Fast Lightweight Suffix Array Construction (and Checking)

Stefan Burkhardt Juha Kärkkäinen

MPI für Informatik

CPM 2003, Morelia, June 25–27

Yet Another Suffix Array Construction Algorithm

Fast

- $\mathcal{O}(n \log n)$ time for general alphabet
- fast in practice

Lightweight

- sublinear ($\mathcal{O}(n/\sqrt{\log n})$) extra space in addition to input (string) and output (suffix array)
- less than n bytes in practice

Tradeoff

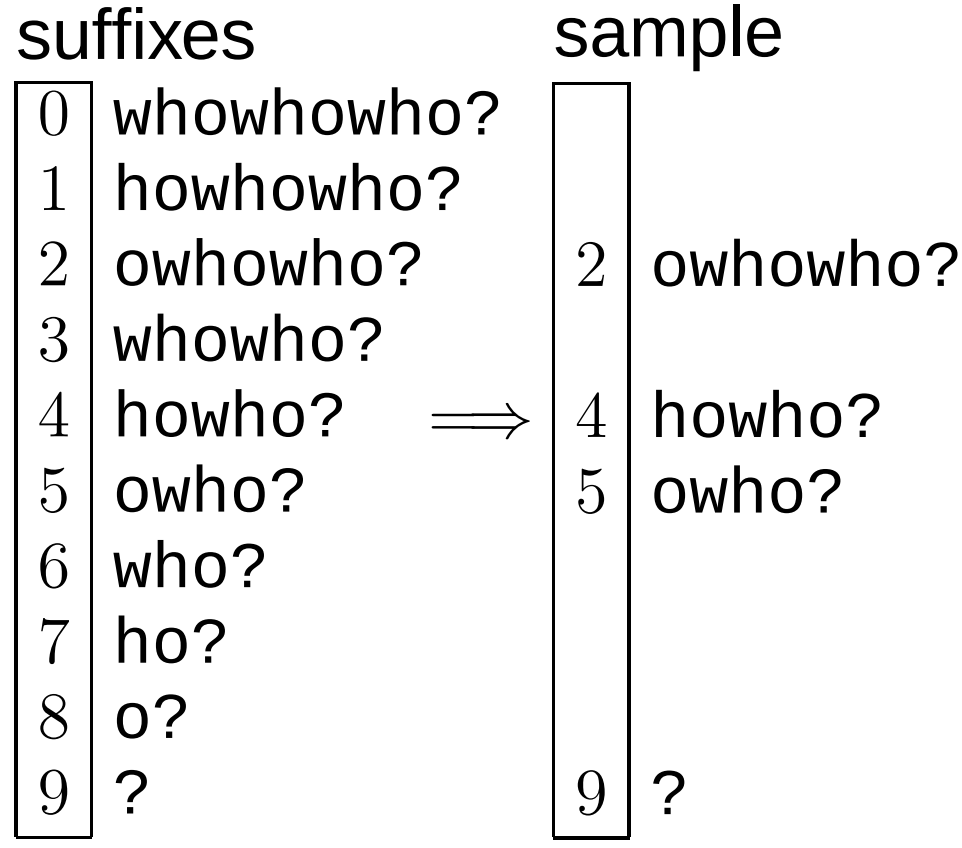
- $\mathcal{O}(vn + n \log n)$ time in $\mathcal{O}(n/\sqrt{v})$ extra space for $v \in [2, n]$

Related work

time	alphabet	extra space	reference
Fast in the worst case			
$\mathcal{O}(n)$ $\mathcal{O}(n \log n)$	integer general	$\geq 4n$ bytes	Larsson & Sadakane '99 many others
Fast on average (string sorting + heuristics)			
$\Omega(n^2)$ worst $\mathcal{O}(n \log n)$ avg.	general	little	Manzini & Ferragina '02 others
Compressed suffix array			
$\mathcal{O}(n \log n)$ $\mathcal{O}(n \log \log \sigma)$	constant integer	$6n$ bits $\mathcal{O}(n \log \sigma)$ bits	Lam & al. '02 Hon & al. FOCS '03

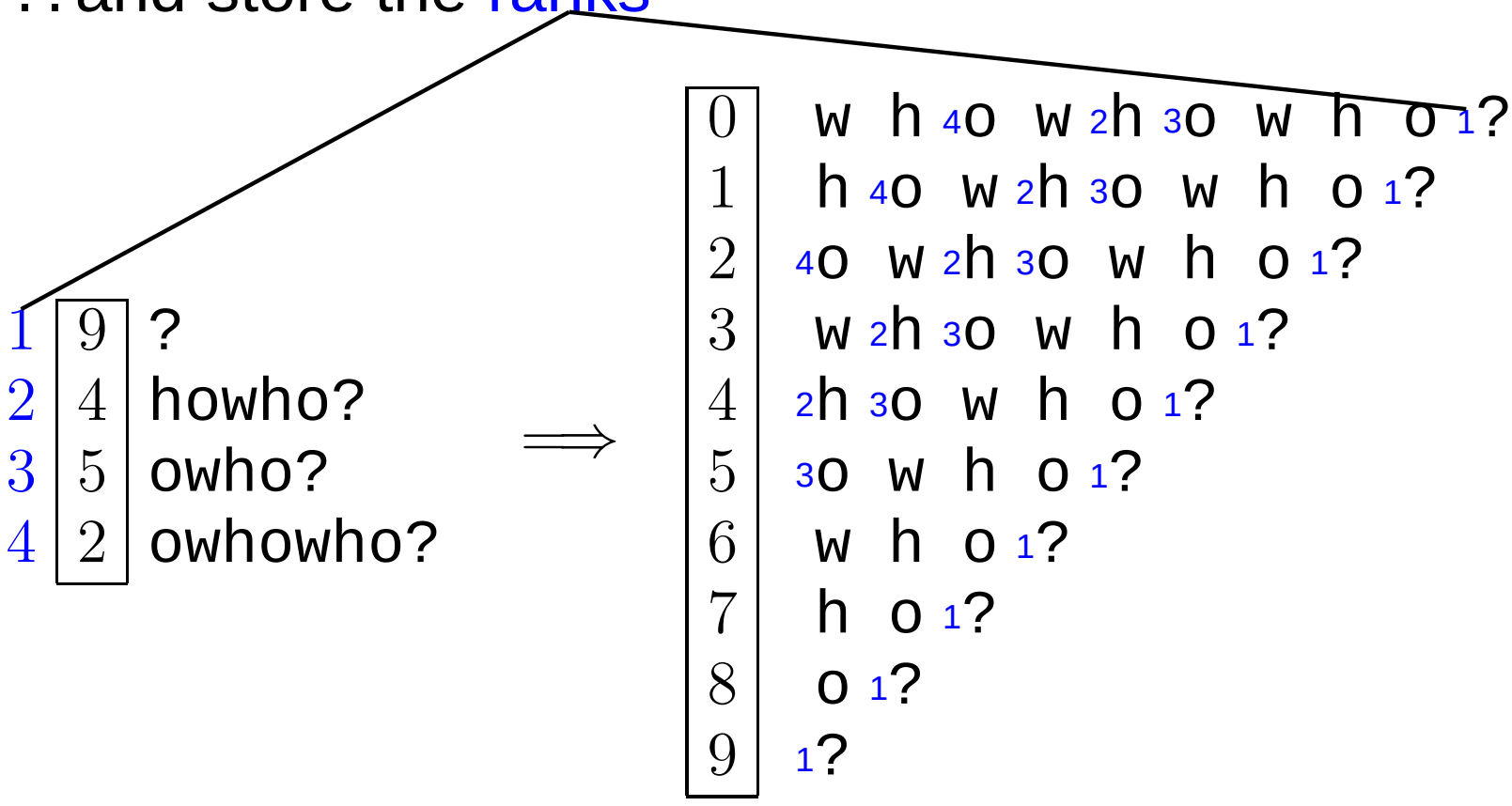
Algorithm Outline

1. Choose a sample of suffixes
2. Sort the sample
3. Sort all suffixes with the help of the sample



Step 2: Sort the sample . . .

. . . and store the ranks



Step 3: Sort all suffixes

Order two suffixes by the first pair of

- mismatching characters or

$\boxed{\begin{matrix}6\\3\end{matrix}}$ w h o$_1$**?**
w$_2$h$_3$o **w** h o$_1$?

$\boxed{\begin{matrix}1\\0\end{matrix}}$ **h**$_4$o w$_2$h$_3$o w h o$_1$?
w h$_4$o w$_2$h$_3$o w h o$_1$?

- aligned sample suffixes (anchor pair)

$\boxed{\begin{matrix}3\\0\end{matrix}}$ w$_2$h$_3$o w h o$_1$?
w h$_4$o w$_2$h$_3$o w h o$_1$?

$\boxed{\begin{matrix}4\\5\end{matrix}}$ **$_2$**h$_3$o w h o$_1$?
$_3$o w h o$_1$?

$\Longrightarrow$ choose the sample to ensure early anchor pairs

How to choose the sample?

Regular sample (every k-th suffix)

- regularity $\Longrightarrow$ fast sorting
- no anchor pairs for most pairs of suffixes

  w 3h o 7w h 5o w 2h o 6w h 4o w 1h o
  7w h 5o w 2h o 6w h 4o w 1h o

Random sample (of size n/k)

- expected distance to an anchor pair is k^2
- sorting time $\Omega(n^2/k)$ in the worst case
- not good for any k

Difference cover sample

Choose a v-periodic sample $\{i \in [0, n) \mid i \bmod v \in D\}$
for a difference cover D modulo v

A difference cover D modulo v is a
subset of $[0, v)$ such that for all $i \in [0, v)$
there exist $j, k \in D$ with $i \equiv k - j \pmod{v}$

Example:
$D = \{2, 4, 5\}$ is a difference cover modulo 7

i	$k - j$
0	$2 - 2$
1	$5 - 4$
2	$4 - 2$
3	$5 - 2$
4	$2 - 5$
5	$2 - 4$
6	$4 - 5$

Difference cover sample

Choose a v-periodic sample $\{i \in [0,n) \mid i \bmod v \in D\}$
for a difference cover D modulo v

A difference cover D modulo v is a subset of $[0,v)$ such that for all $i \in [0,v)$ there exist $j,k \in D$ with $i \equiv k - j \pmod v$

i	$k-j$
0	$2-2$
1	$5-4$
2	$4-2$
3	$5-2$
4	$2-5$
5	$2-4$
6	$4-5$

Example:
$D = \{2,4,5\}$ is a difference cover modulo 7

D-sample:

0 1 2 3 4 5 6	7 8 9 10 11 12 13	14 15 16 17 18 19 20	...

Difference cover sample

Choose a v-periodic sample $\{i \in [0,n) \mid i \bmod v \in D\}$
for a difference cover D modulo v

A difference cover D modulo v is a
subset of $[0,v)$ such that for all $i \in [0,v)$
there exist $j,k \in D$ with $i \equiv k - j \pmod v$

i	$k-j$
0	$2-2$
1	$5-4$
2	$4-2$
3	$5-2$
4	$2-5$
5	$2-4$
6	$4-5$

Example:
$D = \{2,4,5\}$ is a difference cover modulo 7

D-sample:

0 1 2 3 4 5 6	7 8 9 10 11 12 13	14 15 16 17 18 19 20	...

$\leftarrow 10 \equiv 3 \pmod 7 \rightarrow$

0 1 2 3 4 5 6	7 8 9 10 ...

Difference cover sample

Key properties

- ensures an anchor pair within distance v
- closest anchor pair can be found in constant time after $\mathcal{O}(v)$ time and space preprocessing
- [Colbourn & Ling '00]: difference cover of size $\mathcal{O}(\sqrt{v})$ is easy to compute for all $v \Longrightarrow$ sample size $\mathcal{O}(n/\sqrt{v})$
- v-periodic $\Longrightarrow$ sample sorting in $\mathcal{O}(\sqrt{v}n + (n/\sqrt{v})\log n)$ time and $\mathcal{O}(v + n/\sqrt{v})$ space

Difference cover sample algorithm

1. Compute and preprocess the difference cover
   $\mathcal{O}(v)$ time and space
2. Sort the sample
   $\mathcal{O}(\sqrt{v}n + (n/\sqrt{v})\log n)$ time and $\mathcal{O}(v + n/\sqrt{v})$ space
3. Sort all suffixes in place
   (a) Sort suffixes by first v characters
   $\mathcal{O}(vn + n\log n)$ time and $\mathcal{O}(\log n)$ extra space
   (b) Finish sorting using anchor pairs
   $\mathcal{O}(n\log n)$ time and $\mathcal{O}(v + n/\sqrt{v})$ extra space

Total: $\mathcal{O}(vn + n\log n)$ time and $\mathcal{O}(v + n/\sqrt{v})$ extra space
If $v = \log n$: $\mathcal{O}(n\log n)$ time and $\mathcal{O}(n/\sqrt{\log n})$ extra space

Experimental results

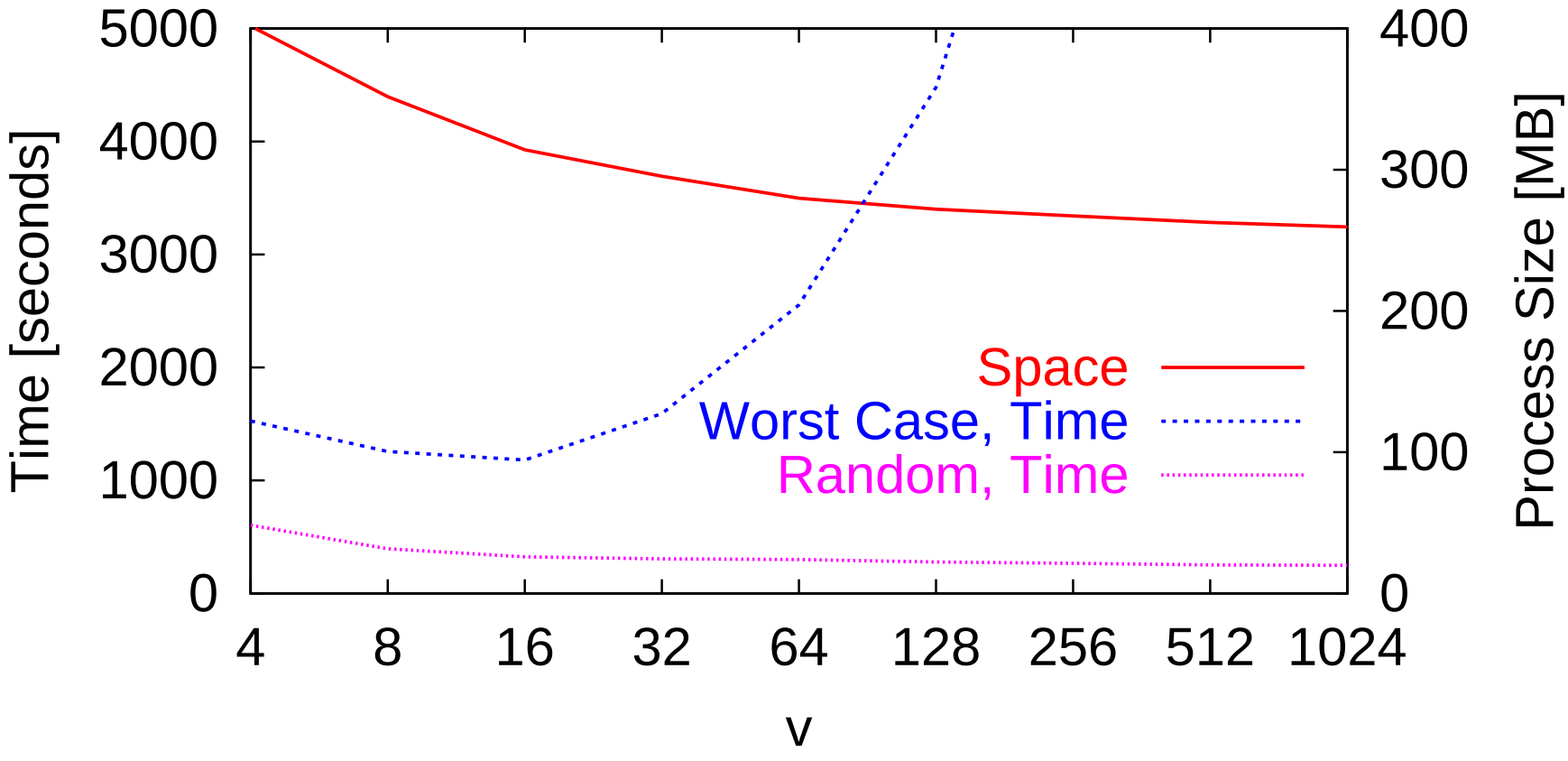
LS = Larsson & Sadakane '99 MF = Manzini & Ferragina '02

DC = difference cover algorithm with $v = 64$

	time [sec]			space [MB]		
text	LS	DC	MF	LS	DC	MF
bible (4.0M)	12	5.5	2.1	33	24	22
e.coli (4.6M)	14	5.6	2.1	38	27	25
gcc source (86M)	529	512	299	694	484	439
(A)^{50M}	160	494	8.3	401	280	252
(1K random)^{50K}	1222	2251	1M[a]	401	280	252
(500K random)^{100}	1311	581	1M[a]	401	280	252

[a] Rough estimate by extrapolation

Experimental results



text length = 50M

alphabet size = 26

Worst Case = (1K random)^{50K}

Conclusion

Summary

- fast and lightweight suffix array construction in theory and practice
- based on difference cover samples

Extensions

- LCP computation
- sorting arbitrary subsets of suffixes

Future work

- still faster implementation