On the Preprocessing Algorithm Used in the Boyer-Moore Algorithm for String Searching

ICHIRO SEMBA*

We consider the preprocessing algorithm for Boyer-Moore string-searching.

This paper proposes a natural preprocessing algorithm coming to our mind. The average running time per pattern of length m is proved to be $O(m+m^2/q)$ under the assumption that q possible characters appear uniformly.

The computer experiments show that the average running time per pattern of our algorithm is better than previous algorithms proposed.

1. Introduction

We consider an algorithm for computing the table of pattern shifts used in the Boyer-Moore algorithm[1] for string searching. We assume that the pattern of length m is given by the array pattern[1:m] and q possible characters are used. Knuth[2] has shown the $O(m)$ algorithm and Rytter[3] and Noshita[4] have revised Knuth's algorithm.

In this paper we will present a natural algorithm coming to our mind. Even though it is $O(m^2)$ algorithm, the average running time per pattern of length m is proved to be $O(m+m^2/q)$ under the assumption that q possible characters appear uniformly. Practically we are able to expect that $m<q$. It follows that the average running time of our algorithm is linear.

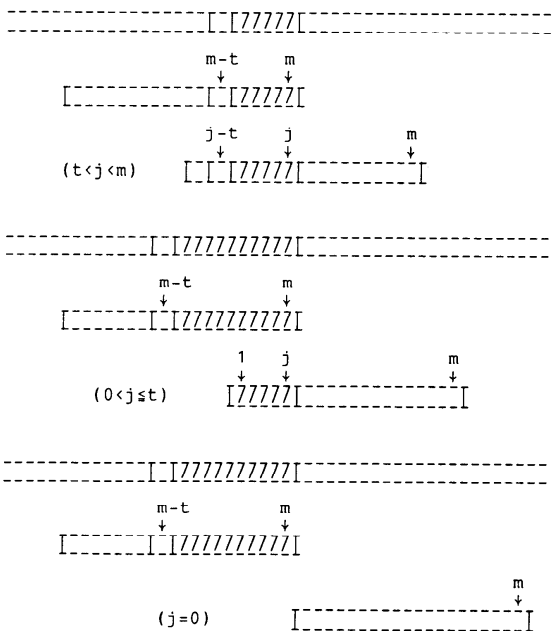
The computer experiments indicate that the average running time per pattern of our algorithm is better than previous algorithms proposed.

2. Algorithm

When a mismatch occurs at the position $m-t$ $(1\leqq t\leqq m-1)$ in the pattern, the pattern can be moved to the right. In order not to fail to find the pattern in the text (processing string), we have to check the substrings in the pattern such that pattern $[j-t+1:j]$=pattern $[m-t+1:m]$ $(t<j<m)$or pattern $[1:j]$=pattern $[m-j+1:m]$ $(0<j\leqq t)$ or empty string $(j=0)$.

This substring is represented by the rightmost position. The set of the rightmost positions is denoted by $S(t)$ $(1\leqq t\leqq m-1)$ and defined as follows.

$S(t)=\{j \mid j=0$
or $(0<j\leqq t$ and
pattern[1:j]=pattern[$m-j+1:m$])
or $(t<j<m$ and
pattern[$j-t+1:j$]=pattern[$m-t+1:m$])$\}$



The pattern shift is determined by choosing the maximum element j such that $(j\in S(t)$ and $j\leqq t)$ or $(j\in S(t)$ and $t<j$ and pattern[$j-t$]$\neq$pattern[$m-t$]). Let the maximum element be denoted by $p[t]$ $(1\leqq t\leqq m-1)$. It is given as follows.

$p[t]=\max\{j \mid (j\in S(t)$ and $j\leqq t)$
or $(j\in S(t)$ and $t<j$
and pattern[$j-t$]$\neq$pattern[$m-t$])$\}$

Therefore the pattern can be moved to the right by $m-p[t]$.

Example. Three patterns are taken by way of ex-

*College of General Education, Ibaraki University, 2-1-1 Bunkyo, Mito, 310, Japan

amples.

(1) pattern[1:5]=aaaaa.
$S(1)=S(2)=S(3)=S(4)=\{0, 1, 2, 3, 4\}$
$p[1]=1$, $p[2]=2$, $p[3]=3$ and $p[4]=4$

(2) pattern[1:5]=abcde.
$S(1)=S(2)=S(3)=S(4)=\{0\}$
$p[1]=p[2]=p[3]=p[4]=0$

(3) pattern[1:10]=bcaacbcabc.
$S(1)=\{2, 5, 7\}$ $p[1]=5$
$S(2)=\{2, 7\}$ $p[2]=7$
$S(t)=\{2\}$ $(3\leqq t\leqq 9)$ $p[t]=2$ $(3\leqq t\leqq 9)$

Let us denote the table of pattern shifts by $D[j]$ $(1\leqq j\leqq m-1)$. The definition of D given in [2] is:

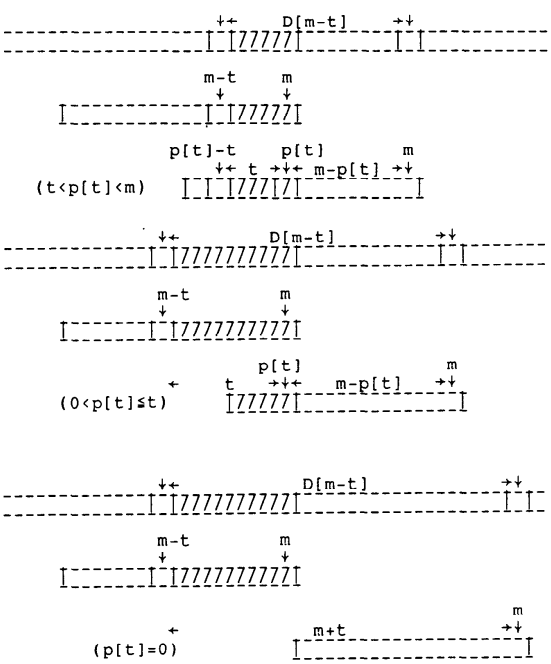
$$D[j]=\min\{s+m-j \mid s\geqq 1 \text{ and } (s\geqq j \text{ or pattern}[j-s]\neq\text{pattern}[j]) \text{ and } ((s\geqq i \text{ or pattern}[i-s]=\text{pattern}[i]) \text{ for } j<i\leqq m))\}$$

The table $D[j]$ $(1\leqq j\leqq m-1)$ can be represented by the table $p[t]$ $(1\leqq t\leqq m-1)$.

Property 2.1 For $1\leqq t\leqq m-1$,

$$D[m-t]=t+m-p[t].$$

Proof. By the following figures, the above result is easily understood.



This property means that the table $D[j]$ $(1\leqq j\leqq m-1)$ obtained by computing the table $p[t]$ $(1\leqq t\leqq m-1)$.

Now we will present the algorithm determining the table $p[t]$ $(1\leqq t\leqq m-1)$. The following property is important.

Property 2.2 For $1\leqq t\leqq m-2$,

$$S(t+1)=\{j \mid (j\in S(t) \text{ and } j\leqq t) \text{ or } (j\in S(t) \text{ and } t<j \text{ and pattern}[j-t]=\text{pattern}[m-t])\}$$

Proof. By the definition of the set $S(t+1)$, it is obvious.

We note that the right-hand side of the above equation is similar to the definition of the table $p[t]$. This fact means that both $S(t+1)$ and $p[t]$ are derived from the set $S(t)$ at once.

Example. Let pattern[1:12]=bbaacbcbaacb.

j	1	2	3	4	5	6	7	8	9	10	11	12
pattern[j]	*b*	*b*	*a*	*a*	*c*	*b*	*c*	*b*	*a*	*a*	*c*	*b*

$S(1)=\{0, 1, 2, 6, 8\}$
$S(2)=\{0, 1, 6, 8\}$ $p[1]=2$
$S(3)=\{0, 1, 6\}$ $p[2]=8$
$S(4)=\{0, 1, 6\}$ $p[3]=1$
$S(5)=\{0, 1, 6\}$ $p[4]=1$
$S(6)=\{0, 1\}$ $p[5]=1$
$S(7)=\{0, 1\}$ $p[6]=1$
$S(8)=\{0, 1\}$ $p[7]=1$
$S(9)=\{0, 1\}$ $p[8]=1$
$S(10)=\{0, 1\}$ $p[9]=1$
$S(11)=\{0, 1\}$ $p[10]=1$
$p[11]=1$

The following property will speed up the running time.

Property 2.3
If $\max\{j \mid j\in S(t)\}\leqq t$ for some t, then we have $S(t)=\ldots=S(m-1)$ and $p[t]=\ldots=p[m-1]=\max\{j \mid j\in S(t)\}$.

Proof. Obvious.

3. Implementation

Combining Property 2.1, 2.2 and 2.3, our algorithm is constructed.

```
create the set S(1); t:=1;
while max {j|j∈S(t)} >t do begin
  create the set S(t+1) and determine p[t];
  D[m-t]:=t+m-p[t]; t:=t+1;
end;
D[m-t]:=t+m-max {j|j∈S(t)};
  for k:=t+1 to m-1 do D[m-k]:=D[m-k+1]
  +1;
```

$$X[j]=\begin{cases}\max\{i \mid i=0 \text{ or } (i<j \text{ and pattern}[i]=\text{pattern}[j])\} & \text{if pattern}[j]=\text{pattern}[m].\\ \text{undefined} & \text{if pattern}[j]\neq\text{pattern}[m].\end{cases}$$

In our program, the set $S(1)$ is represented by the array $X[1{:}m]$. The element $X[j]$ $(1\leqq j\leqq m)$ is defined as above.

We note that $X[m]$ corresponds to $\max\{j \mid j\in S(1)\}$.

Example. Let pattern[1:12]=bbaacbcbaacb.

j	1	2	3	4	5	6	7	8	9	10	11	12
pattern[j]	b	b	a	a	c	b	c	b	a	a	c	b
$X[j]$	0	1				2		6				8

The process of creating the set $S(1)$ is given as follows.

```
k:=0;
for j:=1 to m-1 do begin
  if pattern[j]=pattern[m] then begin X[j]:=k;
  k:=j; end;
end;
X[m]:=k;
```

The process of creating the set $S(t+1)$ and determining $p[t]$ is described as follows.

```
k:=m; l:=X[m]; p[t]:=0;
while l>t do begin
  if pattern[m-t]=pattern[l-t] then begin
  X[k]:=l; k:=l;
  end else begin
  p[t]:=max {p[t], l}
  end;
  l:=X[l];
end;
p[t]:=max {p[t], l};
X[k]:=l;
```

We note that the array $x[1{:}m]$ represents the set $S(t+1)$ after the above process and $X[m]$ corresponds to $\max\{j \mid j\in S(t+1)\}$.

Now we will estimate the average running time per pattern of length m. We assume that q possible characters appear uniformly. The algorithm suggests that the number of character comparisons in the pattern is a good measure of the running time.

Let $t0=\min\{t \mid X[m]\leqq t\}$. By the fact that $|S(1)|\geqq |S(2)|\geqq\ldots\geqq|S(m-1)|$ and $t0\leqq m-1$, we can see that the number of character comparisons is bounded by

$$\begin{aligned}m+|S(1)|+|S(2)|+\ldots+|S(t0)|&\leqq m+|S(1)|t0\\ &\leqq m+|S(1)|(m-1).\end{aligned}$$

Let the probability such that $|S(1)|=k$ $(0\leqq k\leqq m-1)$ be denoted by $\text{Prob}(|S(1)|=k)$. We have

$$\text{Prob}(|S(1)|=k)=\binom{m-1}{k}(1-1/q)^{m-k-1}/q^k$$

The average number of character comparisons $A(q, m)$ is derived as follows.

$$\begin{aligned}A(q,m)&\leqq m+\sum_{k=1}^{m-1}(m-1)k\text{Prob}(|S(1)|=k)\\ &=m+(m-1)\left[\binom{m-1}{1}(1-1/q)^{m-2}/q\right.\\ &\quad+2\binom{m-1}{2}(1-1/q)^{m-3}/q^2+\cdots\\ &\quad\left.+(m-1)\binom{m-1}{m-1}(1-1/q)^0/q^{m-1}\right]\\ &=m+(m-1)[(m-1)(1/q)\{1/q+(1-1/q)\}^{m-2}]\\ &=m+(m-1)^2/q\end{aligned}$$

Thus we obtain the following theorem.

Theorem 1

If q possible characters appear uniformly, then the average running time per pattern of length m of our algorithm is $O(m+m^2/q)$.

We note that the average running time becomes linear when $m<q$.

Table 1 The average running time per pattern of length m under the assumption that q possible characters appear uniformly. (times in milliseconds)

q	m	patterns	Rytter's algorithm	Our algorithm	rate
3	4	16	0.37	0.25	0.66
2	5	32	0.50	0.31	0.62
2	6	64	0.56	0.40	0.72
2	7	128	0.62	0.46	0.75
2	8	256	0.69	0.53	0.76
2	9	512	0.79	0.61	0.77
2	10	1024	0.87	0.69	0.78
2	11	2048	0.94	0.76	0.81
2	12	4096	1.02	0.84	0.82
2	13	8192	1.10	0.91	0.83
2	14	16384	1.17	0.99	0.84
2	15	32768	1.24	1.05	0.84
2	16	65536	1.30	1.11	0.85
3	3	27	0.22	0.14	0.66
3	4	81	0.34	0.19	0.57
3	5	243	0.41	0.25	0.60
3	6	729	0.49	0.32	0.66
3	7	2187	0.56	0.38	0.67
3	8	6561	0.63	0.43	0.69
3	9	19683	0.70	0.49	0.70
4	3	64	0.21	0.12	0.57
4	4	256	0.32	0.17	0.54
4	5	1024	0.40	0.25	0.62
4	6	4096	0.46	0.30	0.64
4	7	16384	0.53	0.35	0.65

Table 2 The average running time per pattern of length m under the assumption that q possible characters appear uniformly. 1000 patterns are generated. (times in milliseconds)

q	m	patterns	Rytter's algorithm	Our algorithm	rate
8	4	1000	0.34	0.22	0.63
8	8	1000	0.55	0.35	0.64
8	16	1000	1.07	0.69	0.65
8	32	1000	2.08	1.33	0.64
8	64	1000	4.05	2.73	0.67
16	4	1000	0.29	0.16	0.56
16	8	1000	0.53	0.33	0.62
16	16	1000	1.02	0.72	0.70
16	32	1000	1.87	1.28	0.68
16	64	1000	3.92	2.70	0.69
32	4	1000	0.29	0.17	0.58
32	8	1000	0.52	0.33	0.63
32	16	1000	1.01	0.62	0.61
32	32	1000	1.93	1.24	0.64
32	64	1000	3.86	2.48	0.64
64	4	1000	0.30	0.17	0.58
64	8	1000	0.52	0.31	0.60
64	16	1000	1.03	0.69	0.66
64	32	1000	1.98	1.26	0.63
64	64	1000	3.85	2.42	0.62

4. Computer experiments

Computer experiments have been done and the average running time per pattern of length m has been measured under the assumption that q possible characters appear uniformly. All patterns of length m are generated for $q=2, 3, 4$ and the results are shown in Table 1. We have also generated 1000 patterns of length m at random for $q=8, 16, 32, 64$. Since Rytter's algorithm is better than Noshita's algorithm in the average running time, Rytter's algorithm and our algorithm have been compared. The results are shown in Table 2. These results indicate that our algorithm is better than Rytter's algorithm in the average running time.

These algorithms are written in PASCAL and have been executed on a MELCOM-COSMO 700 III.

Acknowledgement

The author would like to thank Prof. Nozaki for his valuable advices and hearty encouragements.

References

1. BOYER, R. S. and MOORE, J. S. A fast string searching algorithm, *Comm. ACM*, **20** (1977), 762–772.
2. KNUTH, D. E., MORRIS, J. H. and PRATT, V. R. Fast pattern matching in strings, *SIAM Journal on Computing*, **6** (1977), 323–350.
3. RYTTER, W. A correct preprocessing algorithm for Boyer-Moore string-searching, *SIAM Journal on Computing*, **9** (1980), 509–512.
4. NOSHITA, K. Fundamental Algorithm, *Information Science* **10** (1983), Iwanami-Koza, Iwanami Shoten Publishers.

(Received August 16, 1985; revised November 25, 1986)