

Parameterized Suffix Arrays for Binary Strings

Satoshi Deguchi, Fumihito Higashijima,
Hideo Bannai, Shunsuke Inenaga and
Masayuki Takeda

Kyushu University, Japan

Contents

- Background
 - Parameterized matching [Baker 93]
- Linear time algorithms for constructing
 - parameterized suffix arrays
 - parameterized LCP arraysfor binary strings
- Computational Experiments
- Summary and Open Problems

Background

sort program by student A

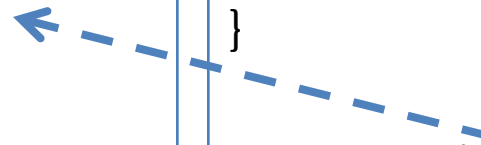
```
void sort(int *a, int n) {
    int p, q, t;

    for (p = 0; p < n; p++) {
        for (q = n-1; q > p; q--) {
            if (a[q] < a[q-1]) {
                t = a[q];
                a[q] = a[q-1];
                a[q-1] = t;
            }
        }
    }
}
```

sort program by student B

```
void sort(int *p, int n) {
    int x, y, t;

    for (x = 0; x < n; x++) {
        for (y = n-1; y > x; y--) {
            if (p[y] < p[y-1]) {
                t = p[y];
                p[y] = p[y-1];
                p[y-1] = t;
            }
        }
    }
}
```



Parameterized pattern matching

[Baker 93]

- Parameterized string

- A string over the *parameter alphabet* Π and *constant alphabet* Σ ($\Pi \cap \Sigma = \phi$)

we will only consider the case $\Sigma = \phi$ in this talk.

- Parameterized match

- Two parameterized strings x, y parameterized match if there exists a bijection f on $\Pi \cup \Sigma$ where $f(c) = c$ for $c \in \Sigma$, and $f(x) = f(x_1 \dots x_n) = f(x_1) \dots f(x_n) = y$

```
if (a[q] < a[q-1]) {  
  t      = a[q];  
  a[q]   = a[q-1];  
  a[q-1] = t;  
}
```

$q \rightarrow y$
 $a \rightarrow p$

```
if (p[y] < p[y-1]) {  
  t      = p[y];  
  p[y]   = p[y-1];  
  p[y-1] = t;  
}
```

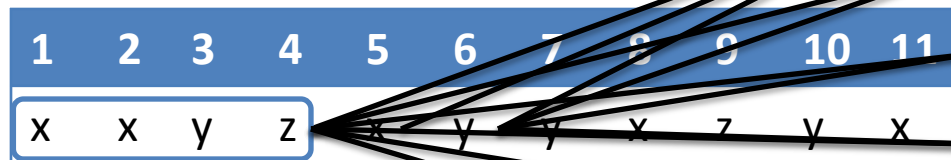
Parameterized pattern matching

[Baker 93]

- Parameterized matching problem
 - Given parameterized strings p and t , find all positions i where p and $t_i \dots t_{i+|p|-1}$ parameterized match

t : xxzyxzyxzyx

p : xyzx



xyzx

xzyx ($y \rightarrow z, z \rightarrow y$)

yxzy ($x \rightarrow y, y \rightarrow x$)

yzxy ($x \rightarrow y, y \rightarrow z, z \rightarrow x$)

zxyz ($x \rightarrow z, y \rightarrow x, z \rightarrow y$)

zyxz ($x \rightarrow z, z \rightarrow x$)

Parameterized pattern matching

[Baker 93]

- $$pv(s)[i] = \begin{cases} 0 & \text{if } s[i] \neq s[j] \text{ for any } 1 \leq j < i \\ i - k & \text{if } k = \max \{j \mid s[i] = s[j] \ 1 \leq j < i\} \end{cases}$$

$pv(\text{abaabaaaabba})$
 $= 002132111513$

Proposition [Baker '93]

Two parameterized strings s, t parameterized match $\Leftrightarrow pv(s) = pv(t)$

$t: \underline{xyzyx}yyxzyx$

$p: xyzx$



$pv(yzxy) = 0003$

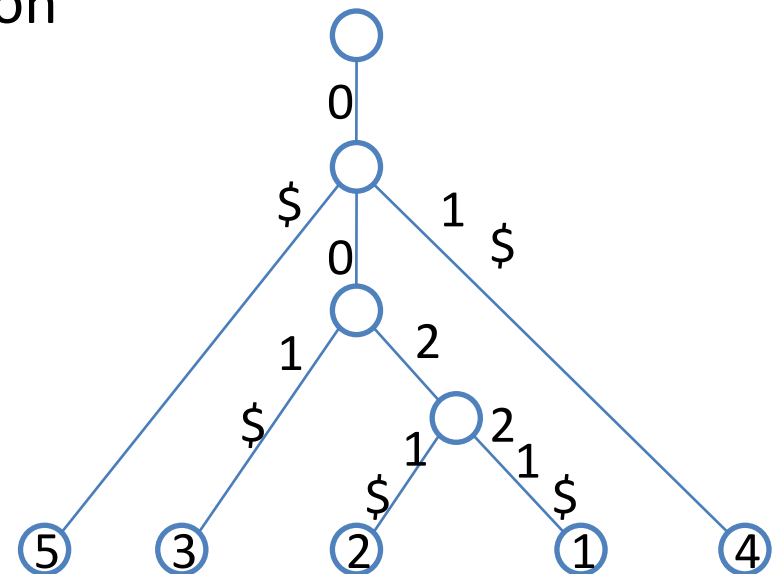


$pv(xyzx) = 0003$

Parameterized suffix tree (p-suffix tree) [Baker 93]

parameterized string t $\xrightarrow{O(|t|) \text{ time}^*}$ p-suffix tree
construction

- | | | | | |
|----|---------|---------------|---------|---------------------|
| 1. | ababb\$ | \rightarrow | 00221\$ | pv() of each suffix |
| 2. | babb\$ | \rightarrow | 0021\$ | |
| 3. | abb\$ | \rightarrow | 001\$ | |
| 4. | bb\$ | \rightarrow | 01\$ | |
| 5. | b\$ | \rightarrow | 0\$ | |



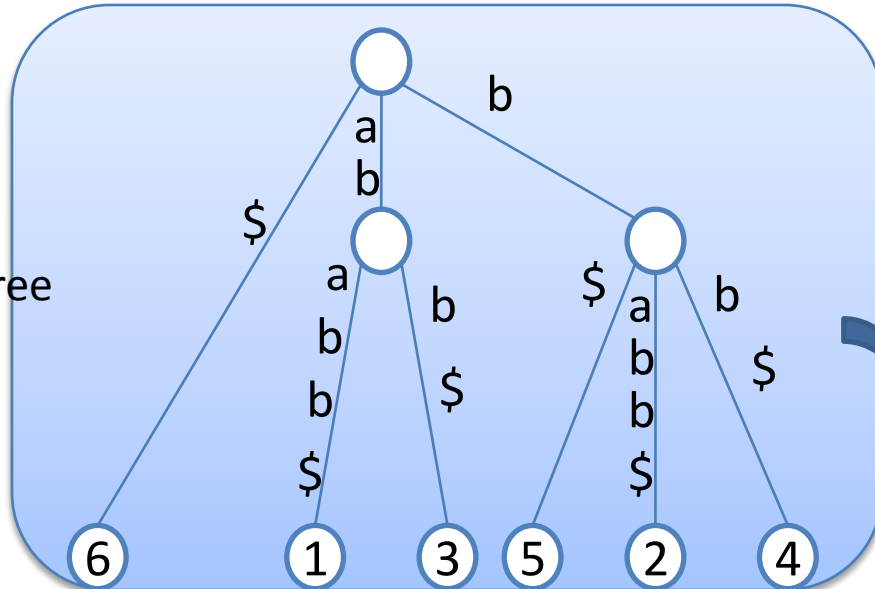
$O(|p| + \#occ)$ time* matching

* Assuming constant size alphabet

Standard suffix trees and arrays

Linear time
[Weiner 73]

suffix tree



Linear time
(simple traversal)

ababb\$

Linear time
[Ko&Aluru 03,
Karkkainen&Sanders 03,
Kim et al. 03]

suffix array
SA[]
LCP array
LCP[]

suffix array	6	1	3	5	2	4
SA[]	-1	0	2	0	1	1
LCP[]	\$	a	a	b	b	b
		b	b	\$	a	b
		a	b		b	\$
		b	\$		b	
		b			\$	
		\$				

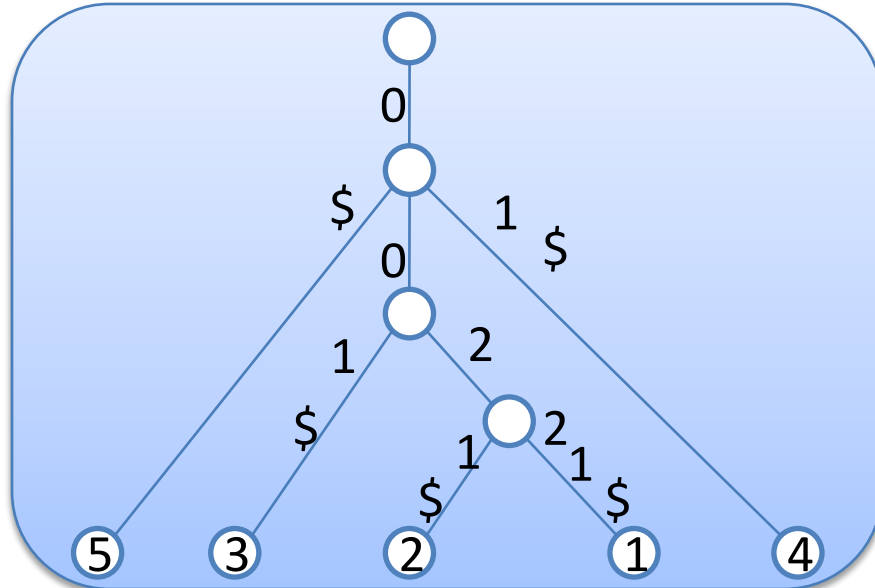
Linear time
[Kasai et al. 01]

Suffix arrays are reported to be superior in terms of speed/memory usage, for many applications

P-suffix trees and arrays

Linear time
[Baker '93]

p-suffix tree



ababb\$

P-suffix array

PSA[]

P-LCP array

PLCP[]

5	3	2	1	4
-1	1	2	3	1
0	0	0	0	0
\$	0	0	0	1
	1	2	2	\$
	\$	1	2	
		\$	1	
			\$	

Linear time
(simple traversal)

Linear time for
binary strings
[this work]

Linear time for
binary strings
[this work]

This work

We

- consider the p-suffix array and p-LCP array
- show linear time algorithms for **direct construction** of **p-suffix array** and its **p-LCP array** for binary strings

Difficulties of pV

A pV of a suffix is not necessarily a suffix of a pV



$$pV(\text{abaabaaaabba}) = 0021\underline{3}2111513$$

suffix
↓

NOT
suffix
↓



$$pV(\text{baabaaaabba}) = 001\underline{3}2111513$$

Difficulties of pv

Lexicographic order between suffixes of two strings is not necessarily preserved, even when they share a common prefix

$$pv(\text{baabaaaabba}) = \begin{array}{c} 001\underline{3}2111513 \end{array} < \begin{array}{c} pv(\text{abaabaaaabba}) = \\ 00\underline{2}132111513 \end{array}$$

suffix
↓

suffix
↓

$$pv(\text{aabaaaabba}) = \begin{array}{c} 0\underline{1}02111513 \end{array} > \begin{array}{c} pv(\text{baabaaaabba}) = \\ 001\underline{3}2111513 \end{array}$$




Linear time algorithms for direct construction of standard suffix arrays won't work on the pv array.

Direct construction of p-suffix array for binary strings

Observation

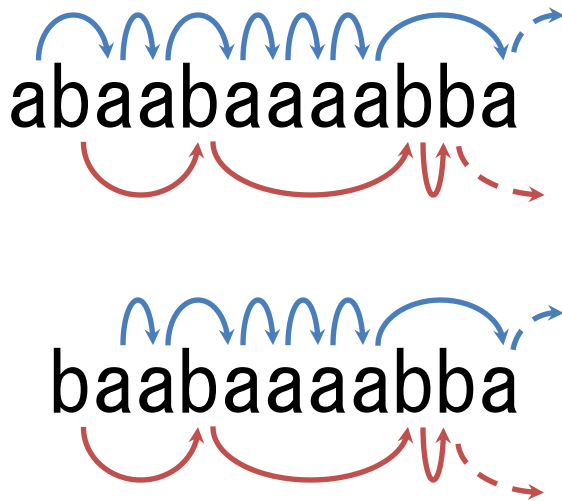
i	$PSA[i]$	$s[PSA[i] : n]$	$pv(s[PSA[i] : n])$
1	12	a	0
2	11	ba	0 0
3	5	baaaabba	0 0 1 1 1 <u>5</u> 1 3
4	9	abba	0 0 1 <u>3</u>
5	2	baabaaaabba	0 0 1 <u>3</u> 2 1 1 1 5 1 3
6	4	abaaaabba	0 0 <u>2</u> 1 1 1 5 1 3
7	1	abaabaaaabba	0 0 <u>2</u> 1 3 2 1 1 1 5 1 3
8	10	bba	0 <u>1</u> 0
9	8	aabba	0 <u>1</u> 0 1 3
10	3	aabaaaabba	0 <u>1</u> 0 2 1 1 1 5 1 3
11	7	aaabba	0 <u>1</u> 1 0 1 3
12	6	aaaabba	0 <u>1</u> 1 1 0 1 3

Monotonically decrease?



fw array

$$fw(s)[i] = \begin{cases} \infty & \text{if } s[i] \neq s[j] \text{ for any } i < j \leq |s| \\ i - k & \text{if } k = \min \{ j \mid s[i] = s[j] \ i < j \leq |s| \} \end{cases}$$



$$fw(\text{abaabaaaabba}) = 2 \ 3 \ 1 \ 2 \ 5 \ 1 \ 1 \ 1 \ 3 \ 1 \ \infty \ \infty$$

suffix

suffix

$$fw(\text{baabaaaabba}) = 3 \ 1 \ 2 \ 5 \ 1 \ 1 \ 1 \ 3 \ 1 \ \infty \ \infty$$

A fw of a suffix is always a suffix of a fw!

Direct construction of p-suffix array for binary strings

Lemma

For any binary string s , the p-suffix array of s is equivalent to the standard suffix array of $fw(s)$

i	$PSA[i]$	$s[PSA[i] : n]$	$pv(s[PSA[i] : n])$	$fw(s[PSA[i] : n])$
1	12	a	0	∞
2	11	ba	0 0	$\infty \infty$
3	5	baaaabba	0 0 1 1 1 <u>5</u> 1 3	<u>5</u> 1 1 1 3 1 $\infty \infty$
4	9	abba	0 0 1 <u>3</u>	<u>3</u> 1 $\infty \infty$
5	8	abba	0 0 1 3 <u>2</u> 1 1 1 5 1 3	<u>3</u> 1 2 5 1 1 1 3 1 $\infty \infty$

Theorem

The p-suffix array for binary string of length n can be constructed directly in $O(n)$ time.

6	10	abba	0 <u>1</u> 0 1 3	<u>1</u> 3 1 $\infty \infty$
7	10	abba	0 <u>1</u> 0 1 3	<u>1</u> 3 1 $\infty \infty$
8	10	abba	0 <u>1</u> 0 1 3	<u>1</u> 3 1 $\infty \infty$
9	10	abba	0 <u>1</u> 0 1 3	<u>1</u> 3 1 $\infty \infty$
10	3	aabaaaabba	0 <u>1</u> 0 2 1 1 1 5 1 3	<u>1</u> 2 5 1 1 1 3 1 $\infty \infty$
11	7	aaabba	0 <u>1</u> 1 0 1 3	<u>1</u> 1 3 1 $\infty \infty$
12	6	aaaabba	0 <u>1</u> 1 1 0 1 3	<u>1</u> 1 1 3 1 $\infty \infty$

(With the reverse order on integers)

Linear time LCP array construction

Standard suffix arrays: [Kasai et al. 2001]

i	SA[i]	$s[SA[i]:n]$	LCP _s [i]
1	9	\$	
2	6	aab\$	
3	7	<u>ab</u> \$	
4	1	<u>ab</u> abbaab\$	
5	3	<u>ab</u> baab\$	$lcp[5] \geq 2-1$
6	8	b\$	
7	5	<u>ba</u> ab\$	
8	2	<u>ba</u> baab\$	$lcp[8] \geq 2-1$
9	4	bbaab\$	

Depends on preservation of lexicographic order of suffixes with common prefix → won't work for parameterized strings

i	PSA[i]	$s[PSA[i]:n]$	$pv(s[PSA[i]:n])$	PLCP _{s} [i]
1	12	a	0	-1
2	11	ba	00	1
3	5	baaaaabba	00111513	2
4	9	abba	0013	3
5	2	baabaaaabba	00132111513	4
6	4	abaaaabba	002111513	2
7	1	abaabaaaabba	002132111513	4
8	10	bba	010	1
9	8	abba	01013	3
10	3	aaabaaaabba	0102111513	3
11	7	aaabba	011013	2
12	6	aaaabba	0111013	3

Lexicographic order is not preserved even when PLCP > 0.

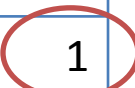
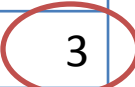
$$3 < 4$$



$$12 > 8$$

Standard LCP algorithm won't work.

$$PLCP : 3 - 1 \not\leq 1$$



Linear time parameterized LCP array construction for Binary Strings

The LCP array of fw can be used to compute the PLCP array of pv

Lemma

For any binary string s ,

$$PLCP_s[i] = \min\{ LCP_{fw(s)}[i] + fw(s)[PSA[i]+1], |s| - PSA[i-1] + 1 \}$$

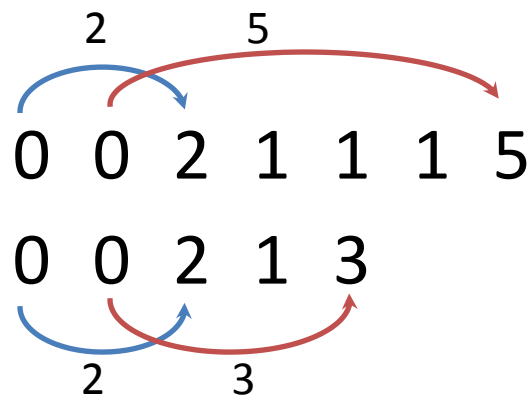
LCP of fw
+ first disagreeing fw element

Length of $PSA[i-1]$ th suffix

i	$PSA[i]$	$s[PSA[i] : n]$	$pv(s[PSA[i] : n])$	$PLCP_s[i]$	$fw(s[PSA[i] : n])$	$LCP_{fw(s)}[i]$
1	12	a	0	-1	∞	-1
2	11	ba	0 0	1	$\infty \infty$	1
3	5	baaaabba	0 0 1 1 1 <u>5</u> 1 3	2	<u>5</u> 1 1 1 3 1 $\infty \infty$	0
4	9	abba	0 0 1 <u>3</u>	3	<u>3</u> 1 $\infty \infty$	0
5	2	baabaaaabba	0 0 1 <u>3</u> 2 1 1 1 5 1 3	4	<u>3</u> 1 2 5 1 1 1 3 1 $\infty \infty$	2
6	4	abaaaabba	0 0 <u>2</u> 1 1 1 5 1 3	2	<u>2</u> 5 1 1 1 3 1 $\infty \infty$	0
7	1	abaabaaaabba	0 0 <u>2</u> 1 3 2 1 1 1 5 1 3	4	<u>2</u> 3 1 2 5 1 1 1 3 1 $\infty \infty$	1
8	10	bba	0 <u>1</u> 0	1	<u>1</u> $\infty \infty$	0
9	8	aabba	0 <u>1</u> 0 1 3	3	<u>1</u> 3 1 $\infty \infty$	1
10	3	aabaaaabba	0 <u>1</u> 0 2 1 1 1 5 1 3	3	<u>1</u> 2 5 1 1 1 3 1 $\infty \infty$	1
11	7	aaabba	0 <u>1</u> 1 0 1 3	2	<u>1</u> 1 3 1 $\infty \infty$	1
12	6	aaaabba	0 <u>1</u> 1 1 0 1 3	3	<u>1</u> 1 1 3 1 $\infty \infty$	2

LCP of fw + first disagreeing fw element

$$PLCP_s[i] = \min\{\underline{LCP}_{fw(s)}[i] + fw(s)[PSA[i]+1], |s| - PSA[i-1] + 1\}$$



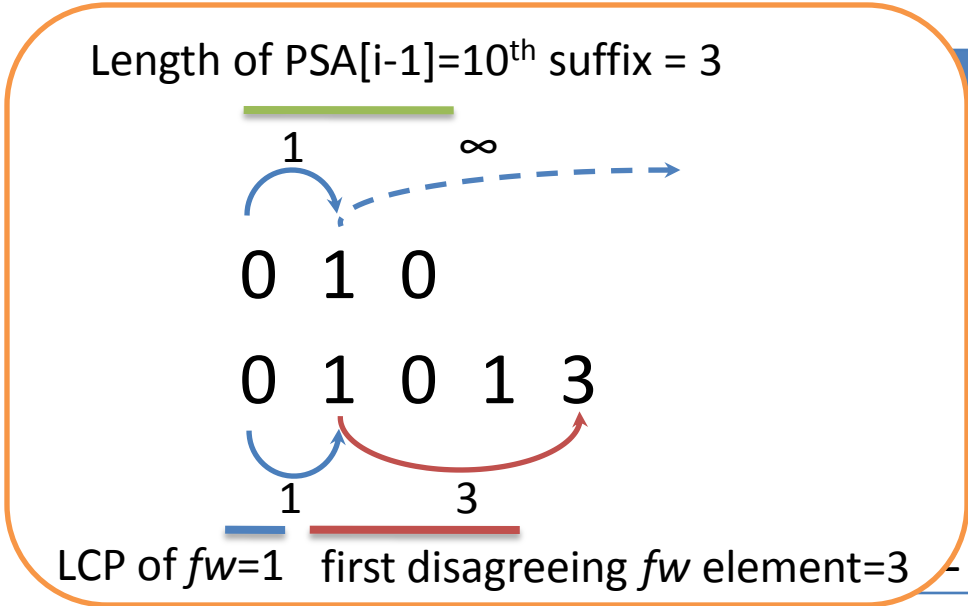
LCP of $fw=1$ first disagreeing fw element=3

	$fw(s[PSA[i]:n])$	$LCP_s[i]$
	∞	-1
	$\infty\infty$	1
	5 1 1 1 3 1 $\infty\infty$	0
	3 1 $\infty\infty$	0
	3 1 2 5 1 1 1 3 1 $\infty\infty$	2
6	2 <u>2 5</u> 1 1 1 3 1 $\infty\infty$	0
7	4 <u>2 3</u> 1 2 5 1 1 1 3 1 $\infty\infty$	1
8	1 1 $\infty\infty$	0
9	3 1 3 1 $\infty\infty$	1
10	3 1 2 5 1 1 1 3 1 $\infty\infty$	1
11	2 1 1 3 1 $\infty\infty$	1
12	3 1 1 1 3 1 $\infty\infty$	2

1+3

Length of PSA[i-1]th suffix

$$PLCP_s[i] = \min\{LCP_{fw(s)}[i] + fw(s)[PSA[i]+/], \underline{|s| - PSA[i-1] + 1}\}$$




	$fw(s[PSA[i]:n])$	$LCP_s[i]$
	∞	-1
	$\infty\infty$	1
	5 1 1 1 3 1 $\infty\infty$	0
	3 1 $\infty\infty$	0
	3 1 2 5 1 1 1 3 1 $\infty\infty$	2
	2 5 1 1 1 3 1 $\infty\infty$	0
	4 2 3 1 2 5 1 1 1 3 1 $\infty\infty$	1
8	10 bba 12 -10 + 1 = 3 1 1 $\infty\infty$	0
9	8 aabba 3 1 3 1 $\infty\infty$	1
10	3 aabaaaabba 3 1 2 5 1 1 1 3 1 $\infty\infty$	1
11	7 aaabba 2 1 1 3 1 $\infty\infty$	1
12	6 aaaabba 3 1 1 1 3 1 $\infty\infty$	2

i	PSA[i]	$s[\text{PSA}[i]:n]$	PLCP _s [i]	$fw(s[\text{PSA}[i]:n])$	LCP _s [i]
1	12	a		∞	-1
2	11	ba		$\infty \infty$	1
3	5	baaaa	$\min\{0+5, 2\}$	<u>5</u> 1 1 1 3 1 $\infty \infty$	0
4	9	abba		<u>3</u> 1 $\infty \infty$	0
5	2	baaba	$\min\{2+2, 4\}$	<u>3</u> 1 <u>2</u> 5 1 1 1 3 1 $\infty \infty$	2
6	4	abaaa		<u>2</u> 5 1 1 1 3 1 $\infty \infty$	0
7	1	abaab	$\min\{1+3, 9\}$	<u>2</u> <u>3</u> 1 2 5 1 1 1 3 1 $\infty \infty$	1
8	10	bba		1 $\infty \infty$	0
9	8	aabba		1 3 1 $\infty \infty$	1
				1 2 5 1 1 1 3 1 $\infty \infty$	1

Theorem

The PLCP array for binary string of length n can be constructed in $O(n)$ time, given its p-suffix array.

Computational Experiments

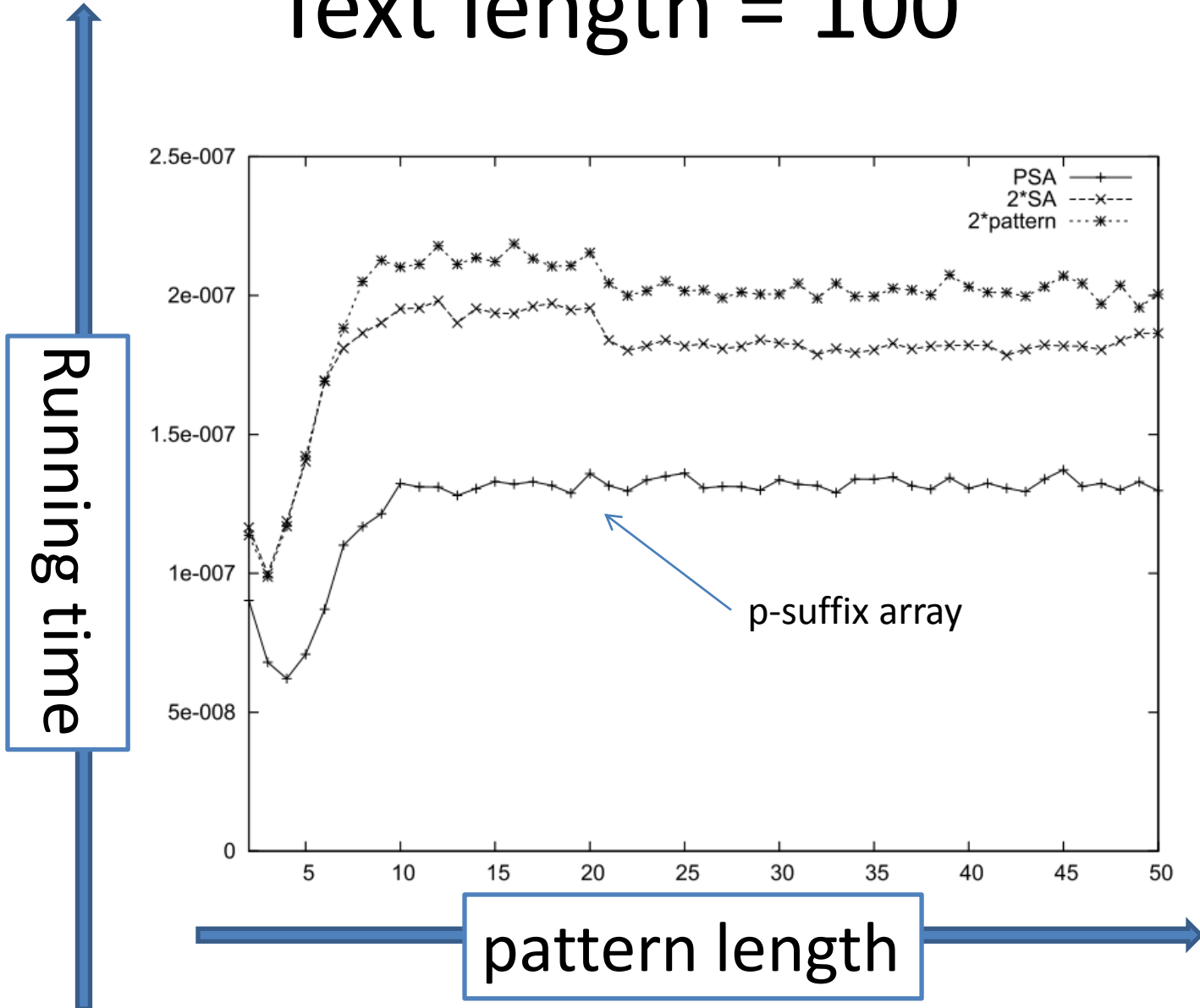
- We compared speed of parameterized pattern matching on random binary strings by:
 - p-suffix array of t matched with $pv(p)$
 - standard suffix array t , matched with p and p' 
 - 2 standard suffix arrays t and t' , each matched with p

bitwise
complement

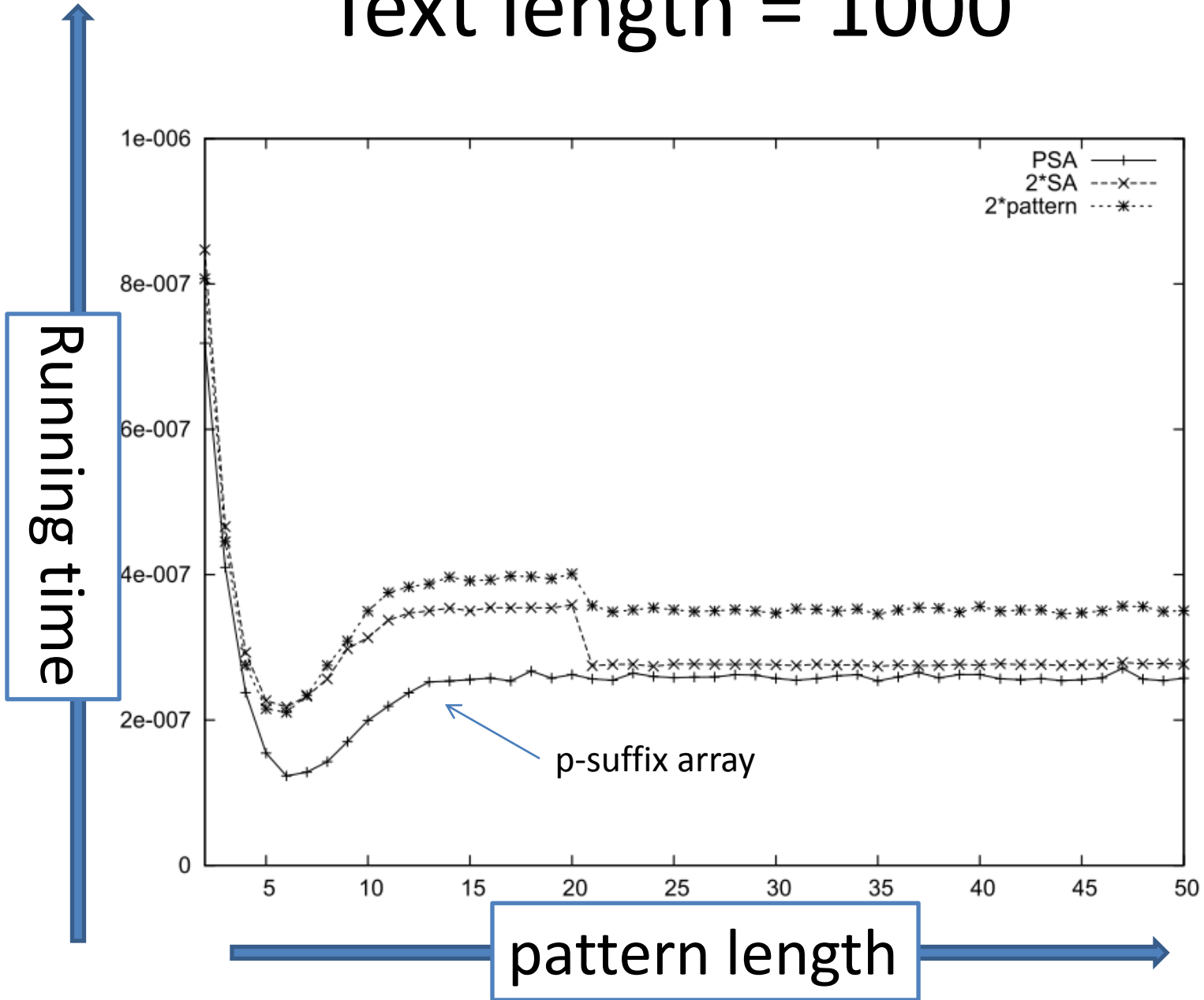
In general, for alphabet size k , we can

- construct $k!$ standard suffix arrays for each permutation of parameterized alphabet
- construct $k!$ different patterns for each permutation of parameterized alphabet
- ➔ and do standard matching $k!$ times

Text length = 100



Text length = 1000



Summary

- Showed linear time algorithm to
 - construct p-suffix arrays and p-LCP arrays *directly* from binary strings
(does not construct p-suffix tree)
- Showed efficiency of p-suffix arrays through computational experiments
 - Up to 200% faster than using suffix arrays twice

Open Problems

- Direct construction of p-suffix array for alphabet size > 2 .
- Reverse problem: Infer the parameterized string whose p-suffix array is a given permutation.