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 arrays

  for 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（Π∩Σ = ϕ）
- Parameterized match
  - Two parameterized strings *x*,*y* *parameterized match* if there exists a bijection *f* on Π∪Σ where
    $f(c) = c$ for $c \in \Sigma$, and $f(x) = f(x_1 \ldots x_n) = f(x_1) \ldots f(x_n) = y$

we will only consider the case Σ = ϕ in this talk.

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

q → y
a → p

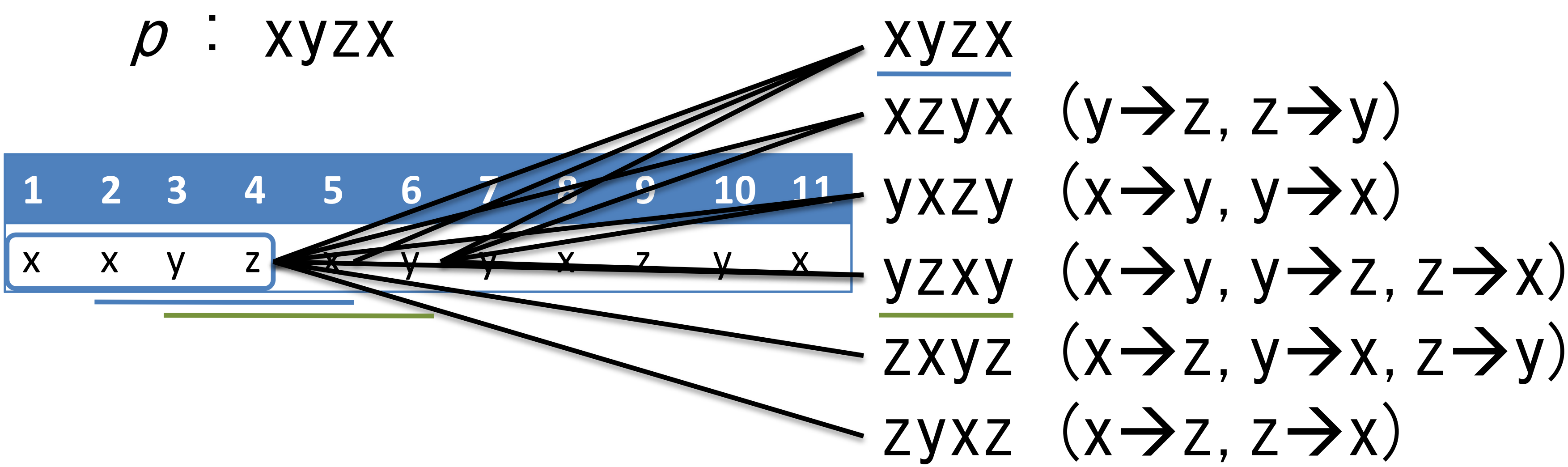
```
if (p[y] < p[y-1]) {
  t      = p[y];
  p[y]   = p[y-1];
  p[y-1] = t;
}
```

Parameterized pattern matching [Baker 93]

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

t : xxyzxyyxzyx

p : xyzx



Parameterized pattern matching [Baker 93]

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

$pv($ abaabaaaabba $)$

= 002132111513

Proposition [Baker `93]

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

t: xxyzxyyxzyx p: xyzx

pv(yzxy) = 0003 pv(xyzx) = 0003

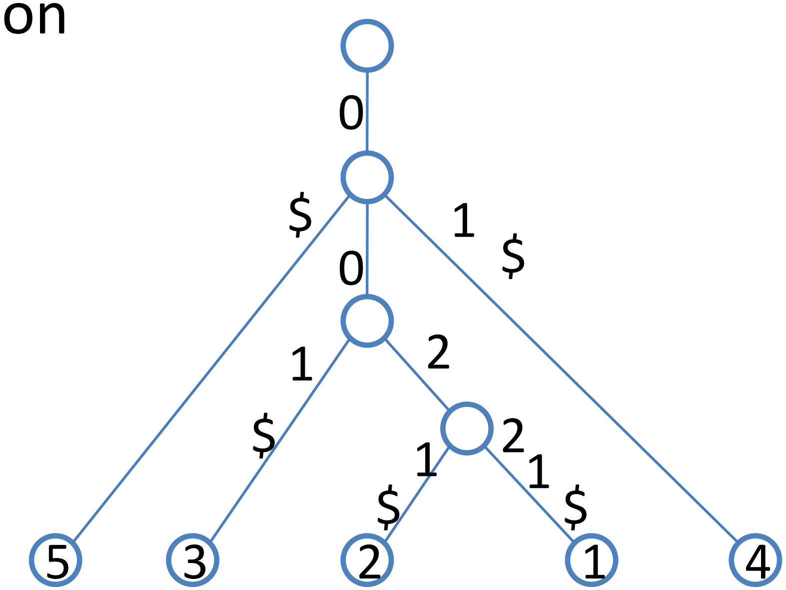
Parameterized suffix tree (p-suffix tree) [Baker 93]

parameterized string t → p-suffix tree

$O(|t|)$ time * construction

1. ababb\$ → 00221\$
2. babb\$ → 0021\$
3. abb\$ → 001\$
4. bb\$ → 01\$
5. b\$ → 0\$

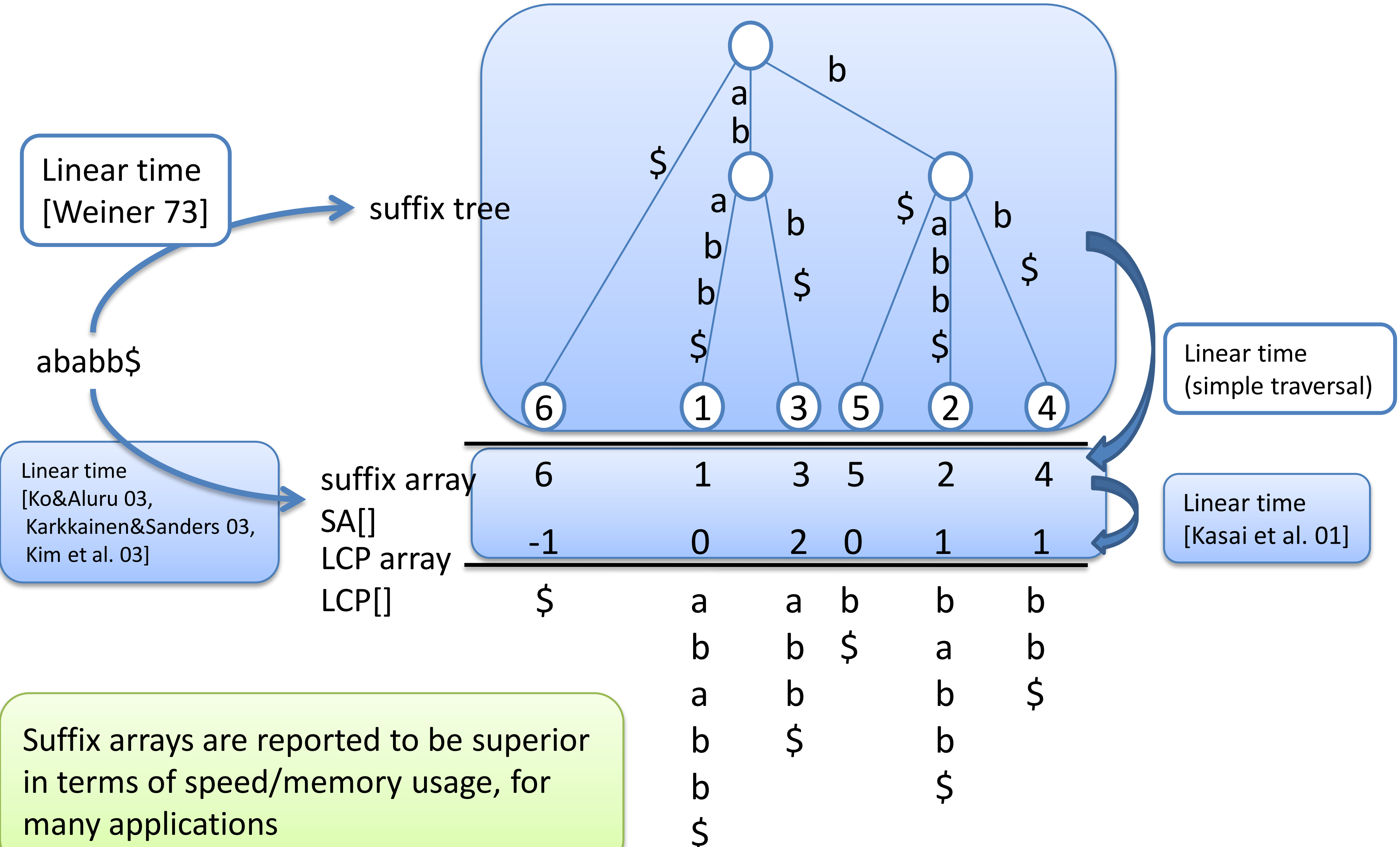
pv() of each suffix



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

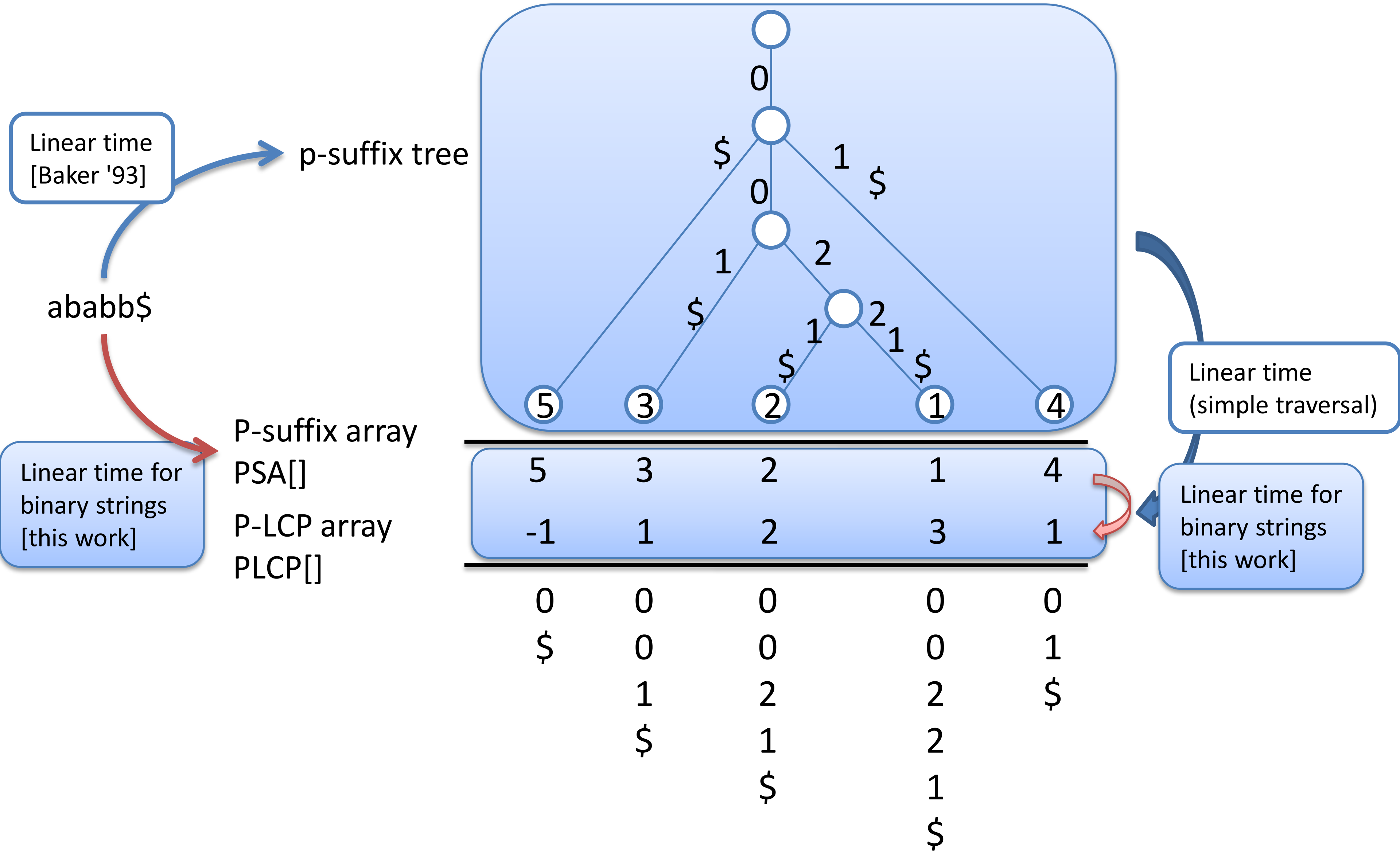
* Assuming constant size alphabet

Standard suffix trees and arrays



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

P-suffix trees and arrays



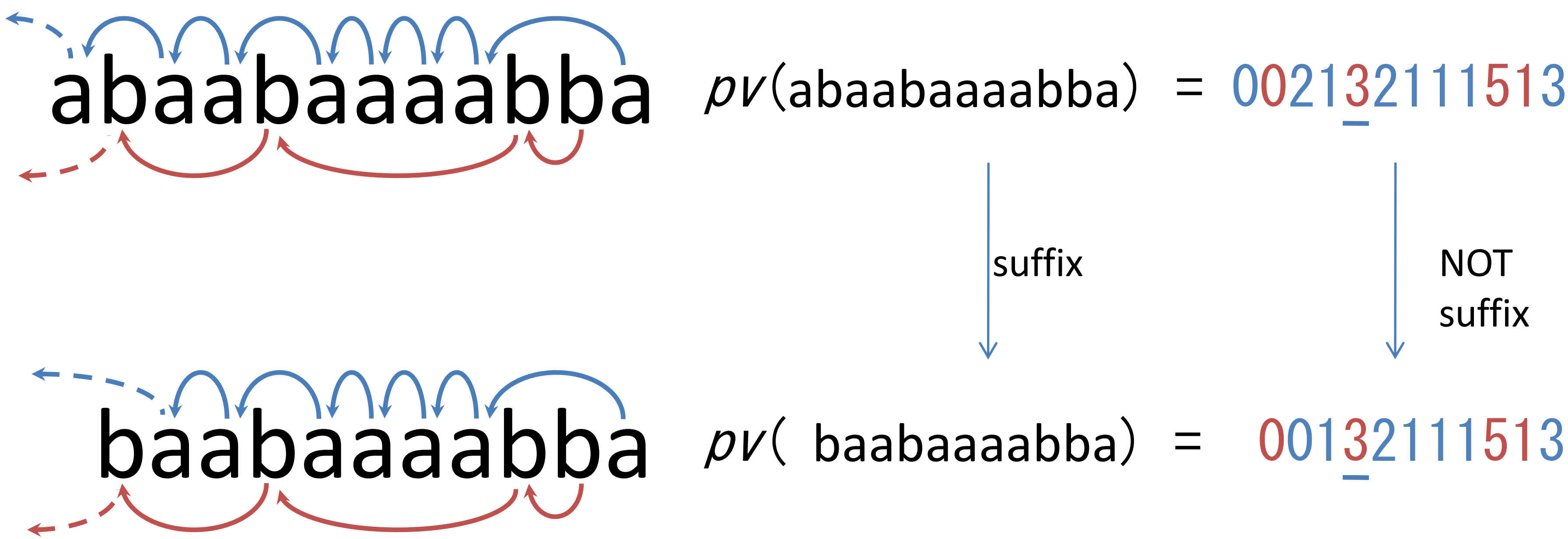
This work

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*



Difficulties of *pv*

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

pv(baabaaaabba) = 00132111513 $<$ *pv*(abaabaaaabba) = 002132111513

↓ suffix ↓ suffix

pv(aabaaaabba) = 0102111513 $>$ *pv*(baabaaaabba) = 00132111513

➡ 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

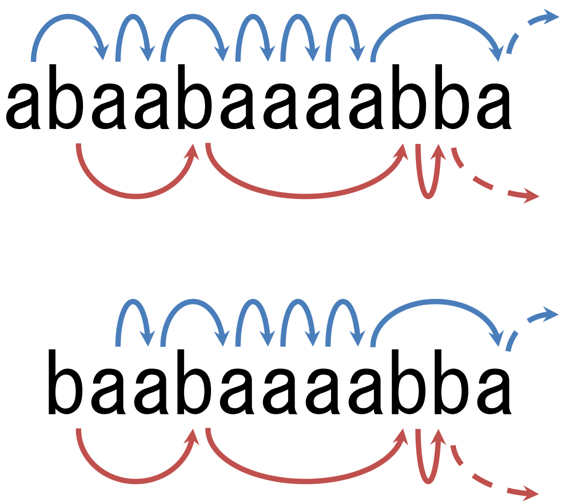
Observation

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 5 1 3
4	9	abba	0 0 1 3
5	2	baabaaaabba	0 0 1 3 2 1 1 1 5 1 3
6	4	abaaaabba	0 0 2 1 1 1 5 1 3
7	1	abaabaaaabba	0 0 2 1 3 2 1 1 1 5 1 3
8	10	bba	0 1 0
9	8	aabba	0 1 0 1 3
10	3	aabaaaabba	0 1 0 2 1 1 1 5 1 3
11	7	aaabba	0 1 1 0 1 3
12	6	aaaabba	0 1 1 1 0 1 3

Monotonically decrease?

fw array

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



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

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 $\underline{5}$ 1 3	$\underline{5}$ 1 1 1 3 1 $\infty\infty$
4	9	abba	0 0 1 $\underline{3}$	$\underline{3}$ 1 $\infty\infty$
[illegible]	[illegible]	[illegible]	0 0 1 3 2 1 1 1 5 1 3	3 1 2 5 1 1 1 3 1 $\infty\infty$
10	3	aabaaaabba	0 $\underline{1}$ 0 2 1 1 1 5 1 3	$\underline{1}$ 2 5 1 1 1 3 1 $\infty\infty$
11	7	aaabba	0 $\underline{1}$ 1 0 1 3	$\underline{1}$ 1 3 1 $\infty\infty$
12	6	aaaabba	0 $\underline{1}$ 1 1 0 1 3	$\underline{1}$ 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.

(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	ab$	
4	1	ababbaab$	
5	3	abbaab$	
6	8	b$	
7	5	baab$	
8	2	babbaab$	
9	4	bbaab$	

lcp[5] ≧ 2–1

lcp[8] ≧ 2–1

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	baaaabba	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
[illegible]	[illegible]	[illegible]abba	01013	3
[illegible]	3	aabaaaabba	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 : $3 - 1 \nleq 1$

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*,

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

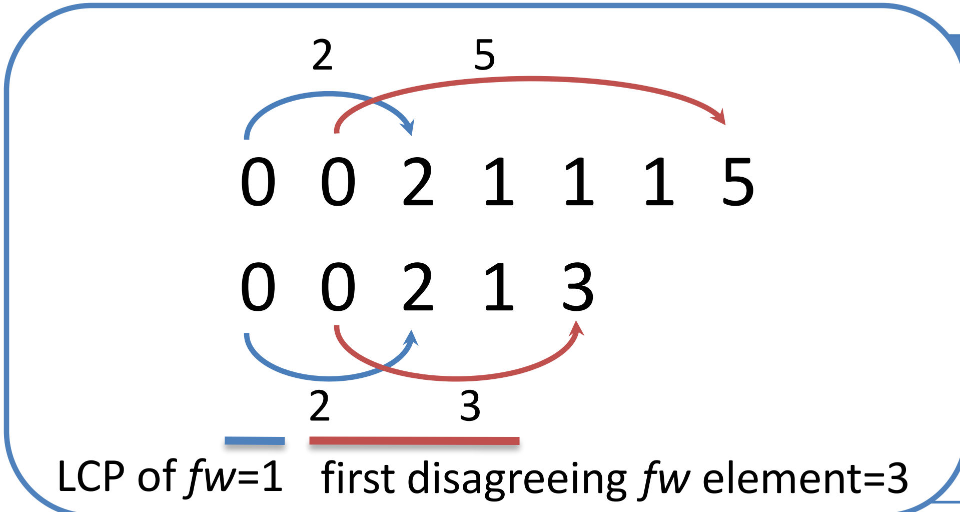
LCP of *fw*
+ first disagreeing *fw* element

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	∞∞	1
3	5	baaaabba	0 0 1 1 1 5 1 3	2	5 1 1 1 3 1 ∞∞	0
4	9	abba	0 0 1 3	3	3 1 ∞∞	0
5	2	baabaaaabba	0 0 1 3 2 1 1 1 5 1 3	4	3 1 2 5 1 1 1 3 1 ∞∞	2
6	4	abaaaabba	0 0 2 1 1 1 5 1 3	2	2 5 1 1 1 3 1 ∞∞	0
7	1	abaabaaaabba	0 0 2 1 3 2 1 1 1 5 1 3	4	2 3 1 2 5 1 1 1 3 1 ∞∞	1
8	10	bba	0 1 0	1	1 ∞∞	0
9	8	aabba	0 1 0 1 3	3	1 3 1 ∞∞	1
10	3	aabaaaabba	0 1 0 2 1 1 1 5 1 3	3	1 2 5 1 1 1 3 1 ∞∞	1
11	7	aaabba	0 1 1 0 1 3	2	1 1 3 1 ∞∞	1
12	6	aaaabba	0 1 1 1 0 1 3	3	1 1 1 3 1 ∞∞	2

LCP of *fw* + first disagreeing *fw* element

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

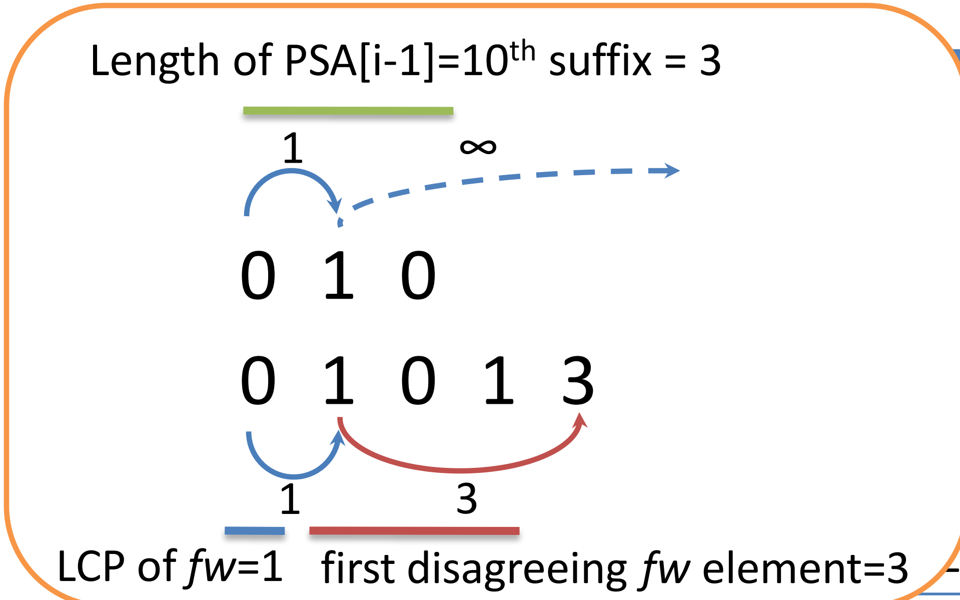


				fw(*s*[PSA[*i*]:*n*])	$LCP_s[i]$
				∞	-1
				∞∞	1
				5 1 1 1 3 1 ∞∞	0
				3 1∞∞	0
				3 1 2 5 1 1 1 3 1∞∞	2
6	4	abaaaabba	2	2 5 1 1 1 3 1∞∞	0
7	1	abaabaaaabba	4	2 3 1 2 5 1 1 1 3 1∞∞	1
8	10	bba	1	1∞∞	0
9	8	aabba	3	1 3 1∞∞	1
10	3	aabaaaabba	3	1 2 5 1 1 1 3 1∞∞	1
11	7	aaabba	2	1 1 3 1∞∞	1
12	6	aaaabba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

1+3

Length of PSA[i-1]th suffix

$$\mathrm{PLCP}_s[i] = \min\{\mathrm{LCP}_{fw(s)}[i] + fw(s)[\mathrm{PSA}[i]+l],\ |s| - \mathrm{PSA}[i-1] + 1\}$$



				$fw(s[\mathrm{PSA}[i]:n])$	$\mathrm{LCP}_s[i]$
				∞	-1
				∞∞	1
				5 1 1 1 3 1 ∞∞	0
				3 1∞∞	0
				3 1 2 5 1 1 1 3 1∞∞	2
				2 5 1 1 1 3 1∞∞	0
7	1	abaabaaaabba	4	2 3 1 2 5 1 1 1 3 1∞∞	1
8	10	bba	1	1∞∞	0
9	8	aabba	3	1 3 1∞∞	1
10	3	aabaaaabba	3	1 2 5 1 1 1 3 1∞∞	1
11	7	aaabba	2	1 1 3 1∞∞	1
12	6	aaaabba	3	1 1 1 3 1∞∞	2

12 −10 + 1 = 3

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

min{ 0+5, 2 }

min{ 2+2, 4 }

min{ 1 + 3, 9 }

Theorem

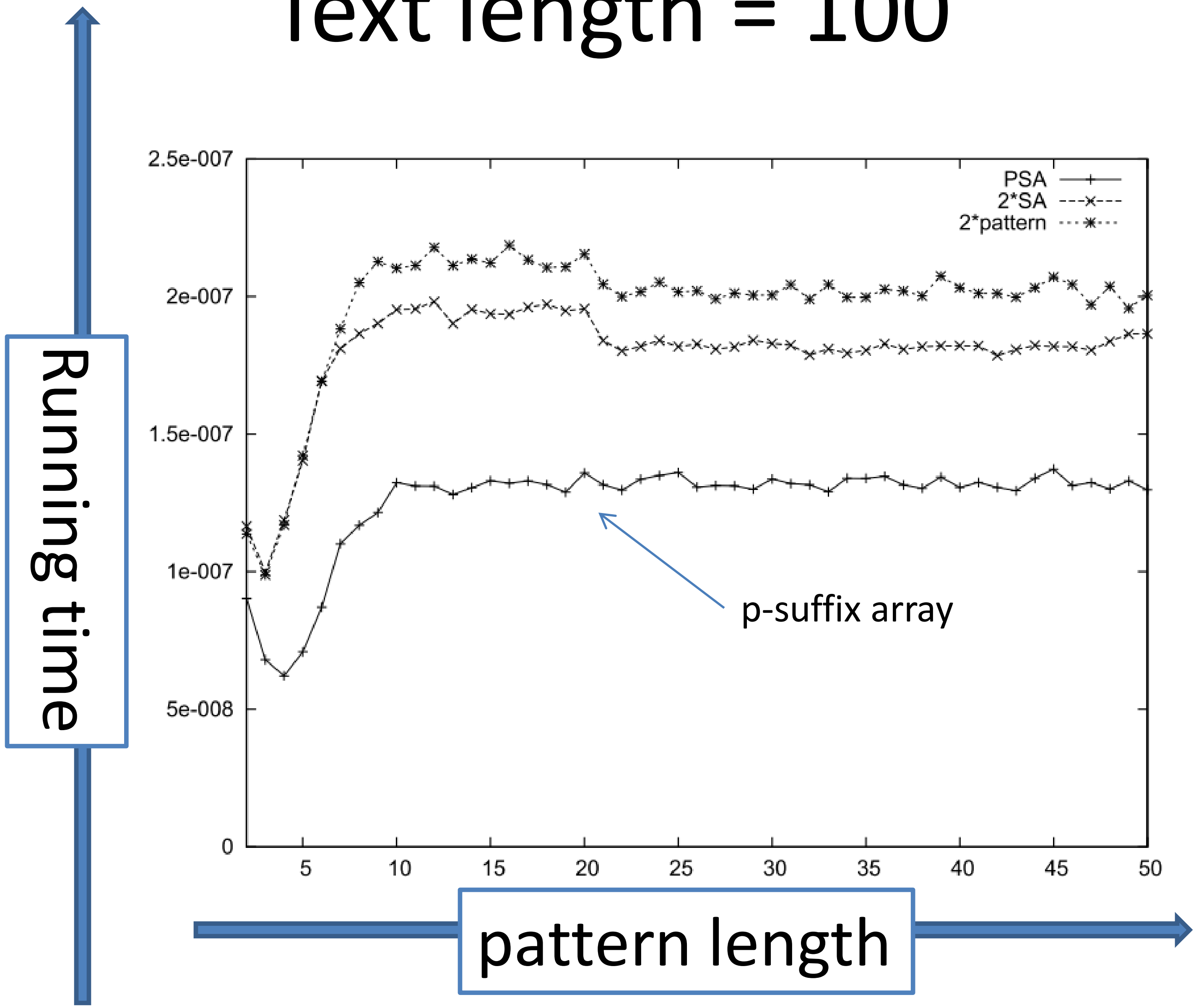
The PLCP array for binary string of length n can be constructed in O(n) time, given its p-suffix array.

Computational Experiments

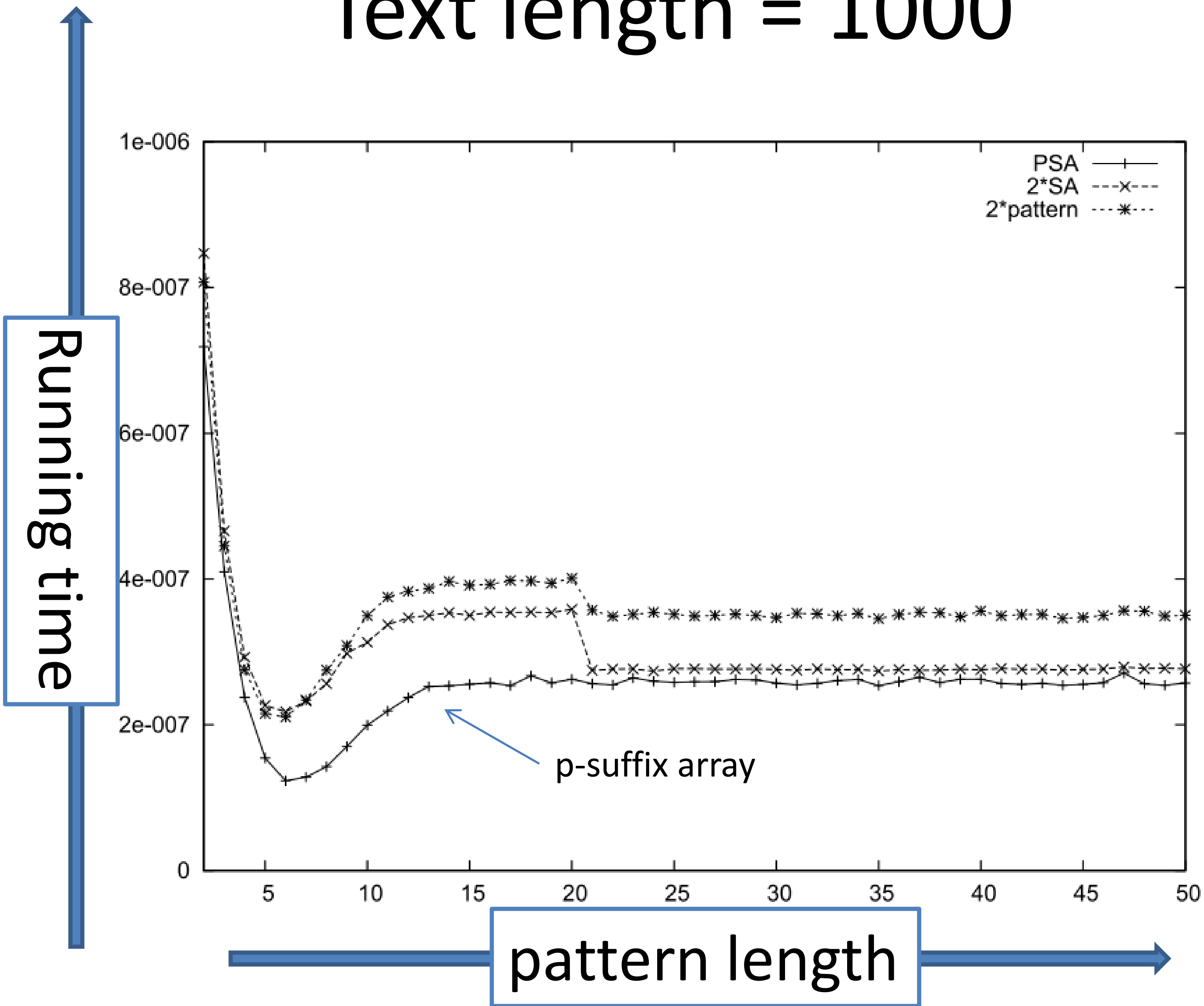
- We compared speed of parameterized pattern matching on random binary strings by:
  - p-suffix array of *t* matched with *pv*(*p*)
  - standard suffix array *t*, matched with *p* and *p'* (bitwise complement)
  - 2 standard suffix arrays *t* and *t*', each matched with *p*

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.