく $O(mn\log n)$ 時間アルゴリズム、および、Landau-Vishkin アルゴリズムに基づく $O((k+\log n)gn)$ 時間アルゴリズムを示す。なお、m はパターン長、n はテキスト長、g は可変長ギャップ記号の個数である。

Approximate String Matching with Variable Length Don't Cares

Tatsuya Akutsu
Department of Computer Science, Gunma University
1-5-1 Tenjin, Kiryu, Gunma 376 Japan
e-mail: akutsu@cs.gunma-u.ac.jp

This paper studies an approximate string matching problem, in which a pattern string may contain variable length don't care characters. A variable length don't care character can match any substring whose length is in a specified range. This problem is important for searching DNA sequences or amino acid sequences. This paper presents an $O(mn\log n)$ time algorithm and an $O((k+\log n)gn)$ time algorithm, where m denotes the length of a pattern, n denotes the length of a text and g denotes the number of variable length don't cares.

1 Introduction

Searching for DNA or amino acid sequences which are similar to a given pattern string is very important in molecular biology. In fact, a lot of programs and algorithms have been developed. Most of them are based on alignment of strings or approximate string matching. However, they do not seem to be adequate in some cases. For example, the DNA pattern TATA (known as TATA box) is a common promoter that often appears after the pattern CAATCT (known as CAAT box) within 30 to 50 spaces [2, 5]. To find strings containing such a pattern, string matching with *variable length don't cares* is required, where a variable length don't care character can match any substring whose length is in a specified range. Moreover, exact matching is not sufficient but approximate string matching is required, since the patterns can appear with some probability of error. Thus this paper studies the problem of approximate string matching with variable length don't cares.

Here, we briefly review the previous works. Exact string matching with don't cares was studies by Fisher and Paterson [3]. A lot of studies have been done for approximate string matching [4]. Myers and Miller studied approximate string matching of regular expressions [6], and Zhang, Shasha and Wang studied approximate tree matching with variable length don't cares [7]. These two works are close to our problem. However, in their works, the range of the length of a substring matching to a don't care character can not be specified. Exact string matching with variable length don't cares was studied by Manber and Baeza-Yates [5]. In their work, the range can be specified, but approximate matching was not considered. We also developed an algorithm for approximate string matching with don't cares [1]. However, variable length don't cares can not be treated.

2 Description of the problem

First, we describe the *k-differences problem* [4], which is a conventional problem of approximate string matching. 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, where we assume that $|\Sigma|$ is bounded by a constant. 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 the minimum number of differences between the pattern string P and any substring of the text string T ending at t_j, is less than or equal to k, 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 a 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:

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

Next, we introduce a *variable length don't care character* '$*_{i-j}$', which matches any substring of T with length i to j $(0 \leq i \leq j)$. In approximate string matching with variable length don't cares, variable length don't care characters may appear in P. Then, the problem is defined in the same way as the k-differences problem, where each don't care character must match a substring whose length is in a specified range. Note that errors for don't cares are not allowed. In this paper, g denotes the number of variable length don't cares.

Example 2: P = bcd$*_{3-5}$gi occurs at position 11 of T = abedeeeeghij with differences 2 by the following correspondence:

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

$P = *_{3-4}$ab can not occur at any position of T = ab with any differences since errors for $*_{3-4}$ are not allowed.

3 A simple algorithm and its improvement

3.1 A simple algorithm

A simple $O(mn)$ time algorithm based on the dynamic programming technique is well known for the conventional approximate string matching problem [4]. Here we briefly overview the algorithm. It computes the matrix $D[i,j]$, where $D[i,j]$ shows the minimum number of differences between $p_1 \cdots p_i$ and any substring of T ending at t_j. $D[i,j]$ is determined by

$$D[i,j] = \min(\, D[i-1,j]+1,\ D[i,j-1]+1,\ D[i-1,j-1]+1-\delta(p_i,t_j)\,),$$

where $\delta(p_i,t_j) = 1$ if $p_i = t_j$ and $\delta(p_i,t_j) = 0$ otherwise. Since the above expression can be evaluated in $O(1)$ time and the size of the matrix is $O(mn)$, the simple algorithm works in $O(mn)$ time.

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

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

We can solve our problem executing the simple algorithm $g+1$ times in the following way. Let p_{m_i} $(1 \leq i \leq g)$ denote the i-th don't care character in P. The i-th execution is

done for the subpattern $p_{m_{i-1}+1} \cdots p_{m_i-1}$ and T, where the matrix is re-initialized using the result of the $(i-1)$-th execution. The following procedure describes the modified algorithm.

```
Procedure SimpleMatch(P, T, k)
begin
  for j = 0 to n do D[0, j] ← 0;
  for i = 0 to m_1 − 1 do D[i, 0] ← i;
  for i = m_1 to m do D[i, 0] ← ∞;
  for i = 1 to m do
    if p_i = *_{p−q} then
      begin
        for j = 0 to n do D[i, j] ← ∞;
        for j = 0 to n do                                              (#)
          for h = j + p to j + q do                                    (#)
            if h ≤ n and D[i − 1, j] < D[i, h] then D[i, h] ← D[i − 1, j]   (#)
      end
    else
      for j = 1 to n do
        D[i, j] ← min(D[i − 1, j] + 1, D[i, j − 1] + 1, D[i − 1, j − 1] + 1 − δ(p_i, t_j));
  for j = 0 to n do
    if D[m, j] ≤ k output "Match at j"
end
```

The correctness of the algorithm is almost trivial. The time complexity is $O(gn^2)$, since $O(n^2)$ time is required per execution of part (#) if $q-p$ is $O(n)$.

3.2 Improvement

The simple dynamic programming algorithm can be improved if we modify the order of updating $D[i,h]$'s in part (#). In the simple algorithm, the values of $D[i-1,j]$'s are used from $j=0$ to n. In the improved algorithm, the values of $D[i-1,j]$'s are used in ascending order, that is, the values of $D[i-1,j]$'s are used in the following order

$$D[i-1,\pi(0)] \leq D[i-1,\pi(1)] \leq \cdots \leq D[i-1,\pi(n)] ,$$

where $\pi(i)$ means a permutation of $(0,1,\cdots,n)$. Then, only $D[i,h]$'s such that $D[i,h] = \infty$ need be updated. Moreover, the following property holds (see Fig. 1).

Proposition 1: Only a consecutive part of $D[i,h]$'s is updated by each $D[i-1,j]$.

Thus, using an adequate sorting algorithm (e.g. $O(n)$ time bucket sort, or $O(n \log n)$ time merge sort) and using an adequate data structure (e.g. balanced binary trees) for maintaining intervals with $D[i,j] = \infty$, we get the following theorem.

Theorem 1: The k-differences problem with variable length don't cares can be solved in $O(mn \log n)$ time.

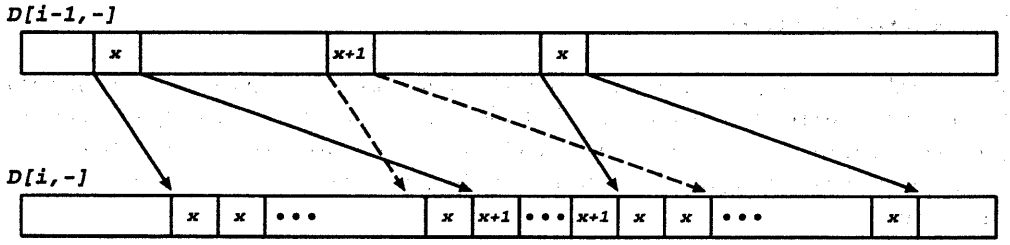

Figure 1: Intervals updated by $D[i-1, j]$.

4 A Landau-Vishkin type algorithm

4.1 Landau-Vishkin algorithm

Landau and Vishkin have developed an $O(kn)$ time algorithm for the conventional approximate string matching problem [4], which is much faster than the simple algorithm for $k = o(m)$. Their algorithm computes the same information 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 $j - i = d$. In the case of Example 3, $L[3, 0] = 0$, $L[3, 1] = 3$ and $L[3, 2] = 4$. Note that the value of $D[i, j]$ such that $j - i = d$ grows monotonically as i grows. Thus, for the k differences problem, 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 required to be computed is $O(kn)$. Moreover, Landau and Vishkin showed that $L[d, e]$'s could be computed in $O(kn)$ time using the suffix tree.

4.2 Modification

Landau-Vishkin algorithm can be modified for approximate string matching with variable length don't cares. For that purpose, we employ a similar approach as in section 3: Landau-Vishkin algorithm is executed $g + 1$ times, where $L[d, e]$'s are re-initialized based on the result of the previous execution. Although details such as the method for re-initialization are a little complicated, the modification is straightforward, where the technique described in subsection 3.2 is used too. Thus details are omitted in this paper.

Here we briefly analyze the time complexity. Since Landau-Vishkin algorithm is executed $g + 1$ times, it takes $O(gkn)$ time. Moreover, $O(gn \log n)$ time is required for re-initialization, because $O(n \log n)$ time is required for re-initialization before each execution of Landau-Vishkin algorithm. Thus we obtain the following theorem.

Theorem 2: The k-differences problem with variable length don't cares can be solved in $O((k + \log n)gn)$ time.

Note that if $(k + \log n)g = o(m)$, this time complexity is $o(mn)$. In most practical cases, g is very small (e.g. 1) and k is much smaller than m. Thus the algorithm is expected to be much faster than the simple algorithm in such cases.

5 Conclusion

We have shown an $O(mn \log n)$ time algorithm and an $O((k + \log n)gn)$ time algorithm for approximate string matching with variable length don't cares. Although we have not yet succeeded, we believe that $O(\log n)$ factor can be removed using more sophisticated technique or data structure. However it is much more interesting to develop much faster algorithms.

Acknowledgement

This research was partially supported by the Grant-in-Aid for Scientific Research on Priority Areas, "Genome Informatics", of the Ministry of Education, Science and Culture of Japan.

References

[1] T. Akutsu, "Approximate string matching with don't care characters," *Proc. 4th Symp. Combinatorial Pattern Matching*, Lecture Notes in Computer Science 807, pp. 240-249, 1994.

[2] C. Branden and J. Tooze, *Introduction to Protein Structure*, Garland Publishing, 1991.

[3] M. Fisher and M. Paterson, "String matching and other products," *Complexity of Computation*, SIAM-ACM Proceedings 7, pp. 113-125, 1974.

[4] G. M. Landau and U. Vishkin, "Fast parallel and serial approximate string matching," *J. Algorithms*, Vol. 10, pp. 157-169, 1989.

[5] U. Manber and R. Baeza-Yates, "An algorithm for string matching with a sequence of don't cares," *Information Processing Letters*, Vol. 37, pp. 133-136, 1991.

[6] E. W. Myers and W. Miller, "Approximate matching of regular expressions," *Bulletin of Mathematical Biology*, Vol. 51, pp. 5-37, 1989.

[7] K. Zhang, D. Shasha and J. Wang, "Approximate tree matching in the presence of variable length don't cares," *J. Algorithms*, Vol. 16, pp. 33-66, 1994.