On String Matching in Chunked Texts

Hannu Peltola and Jorma Tarhio

{hpeltola, tarhio}@cs.hut.fi

Department of Computer Science and Engineering
Helsinki University of Technology
P.O. Box 5400, FI-02015 HUT, Finland

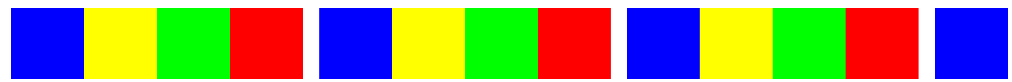
Overview

- Problem
- Data consists of *chunks*
- Brief history of previous solutions
- New algorithms
- Some experimental results
- Conclusions

Problem

- Exact pattern matching on strings: find all positions where a given pattern can be found in a text
- Text of length n: $T = t_1 t_2 \cdots t_n$
- Pattern of length m: $P = p_1 p_2 \cdots p_m$
- Texts are special: *chunked*

Texts are chunked

- Now texts consist of consecutive fixed-length chunks:  . . .
- Each byte position (, , ,) in every chunk has a character distribution of its own
- A chunk can also be interpreted as a character of a larger alphabet
- q is the probability that two randomly chosen bytes from text and pattern match
- Thierry Lecroq: Experiments on string matching in memory structures. *SPE*, **28**(5):561–568, 1998.

Some pattern matching algorithms

- Boyer–Moore (BM)
- Horspool (Hor) – shift is simplified: based on character that is aligned with the end of the pattern
- Sunday's Quick Search (QS) – shift is based on character that is after the end of the pattern
- Zhu–Takaoka, Baeza-Yates, etc. – shift is based on two or more characters

Implementation

- Already Boyer & Moore noticed that random text character rarely matches with the corresponding character in pattern
- So usually algorithms check one character and move forward – *skip loop*
- TBM = Tuned Boyer–Moore uses *ufast* skip loop (original implementation by Hume & Sunday)
- *Guard*: an additional test before comparison of the entire pattern

The speed of QS and Hor should be almost equal

- If characters are statistically independent of each other
- Expected shift length of Hor is $\frac{1-(1-q)^m}{q}$
- Expected shift length of QS is $\frac{1-(1-q)^{m+1}}{q}$
- When comparison is made forward; an example:

Example of the behavior of QS

a	a	a	a	b	a	a	a	a	b	a	a	a	a	b
$\dot{a}$	$\dot{a}$	$\dot{a}$	$\dot{a}$	$\overset{*}{a}$										
	$\dot{a}$	$\dot{a}$	$\dot{a}$	$\overset{*}{a}$	a									
		$\dot{a}$	$\dot{a}$	$\overset{*}{a}$	a	a								
			$\dot{a}$	$\overset{*}{a}$	a	a	a							
				$\overset{*}{a}$	a	a	a	a						
										$\dot{a}$	$\dot{a}$	$\dot{a}$	$\dot{a}$	$\overset{*}{a}$

- $\frac{m(m+1)}{2}$ comparisons per $2m$ characters in text – QS works here in $\mathcal{O}(nm)$

Peculiar behavior

- When comparison is made forward
- $P = a^m, T = (a^{m-1}b)^{n/m}$ — Hor works in $\mathcal{O}(n/m)$ and QS in $\mathcal{O}(nm)$
- $P = a^{m-4}ca^3, T = (ba^{m-2}cb)^{n/m}$ — QS works in $\mathcal{O}(n/m)$ but Hor in $\mathcal{O}(nm)$

Lecroq's data / short integers

SHORTS	symbols	max.freq.	zeros	$1/q$
1	256	1564	1559	248.25
2	44	12500	12500	32.00
Overall	256	14064	14059	86.01

- $2 \cdot 200000 = 400000$ bytes
- Regularities: $5 + i \cdot 32 \equiv$ `'\x00'`; $21 + i \cdot 32 \equiv$ `'\x40'`; $13 + i \cdot 64 \equiv 61 + i \cdot 64 \equiv$ `'\x10'`; $29 + i \cdot 64 \equiv 45 + i \cdot 64 \equiv$ `'\x90'`

Lecroq's data / doubles

DOUBLES	symbols	max.freq.	zeros	$1/q$
1	5	100152	3	2.00
2	215	6371	4	48.07
3	256	863	798	255.71
4	256	1344	1344	254.40
5	256	9667	9667	113.98
6	4	124889	124889	2.29
7	1	200000	200000	1.00
8	1	200000	200000	1.00
Overall	256	536705	536705	8.11

Lecroq's data and experiments

- Data was dumps from computer memory
- On shorts, TBM was fastest on short patterns and QS on long patterns
- On doubles, BM was fastest
- Lecroq did not consider the effects caused by chunks. He was more interested in the effect of the alphabet size
- When a potential match was found, it was checked that it ends on chunk border

What would work better

- Positions with no or little variation are challenging
- We could use two bytes so that at least the other byte would hit a position with rich varying content. We could also peek forward greedily to get longer shifts
- We could shift in a synchronized fashion (with chunk borders) and check the content of the most random byte position in the last chunk of the pattern

Fork(**$h, P = p_1 p_2 \cdots p_m, T = t_1 t_2 \cdots t_n$**)

/* Preprocessing */

for all $c \in \Sigma$ **do** $tmpd[c] \leftarrow m$
for $i \leftarrow 1$ **to** $m - 1$ **do** $tmpd[p_i] \leftarrow m - i$
shift $\leftarrow tmpd[p_m]; \quad tmpd[p_m] \leftarrow 0$
for all $c1 \in \Sigma$ **do**
  if $tmpd[c1] < h$ **then**
    for all $c2 \in \Sigma$ **do** $d[c1, c2] \leftarrow tmpd[c1]$
  else
    for all $c2 \in \Sigma$ **do** $d[c1, c2] \leftarrow m + h$
    for $i \leftarrow 1$ **to** h **do** $d[c1, p_i] \leftarrow m + h - i$
for $i \leftarrow 1$ **to** $m - h$ **do**
  if $tmpd[p_i] \geq h$ **then** $d[p_i, p_{i+h}] \leftarrow m - i$

/* Searching is on next slide */

Fork(**$h, P = p_1 p_2 \cdots p_m, T = t_1 t_2 \cdots t_n$)***

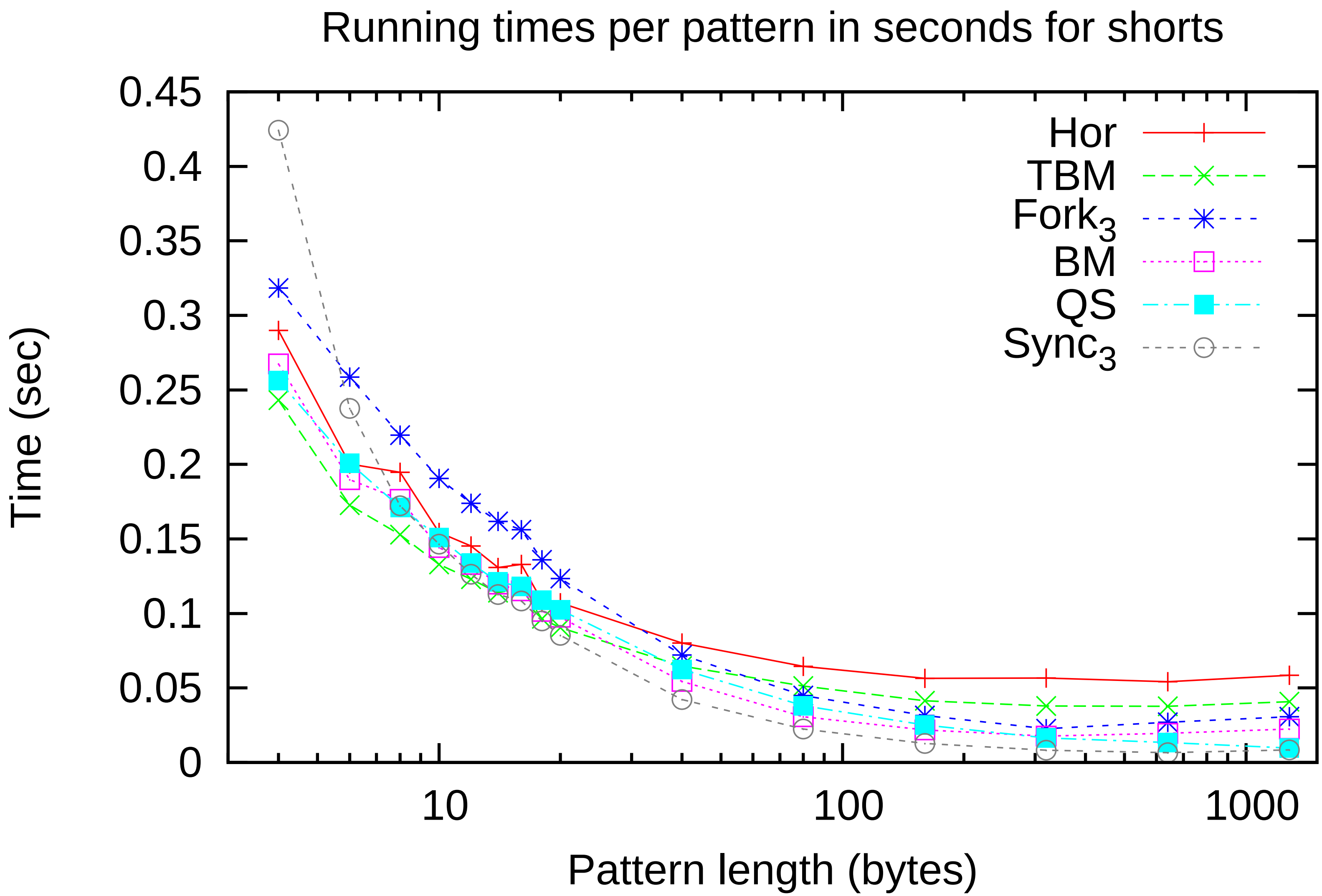
/* Searching */

$t_{n+1} \cdots t_{n+2*m} \leftarrow P + P$ /* Stopper */
$j \leftarrow m$
while $j \leq n$ **do**
  repeat $k \leftarrow d[t_j, t_{j+h}]$; $j \leftarrow j + k$ **until** $k = 0$
  if $j \leq n$ **then**
    if $t_{j-m+1} \cdots t_{j-1} = p_1 \cdots p_{m-1}$
      and j is a multiple of w **then**
        Report match
    $j \leftarrow j +$ *shift*

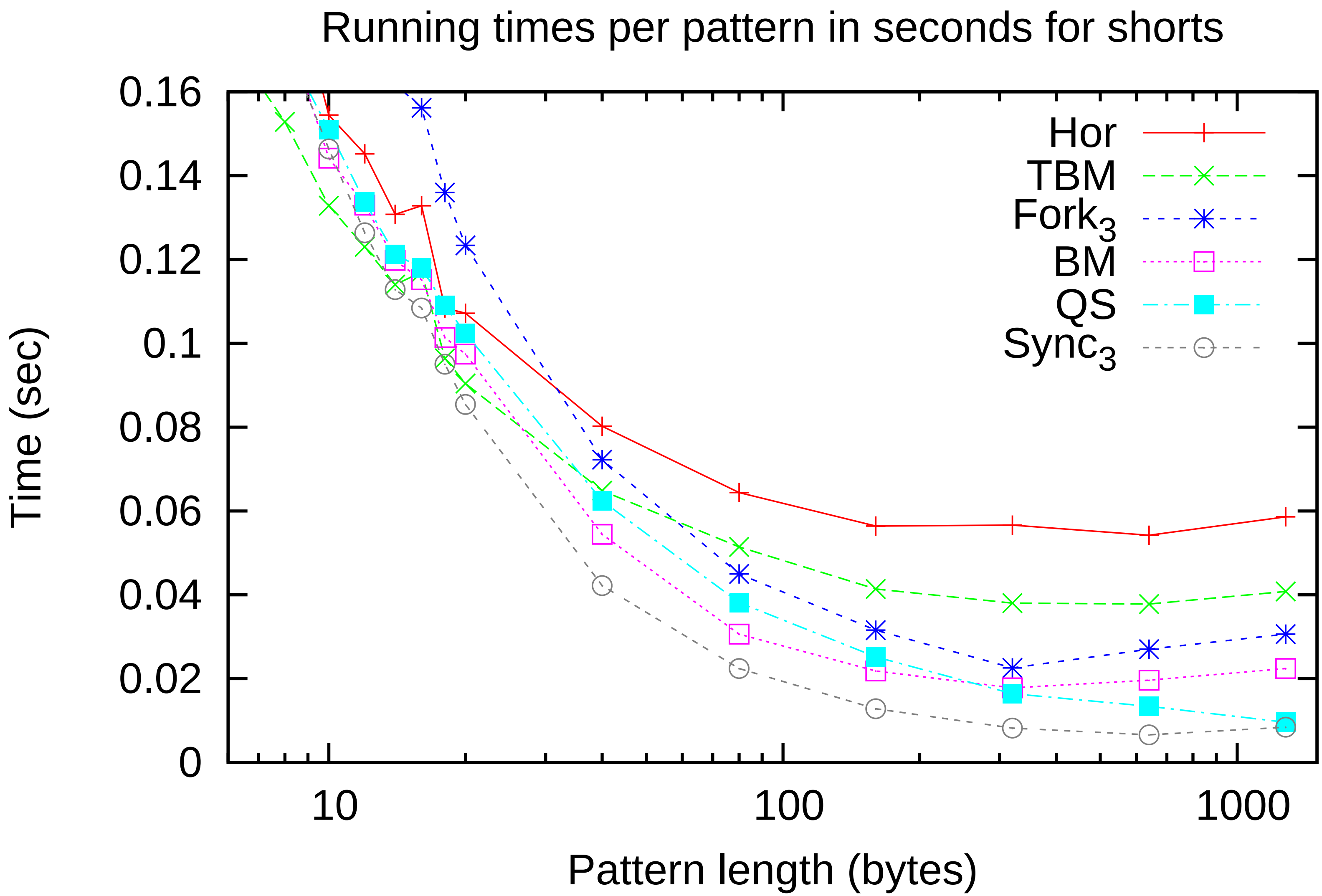
Sync($h, P = p_1p_2\cdots p_m, T = t_1t_2\cdots t_n$***)***

/* Preprocessing */
for all $c \in \Sigma$ **do** $d1[c] \leftarrow m$
for $i \leftarrow w - h$ **step** w **to** $m - h - 1$ **do**
  $d1[p_i] \leftarrow (m - h) - i$
/* Searching */
$s \leftarrow p_{m-h}$
$t_{n+1}..t_{n+m} \leftarrow s^m$ /* Stopper for inner while */
$j \leftarrow m$
while $j \leq n$ **do**
  while $t_{j-h} \neq s$ **do** $j \leftarrow j + d1[t_{j-h}]$
  if $t_{j-m+1}..t_j = P$ **then** Report match
  $j \leftarrow j + d1[s]$

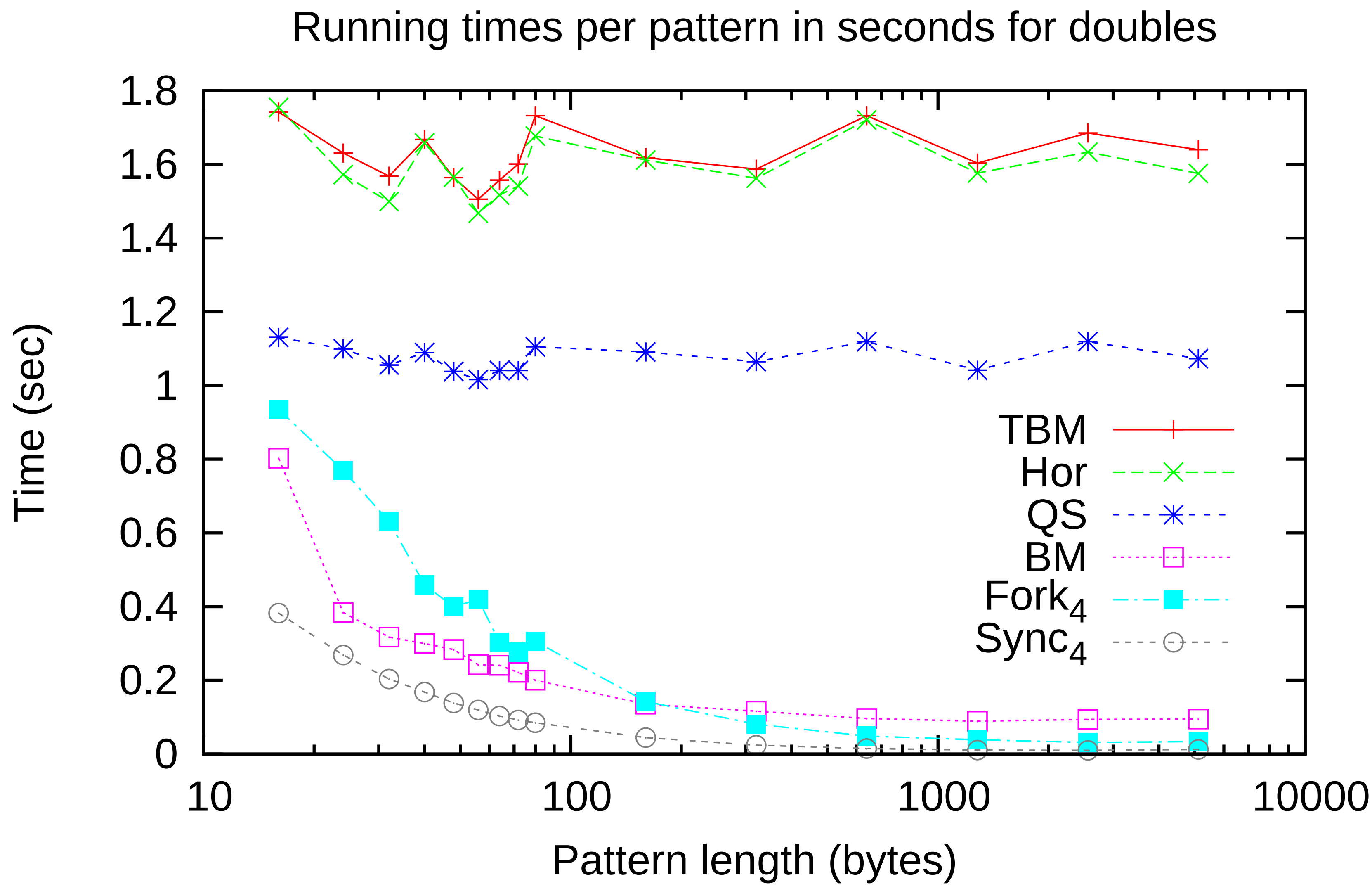
Results for shorts



Results for shorts (long patterns)



Results for doubles



Concluding remarks

- Test runs were repeated on two architectures: on Sparc and on AMD Athlon Thunderbird
- Library routine `memcmp` slower than explicit comparison
- On Sync parameter h corresponding smallest q works usually best
- On Fork the small values seem to be good for parameter h

Conclusions

- Choice of test position is sometimes crucial
- Skip loop improves speed in practice, if test character is not too common
- Instead of maximizing the average shift length, it is often faster to keep the skip loop running
- String matching results are data dependent
- e.g. chunked data can have very different effect on different algorithms