

Fast Exact String Matching Algorithms

Thierry Lecroq

Thierry.Lecroq@univ-rouen.fr

Laboratoire d'Informatique, Traitement de l'Information, Systèmes.

Part of this work has been done with Maxime Crochemore

LAW 2007

King's College London

February 8th and 9th, 2007



Outline

- 1 Introduction
- 2 Best Matching Shift
- 3 Hashing q -grams
- 4 Experimental results

Outline

- 1 Introduction
- 2 Best Matching Shift
- 3 Hashing q -grams
- 4 Experimental results

Exact String Matching

Problem

Find one or more generally all the occurrences of a pattern x of length m in a text y of length n .

Both x and y are build on an alphabet Σ of size σ .

Exact String Matching

Solutions

Many !!

See <http://monge.univ-mlv.fr/~lecroq/string>

Most famous : Knuth–Morris–Pratt and Boyer–Moore, 1977

Exact String Matching

Sliding window mechanism

KMP : from left to right (\longrightarrow)

BM : from right to left (\longleftarrow)

Boyer-Moore

Typical situation



A suffix u of the pattern is found and a mismatch occurs between a character a in the pattern x and a character b in the text y .

Matching shift



The matching shift consists in aligning the substring $u = x[i + 1..m - 1] = y[i + j + 1..j + m - 1]$ with one of its reoccurrences in x .

Three Types of Matching Shift (I)

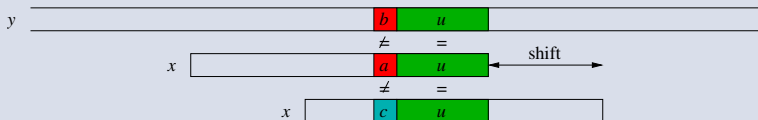
Weak Matching Shift



No condition on the character c preceding u , it is then possible that $c = a$.

Three Types of Matching Shift (II)

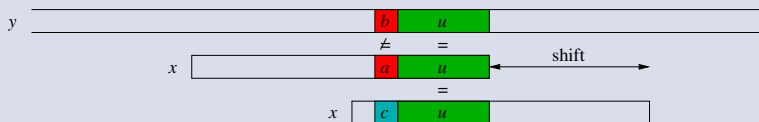
Strong Matching Shift



c must be different from the character a .

Three Types of Matching Shift (III)

Best Matching Shift



c must be equal to b .

Three Types of Matching Shift

- weak and strong matching shift only depend on x
- best matching shift depends on x and the alphabet

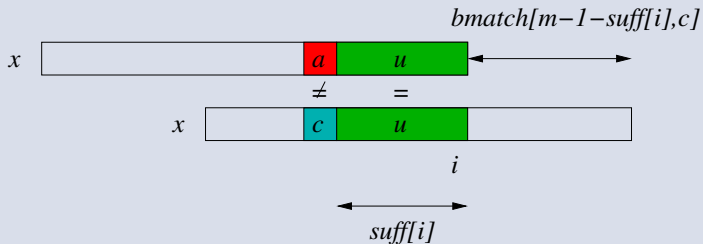
Outline

- 1 Introduction
- 2 Best Matching Shift**
- 3 Hashing q -grams
- 4 Experimental results

Computation of the best matching shift

For $0 \leq i \leq m - 1$:

$suff[i]$ = length of the longest suffix of x ending at position i in x .



Computation of the best matching shift

Scan the position of the table *suff* from left to right

Another proof of linearity

Computation of the best matching shift

Example

	a	c	g	t
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				

i	0	1	2	3	4	5	6	7	8	9	10	11
$x[i]$	c	a	t	a	c	a	t	a	a	a	t	a
$suff[i]$	0	1	0	3	0	1	0	3	1	1	0	12

Computation of the best matching shift

Example

	a	c	g	t
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11		11		

i	0	1	2	3	4	5	6	7	8	9	10	11
$x[i]$	c	a	t	a	c	a	t	a	a	a	t	a
$suff[i]$	0	1	0	3	0	1	0	3	1	1	0	12

Computation of the best matching shift

Example

	a	c	g	t
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10		10		
11		11		

i	0	1	2	3	4	5	6	7	8	9	10	11
$x[i]$	c	a	t	a	c	a	t	a	a	a	t	a
$suff[i]$	0	1	0	3	0	1	0	3	1	1	0	12

Computation of the best matching shift

Example

	a	c	g	t
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10		10		
11		11		9

i	0	1	2	3	4	5	6	7	8	9	10	11
$x[i]$	c	a	t	a	c	a	t	a	a	a	t	a
$suff[i]$	0	1	0	3	0	1	0	3	1	1	0	12

Computation of the best matching shift

Example

	a	c	g	t
0				
1				
2				
3				
4				
5				
6				
7				
8		8		
9				
10		10		
11		11		9

i	0	1	2	3	4	5	6	7	8	9	10	11
$x[i]$	c	a	t	a	c	a	t	a	a	a	t	a
$suff[i]$	0	1	0	3	0	1	0	3	1	1	0	12

Computation of the best matching shift

Example

	a	c	g	t
0				
1				
2				
3				
4				
5				
6				
7				
8		8		
9				
10		10		
11		7		9

i	0	1	2	3	4	5	6	7	8	9	10	11
$x[i]$	c	a	t	a	c	a	t	a	a	a	t	a
$suff[i]$	0	1	0	3	0	1	0	3	1	1	0	12

Computation of the best matching shift

Example

	a	c	g	t
0				
1				
2				
3				
4				
5				
6				
7				
8		8		
9				
10		6		
11		7		9

i	0	1	2	3	4	5	6	7	8	9	10	11
$x[i]$	c	a	t	a	c	a	t	a	a	a	t	a
$suff[i]$	0	1	0	3	0	1	0	3	1	1	0	12

Computation of the best matching shift

Example

	a	c	g	t
0				
1				
2				
3				
4				
5				
6				
7				
8		8		
9				
10		6		
11		7		5

i	0	1	2	3	4	5	6	7	8	9	10	11
$x[i]$	c	a	t	a	c	a	t	a	a	a	t	a
$suff[i]$	0	1	0	3	0	1	0	3	1	1	0	12

Computation of the best matching shift

Example

	a	c	g	t
0				
1				
2				
3				
4				
5				
6				
7				
8		4		
9				
10		6		
11		7		5

i	0	1	2	3	4	5	6	7	8	9	10	11
$x[i]$	c	a	t	a	c	a	t	a	a	a	t	a
$suff[i]$	0	1	0	3	0	1	0	3	1	1	0	12

Computation of the best matching shift

Example

	a	c	g	t
0				
1				
2				
3				
4				
5				
6				
7				
8		4		
9				
10	3	6		
11		7		5

i	0	1	2	3	4	5	6	7	8	9	10	11
$x[i]$	c	a	t	a	c	a	t	a	a	a	t	a
$suff[i]$	0	1	0	3	0	1	0	3	1	1	0	12

Computation of the best matching shift

Example

	a	c	g	t
0				
1				
2				
3				
4				
5				
6				
7				
8		4		
9				
10	2	6		
11		7		5

i	0	1	2	3	4	5	6	7	8	9	10	11
$x[i]$	c	a	t	a	c	a	t	a	a	a	t	a
$suff[i]$	0	1	0	3	0	1	0	3	1	1	0	12

Computation of the best matching shift

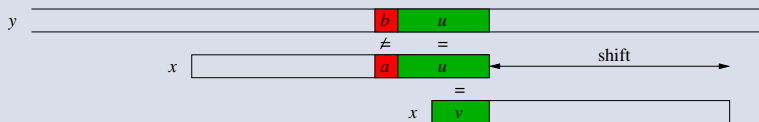
Example

	a	c	g	t
0				
1				
2				
3				
4				
5				
6				
7				
8		4		
9				
10	2	6		
11		7		1

i	0	1	2	3	4	5	6	7	8	9	10	11
$x[i]$	c	a	t	a	c	a	t	a	a	a	t	a
$suff[i]$	0	1	0	3	0	1	0	3	1	1	0	12

Best Matching Shift

Degenerate case



u does not reoccur in x

v is the longest prefix of x matching u which is a suffix of x

v is a border that can be detected when $\text{suff}[i] = i + 1$

Computation of the best matching shift

Example

	a	c	g	t
0	12	12	12	12
1	12	12	12	12
2	12	12	12	12
3	12	12	12	12
4	12	12	12	12
5	12	12	12	12
6	12	12	12	12
7	12	12	12	12
8	12	12	12	12
9	12	12	12	12
10	12	12	12	12
11	12	12	12	12

i	0	1	2	3	4	5	6	7	8	9	10	11
$x[i]$	c	a	t	a	c	a	t	a	a	a	t	a
$suff[i]$	0	1	0	3	0	1	0	3	1	1	0	12

Computation of the best matching shift

Example

	a	c	g	t
0	12	12	12	12
1	12	12	12	12
2	12	12	12	12
3	12	12	12	12
4	12	12	12	12
5	12	12	12	12
6	12	12	12	12
7	12	12	12	12
8	4	12	12	12
9	12	12	12	12
10	2	6	12	12
11	12	7	12	1

i	0	1	2	3	4	5	6	7	8	9	10	11
$x[i]$	c	a	t	a	c	a	t	a	a	a	t	a
$suff[i]$	0	1	0	3	0	1	0	3	1	1	0	12

Outline

- 1 Introduction
- 2 Best Matching Shift
- 3 Hashing q -grams**
- 4 Experimental results

Hashing q -grams

- Compute a hash value in $[0; 255]$ for every q -grams of the pattern x .
- Compute a shift for every hash value.
- Unroll the loops as much as possible.

Hashing q -grams

Example

i	0	1	2	3	4	5	6	7	8	9	10	11
$x[i]$	c	a	t	a	c	a	t	a	a	a	t	a

$shift[i] \leftarrow 10$

$\forall i \in [0; 255]$

Hashing q -grams

Example

i	0	1	2	3	4	5	6	7	8	9	10	11
$x[i]$	c	a	t	a	c	a	t	a	a	a	t	a

$$h(\text{cat}) = ((\text{rank}(\text{c}) \times 2 + \text{rank}(\text{a})) \times 2 + \text{rank}(\text{t})) = 194$$

$$\text{shift}[194] = 10$$

$$\text{shift}[194] \leftarrow 9$$

Hashing q -grams

Example

i	0	1	2	3	4	5	6	7	8	9	10	11
$x[i]$	c	a	t	a	c	a	t	a	a	a	t	a

$$h(\text{ata}) = ((\text{rank}(\text{a}) \times 2 + \text{rank}(\text{a})) \times 2 + \text{rank}(\text{a})) = 205$$

$$\text{shift}[205] = 10$$

$$\text{shift}[205] \leftarrow 8$$

Hashing q -grams

Example

i	0	1	2	3	4	5	6	7	8	9	10	11
$x[i]$	c	a	t	a	c	a	t	a	a	a	t	a

$$h(\mathbf{tac}) = ((\mathit{rank}(\mathbf{t}) \times 2 + \mathit{rank}(\mathbf{a})) \times 2 + \mathit{rank}(\mathbf{c})) = 245$$

$$\mathit{shift}[245] = 10$$

$$\mathit{shift}[245] \leftarrow 7$$

Hashing q -grams

Example

i	0	1	2	3	4	5	6	7	8	9	10	11
$x[i]$	c	a	t	a	c	a	t	a	a	a	t	a

$$h(\text{aca}) = ((\text{rank}(\text{a}) \times 2 + \text{rank}(\text{c})) \times 2 + \text{rank}(\text{a})) = 171$$

$$\text{shift}[171] = 10$$

$$\text{shift}[171] \leftarrow 6$$

Hashing q -grams

Example

i	0	1	2	3	4	5	6	7	8	9	10	11
$x[i]$	c	a	t	a	c	a	t	a	a	a	t	a

$$h(\text{cat}) = ((\text{rank}(\text{c}) \times 2 + \text{rank}(\text{a})) \times 2 + \text{rank}(\text{t})) = 194$$

$$\text{shift}[194] = 9$$

$$\text{shift}[194] \leftarrow 5$$

Hashing q -grams

Example

i	0	1	2	3	4	5	6	7	8	9	10	11
$x[i]$	c	a	t	a	c	a	t	a	a	a	t	a

$$h(\text{ata}) = ((\text{rank}(\text{a}) \times 2 + \text{rank}(\text{t})) \times 2 + \text{rank}(\text{a})) = 205$$

$$\text{shift}[205] = 8$$

$$\text{shift}[205] \leftarrow 4$$

Hashing q -grams

Example

i	0	1	2	3	4	5	6	7	8	9	10	11
$x[i]$	c	a	t	a	c	a	t	a	a	a	t	a

$$h(\mathbf{taa}) = ((\mathit{rank}(\mathbf{t}) \times 2 + \mathit{rank}(\mathbf{a})) \times 2 + \mathit{rank}(\mathbf{a})) = 243$$

$$\mathit{shift}[243] = 10$$

$$\mathit{shift}[243] \leftarrow 3$$

Hashing q -grams

Example

i	0	1	2	3	4	5	6	7	8	9	10	11
$x[i]$	c	a	t	a	c	a	t	a	a	a	t	a

$$h(\text{aaa}) = ((\text{rank}(\text{a}) \times 2 + \text{rank}(\text{a})) \times 2 + \text{rank}(\text{a})) = 167$$

$$\text{shift}[167] = 10$$

$$\text{shift}[167] \leftarrow 2$$

Hashing q -grams

Example

i	0	1	2	3	4	5	6	7	8	9	10	11
$x[i]$	c	a	t	a	c	a	t	a	a	a	t	a

$$h(\mathbf{aat}) = ((\mathit{rank}(\mathbf{a}) \times 2 + \mathit{rank}(\mathbf{a})) \times 2 + \mathit{rank}(\mathbf{t})) = 186$$

$$\mathit{shift}[186] = 10$$

$$\mathit{shift}[186] \leftarrow 1$$

Hashing q -grams

Example

i	0	1	2	3	4	5	6	7	8	9	10	11
$x[i]$	c	a	t	a	c	a	t	a	a	a	t	a

$$h(\text{ata}) = ((\text{rank}(\text{a}) \times 2 + \text{rank}(\text{t})) \times 2 + \text{rank}(\text{a})) = 205$$

$$\text{shift}[205] = 4 \implies \text{sh1} \leftarrow 4$$

$$\text{shift}[205] \leftarrow 0$$

Hashing q -grams

Algorithm $\text{New}_q(x, m, y, n)$ for $q = 3$

▷ Preprocessing

for $i \leftarrow 0$ **to** 255 **do** $\text{shift}[i] \leftarrow m - 2$

for $i \leftarrow 2$ **to** $m - 2$ **do**

$h \leftarrow ((x[i - 2] \times 2 + x[i - 1]) \times 2) + x[i]$

$\text{shift}[h \bmod 256] \leftarrow m - 1 - i$

$h \leftarrow ((x[m - 3] \times 2 + x[m - 2]) \times 2) + x[m - 1]$

$\text{sh1} \leftarrow \text{shift}[h \bmod 256]$

$\text{shift}[h \bmod 256] \leftarrow 0$

Hashing q -grams

Algorithm $\text{New}q(x, m, y, n)$ for $q = 3$

▷ Searching

$y[n..n + m - 1] \leftarrow x$ ▷ Sentinel

$j \leftarrow m - 1$

while TRUE **do**

$sh \leftarrow 1$

while $sh \neq 0$ **do**

$h \leftarrow ((y[j - 2] \times 2 + y[j - 1]) \times 2) + y[j]$

$sh \leftarrow \text{shift}[h \bmod 256]$

$j \leftarrow j + sh$

if $j < n$ **then**

if $x = y[j - m + 1..j]$ **then** REPORT($j - m + 1$)

$j \leftarrow j + sh1$

else RETURN

Outline

- 1 Introduction
- 2 Best Matching Shift
- 3 Hashing q -grams
- 4 Experimental results**

Experimental results

Conditions

- Intel Pentium processor at 1300MHz
- Linux Red Hat version 2.4.20-8
- gcc with the full optimization option `-O3`
- clock function
- 100 patterns randomly chosen in the texts

Texts

- Binary alphabet, random (uniform distribution), 4Mb
- E.coli from Large Canterbury Corpus, 4.6Mb
- Alphabet of size 8, random (uniform distribution), 4Mb
- world192.txt from Large Canterbury Corpus, 4.3Mb

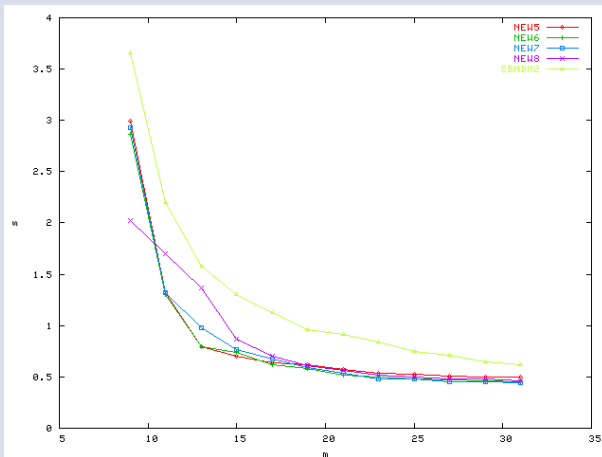
Experimental results

Algorithms

- BM2fast Boyer–Moore with best matching shift and fast loop
- NEW q for $q \in [3; 8]$
- TBM Tuned Boyer–Moore (Hume & Sunday, 1991)
- SSABS (Sheik, Aggarwal, Poddar, Balakrishnan & Sekar, 2004)
- SBNDM2 (Holub & Durian, 2005)

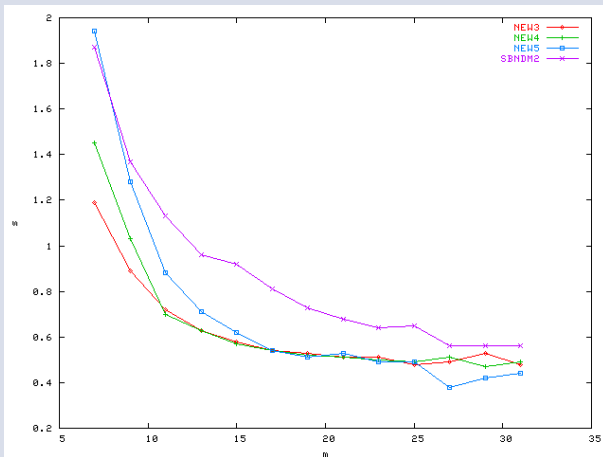
Experimental results

Binary alphabet



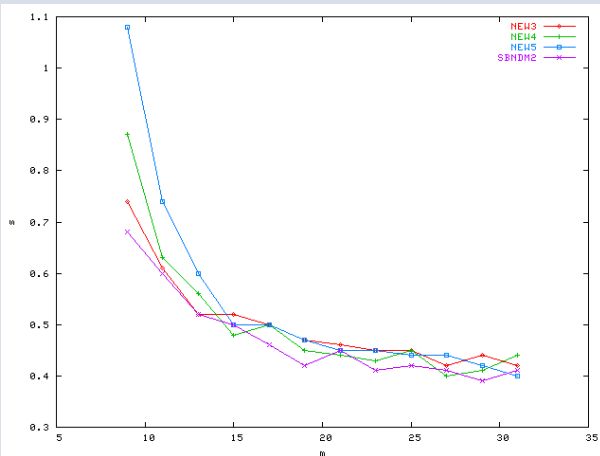
Experimental results

E.coli from the Large Canterbury corpus



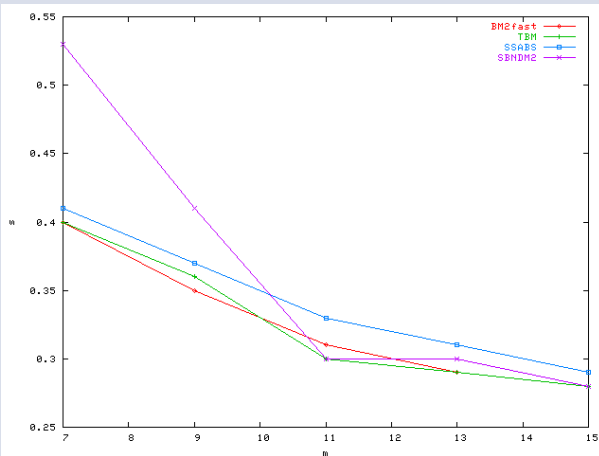
Experimental results

Alphabet of size 8



Experimental results

Natural language



Conclusions

- Binary alphabet : NEW5-8 for $m \in [9; 256]$
- Alphabet of size 4 : NEW3-5 for $m \in [7; 128]$
- Alphabet of size 8 : NEW3-5 for $m \in [13; 64]$
- Natural language : BM2fast for $m \in [7; 15]$

References



ALGORITHMS ON STRINGS

MAXIME CROCHEMORE

CHRISTOPHE HANCART

THIERRY LECROQ

CAMBRIDGE

Coming soon...
Best matching shift and
many other interesting
things

References



M. Crochemore and T. Lecroq

A fast implementation of the Boyer–Moore string matching algorithm

Submitted



T. Lecroq

Fast string matching algorithms

Information Processing Letters

Accepted