Document Listing on Repetitive Collections

Jouni Sirén
University of Chile

Gonzalo Navarro
University of Chile

Travis Gagie
Simon J. Puglisi
University of Helsinki

Kalle Karhu
Aalto University

Compressed suffix arrays and FM-indexes support the following space-efficiently:

- find(P) = [sp, ep] in $O(|P|)$ time;
- locate(sp, ep) = (SA[sp], ..., SA[ep]) in $O(s \cdot occ)$ time.

Time is measured in rank / select operations on bitmaps, typically taking 0.1 to 1 μs each.

We want to support the most basic document retrieval query:

- docs(sp, ep) = { D[sp], ..., D[ep] }

The brute force solution using locate() takes $O(s \cdot occ)$ time. This assumes that the CSA or FMI has an extra structure to map text positions to document identifiers in $O(1)$ time.

In practice, docc = |docs(sp, ep)| can be much smaller than occ.

For most text collections, the problem has been already been solved. Yet if the collection is repetitive, e.g.

- a set of base documents with their version history
- genomes of individuals of the same species

the extra space taken up by the existing solutions can be much larger than the CSA or FMI itself.

Muthukrishnan’s algorithm

We sort the suffixes of the documents into lexicographic order.

Row	C	D	Suffix
1	0	1	$
2	0	2	$
3	0	3	$
4	2	2	al$
5	3	3	e$
6	4	2	imal$
7	5	3	imize$
8	1	1	imum$
9	6	2	inimal$
10	7	3	inimize$
11	8	1	inimum$
12	10	3	ize$
13	9	2	l$
14	11	1	m$
15	13	2	mal$
16	15	2	minimal$
17	12	3	minimize$
18	14	1	minimum$
19	17	3	mize$
20	18	1	mum$
21	16	2	nimal$
22	19	3	nimize$
23	20	1	nimum$
24	23	1	um$
25	22	3	ze$

We sort the suffixes of the documents into lexicographic order.

Array D stores the document identifier of each suffix.

Row	C	D	Suffix
1	0	1	$
2	0	2	$
3	0	3	$
4	2	2	al$
5	3	3	e$
6	4	2	imal$
7	5	3	imize$
8	1	1	imum$
9	6	2	inimal$
10	7	3	inimize$
11	8	1	inimum$
12	10	3	ize$
13	9	2	l$
14	11	1	m$
15	13	2	mal$
16	15	2	minimal$
17	12	3	minimize$
18	14	1	minimum$
19	17	3	mize$
20	18	1	mum$
21	16	2	nimal$
22	19	3	nimize$
23	20	1	nimum$
24	23	1	um$
25	22	3	ze$

We sort the suffixes of the documents into lexicographic order.

Array D stores the document identifier of each suffix.

Array C points to the previous occurrence of the same document identifier.

Row	C	D	Suffix
1	0	1	$
2	0	2	$
3	0	3	$
4	2	2	al$
5	3	3	e$
6	4	2	imal$
7	5	3	imize$
8	1	1	imum$
9	6	2	inimal$
10	7	3	inimize$
11	8	1	inimum$
12	10	3	ize$
13	9	2	l$
14	11	1	m$
15	13	2	mal$
16	15	2	minimal$
17	12	3	minimize$
18	14	1	minimum$
19	17	3	mize$
20	18	1	mum$
21	16	2	nimal$
22	19	3	nimize$
23	20	1	nimum$
24	23	1	um$
25	22	3	ze$

Query docs(14, 20)

Row	C	D	Suffix
1	0	1	$
2	0	2	$
3	0	3	$
4	2	2	al$
5	3	3	e$
6	4	2	imal$
7	5	3	imize$
8	1	1	imum$
9	6	2	inimal$
10	7	3	inimize$
11	8	1	inimum$
12	10	3	ize$
13	9	2	l$
14	11	1	m$
15	13	2	mal$
16	15	2	minimal$
17	12	3	minimize$
18	14	1	minimum$
19	17	3	mize$
20	18	1	mum$
21	16	2	nimal$
22	19	3	nimize$
23	20	1	nimum$
24	23	1	um$
25	22	3	ze$

Query docs(14, 20)

1. Find the minimum value in C[14, 20]: C[14].

Row	C	D	Suffix
1	0	1	$
2	0	2	$
3	0	3	$
4	2	2	al$
5	3	3	e$
6	4	2	imal$
7	5	3	imize$
8	1	1	imum$
9	6	2	inimal$
10	7	3	inimize$
11	8	1	inimum$
12	10	3	ize$
13	9	2	l$
14	11	1	m$
15	13	2	mal$
16	15	2	minimal$
17	12	3	minimize$
18	14	1	minimum$
19	17	3	mize$
20	18	1	mum$
21	16	2	nimal$
22	19	3	nimize$
23	20	1	nimum$
24	23	1	um$
25	22	3	ze$

Query docs(14, 20)

1. Find the minimum value in C[14, 20]: C[14].
2. If C[14] ≥ 14 (original sp), stop.

Row	C	D	Suffix
1	0	1	$
2	0	2	$
3	0	3	$
4	2	2	al$
5	3	3	e$
6	4	2	imal$
7	5	3	imize$
8	1	1	imum$
9	6	2	inimal$
10	7	3	inimize$
11	8	1	inimum$
12	10	3	ize$
13	9	2	l$
14	11	1	m$
15	13	2	mal$
16	15	2	minimal$
17	12	3	minimize$
18	14	1	minimum$
19	17	3	mize$
20	18	1	mum$
21	16	2	nimal$
22	19	3	nimize$
23	20	1	nimum$
24	23	1	um$
25	22	3	ze$

Query docs(14, 20)

1. Find the minimum value in C[14, 20]: C[14].

2. If C[14] ≥ 14 (original sp), stop.

3. Report D[14].

Row	C	D	Suffix
1	0	1	$
2	0	2	$
3	0	3	$
4	2	2	al$
5	3	3	e$
6	4	2	imal$
7	5	3	imize$
8	1	1	imum$
9	6	2	inimal$
10	7	3	inimize$
11	8	1	inimum$
12	10	3	ize$
13	9	2	l$
14	11	1	m$
15	13	2	mal$
16	15	2	minimal$
17	12	3	minimize$
18	14	1	minimum$
19	17	3	mize$
20	18	1	mum$
21	16	2	nimal$
22	19	3	nimize$
23	20	1	nimum$
24	23	1	um$
25	22	3	ze$

Query docs(14, 20)

1. Find the minimum value in C[14, 20]: C[14].
2. If C[14] ≥ 14 (original sp), stop.
3. Report D[14].
4. Call docs(14, 13) and docs(15, 20).

Row	C	D	Suffix
1	0	1	$
2	0	2	$
3	0	3	$
4	2	2	al$
5	3	3	e$
6	4	2	imal$
7	5	3	imize$
8	1	1	imum$
9	6	2	inimal$
10	7	3	inimize$
11	8	1	inimum$
12	10	3	ize$
13	9	2	l$
14	11	1	m$
15	13	2	mal$
16	15	2	minimal$
17	12	3	minimize$
18	14	1	minimum$
19	17	3	mize$
20	18	1	mum$
21	16	2	nimal$
22	19	3	nimize$
23	20	1	nimum$
24	23	1	um$
25	22	3	ze$

Space requirements in addition to CSA or FMI:

- Array C requires $n \log n$ bits.
- Array D requires $n \log d$ bits.
- RMQ requires $2n + o(n)$ bits.

The algorithm solves docs() in O(docc) time.

Sadakane improved the space usage:

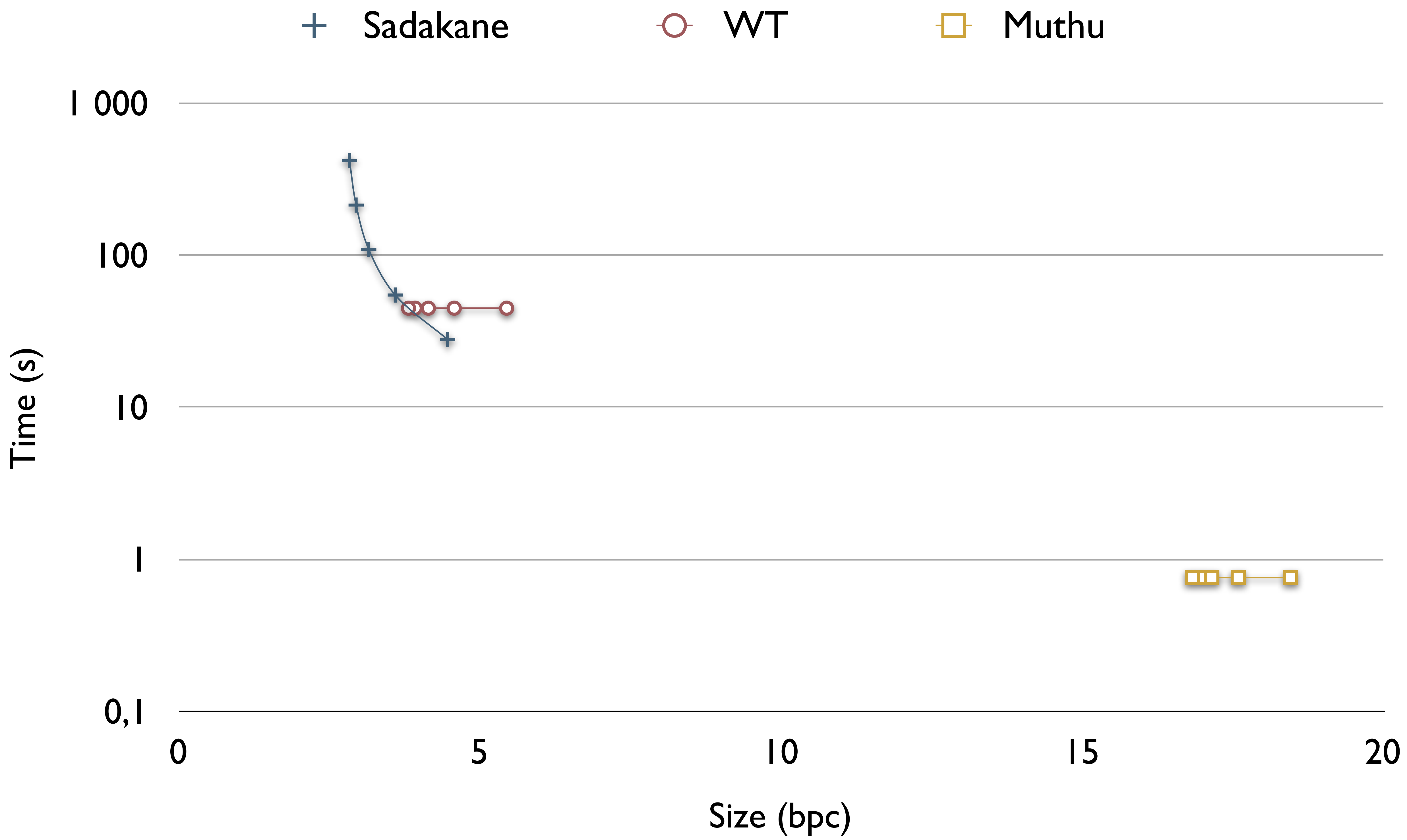
- Use locate() instead of array D.
- Do the recursion in preorder from left to right. $C[i] < sp$ iff $D[i]$ has not been encountered before.

	Time	Space
Muthukrishnan	$O(docc)$	$n \log n + n \log d + 2n + o(n)$
Sadakane	$O(s \cdot docc)$	$2n + o(n)$
Hybrid	$O(docc)$	$n \log d + 2n + o(n)$

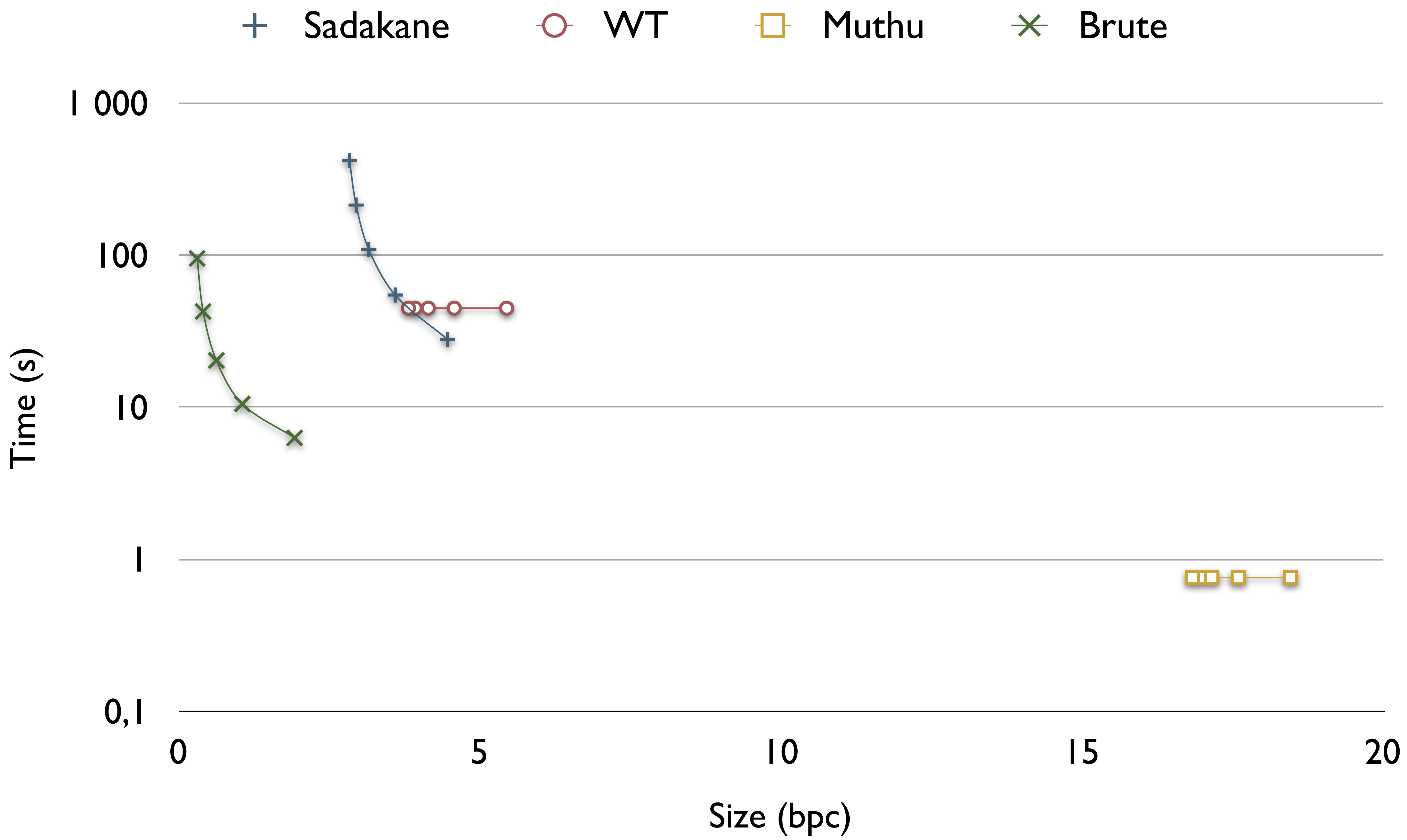
Other solutions

- Use a wavelet tree to list the documents in D[sp, ep] in O(docc log d) time. The best compressed solution is due to Navarro, Puglisi, and Valenzuela (2011).
- Use the top-k document listing structure of Konow and Navarro (2013). It reverts to Sadakane's algorithm in the worst case, but can be faster due to extra structures.

Fiwiki (60 pages, 8834 revisions)
11525 patterns (5.94M occ, 2.84M docc)



Fiwiki (60 pages, 8834 revisions)
11525 patterns (5.94M occ, 2.84M docc)



Interleaved LCP array

LCP[i] is the length of the longest common prefix of suffixes SA[i] and SA[i - 1].

Row	LCP	D	Suffix
1	0	1	$
2	0	1	imum$
3	1	1	inimum$
4	0	1	m$
5	1	1	minimum$
6	1	1	mum$
7	0	1	nimum$
8	0	1	um$
1	0	2	$
2	0	2	al$
3	0	2	imal$
4	1	2	inimal$
5	0	2	l$
6	0	2	mal$
7	1	2	minimal$
8	0	2	nimal$
1	0	3	$
2	0	3	e$
3	0	3	imize$
4	1	3	inimize$
5	1	3	ize$
6	0	3	minimize$
7	2	3	mize$
8	0	3	nimize$
9	0	3	ze$

LCP[i] is the length of the longest common prefix of suffixes SA[i] and SA[i - 1].

The only row in [sp, ep], where LCP[i] < |P|, is sp.

Row	LCP	D	Suffix
1	0	1	$
2	0	1	imum$
3	1	1	inimum$
4	0	1	m$
5	1	1	minimum$
6	1	1	mum$
7	0	1	nimum$
8	0	1	um$
1	0	2	$
2	0	2	al$
3	0	2	imal$
4	1	2	inimal$
5	0	2	l$
6	0	2	mal$
7	1	2	minimal$
8	0	2	nimal$
1	0	3	$
2	0	3	e$
3	0	3	imize$
4	1	3	inimize$
5	1	3	ize$
6	0	3	minimize$
7	2	3	mize$
8	0	3	nimize$
9	0	3	ze$

The ILCP array is built by interleaving the LCP arrays of individual documents according to array D.

Row	ILCP	D	Suffix
1	0	1	$
2	0	2	$
3	0	3	$
4	0	2	al$
5	0	3	e$
6	0	2	imal$
7	0	3	imize$
8	0	1	imum$
9	1	2	inimal$
10	1	3	inimize$
11	1	1	inimum$
12	1	3	ize$
13	0	2	l$
14	0	1	m$
15	0	2	mal$
16	1	2	minimal$
17	0	3	minimize$
18	1	1	minimum$
19	2	3	mize$
20	1	1	mum$
21	0	2	nimal$
22	0	3	nimize$
23	0	1	nimum$
24	0	1	um$
25	0	3	ze$

The ILCP array is built by interleaving the LCP arrays of individual documents according to array D.

ILCP[i] < |P| iff C[i] < sp, so we can use Sadakane's algorithm with ILCP.

Row	ILCP	C	Suffix
1	0	0	$
2	0	0	$
3	0	0	$
4	0	2	al$
5	0	3	e$
6	0	4	imal$
7	0	5	imize$
8	0	1	imum$
9	1	6	inimal$
10	1	7	inimize$
11	1	8	inimum$
12	1	10	ize$
13	0	9	l$
14	0	11	m$
15	0	13	mal$
16	1	15	minimal$
17	0	12	minimize$
18	1	14	minimum$
19	2	17	mize$
20	1	18	mum$
21	0	16	nimal$
22	0	19	nimize$
23	0	20	nimum$
24	0	23	um$
25	0	22	ze$

If the documents are similar, the ILCP array should have long runs of identical values.

Row	ILCP	D	Suffix
1	0	1	$
2	0	2	$
3	0	3	$
4	0	2	al$
5	0	3	e$
6	0	2	imal$
7	0	3	imize$
8	0	1	imum$
9	1	2	inimal$
10	1	3	inimize$
11	1	1	inimum$
12	1	3	ize$
13	0	2	l$
14	0	1	m$
15	0	2	mal$
16	1	2	minimal$
17	0	3	minimize$
18	1	1	minimum$
19	2	3	mize$
20	1	1	mum$
21	0	2	nimal$
22	0	3	nimize$
23	0	1	nimum$
24	0	1	um$
25	0	3	ze$

If the documents are similar, the ILCP array should have long runs of identical values.

We build RMQ only for the run heads.

Heads	ILCP	D	Suffix
X	0	1	$
	0	2	$
	0	3	$
	0	2	al$
	0	3	e$
	0	2	imal$
	0	3	imize$
	0	1	imum$
X	1	2	inimal$
	1	3	inimize$
	1	1	inimum$
	1	3	ize$
X	0	2	l$
	0	1	m$
	0	2	mal$
X	1	2	minimal$
X	0	3	minimize$
X	1	1	minimum$
X	2	3	mize$
X	1	1	mum$
X	0	2	nimal$
	0	3	nimize$
	0	1	nimum$
	0	1	um$
	0	3	ze$

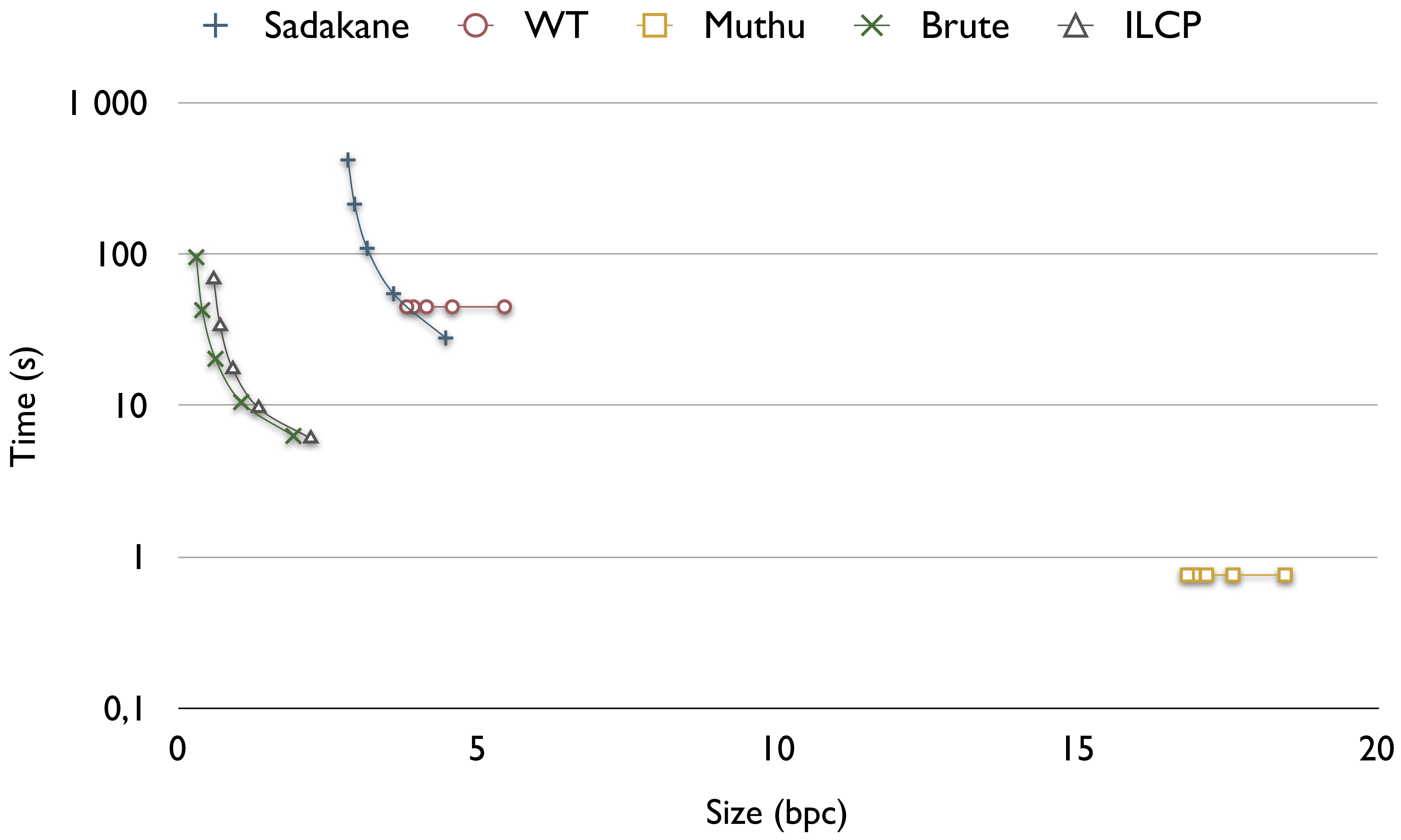
If the documents are similar, the ILCP array should have long runs of identical values.

We build RMQ only for the run heads.

If D[a] has not been encountered, we report the entire run D[a, b], using the faster version of locate().

Heads	ILCP	D	Suffix
X	0	1	$
	0	2	$
	0	3	$
	0	2	al$
	0	3	e$
	0	2	imal$
	0	3	imize$
	0	1	imum$
X	1	2	inimal$
	1	3	inimize$
	1	1	inimum$
	1	3	ize$
X	0	2	l$
	0	1	m$
	0	2	mal$
X	1	2	minimal$
X	0	3	minimize$
X	1	1	minimum$
X	2	3	mize$
X	1	1	mum$
X	0	2	nimal$
	0	3	nimize$
	0	1	nimum$
	0	1	um$
	0	3	ze$

Fiwiki (60 pages, 8834 revisions)
11525 patterns (5.94M occ, 2.84M docc)



Precomputed document listing

The BWT contains the previous character for each suffix.

Row	BWT	D	Suffix
1	m	1	$
2	m	2	$
3	m	3	$
4	m	2	al$
5	z	3	e$
6	n	2	imal$
7	n	3	imize$
8	n	1	imum$
9	m	2	inimal$
10	m	3	inimize$
11	m	1	inimum$
12	m	3	ize$
13	a	2	l$
14	u	1	m$
15	i	2	mal$
16	$	2	minimal$
17	$	3	minimize$
18	$	1	minimum$
19	i	3	mize$
20	i	1	mum$
21	i	2	nimal$
22	i	3	nimize$
23	i	1	nimum$
24	m	1	um$
25	i	3	ze$

The BWT contains the previous character for each suffix.

For a repetitive collection, there should be long runs of identical characters.

Row	BWT	D	Suffix
1	m	1	$
2	m	2	$
3	m	3	$
4	m	2	al$
5	z	3	e$
6	n	2	imal$
7	n	3	imize$
8	n	1	imum$
9	m	2	inimal$
10	m	3	inimize$
11	m	1	inimum$
12	m	3	ize$
13	a	2	l$
14	u	1	m$
15	i	2	mal$
16	$	2	minimal$
17	$	3	minimize$
18	$	1	minimum$
19	i	3	mize$
20	i	1	mum$
21	i	2	nimal$
22	i	3	nimize$
23	i	1	nimum$
24	m	1	um$
25	i	3	ze$

The BWT contains the previous character for each suffix.

For a repetitive collection, there should be long runs of identical characters.

Given a run in BWT

Row	BWT	D	Suffix
1	m	1	$
2	m	2	$
3	m	3	$
4	m	2	al$
5	z	3	e$
6	n	2	imal$
7	n	3	imize$
8	n	1	imum$
9	m	2	inimal$
10	m	3	inimize$
11	m	1	inimum$
12	m	3	ize$
13	a	2	l$
14	u	1	m$
15	i	2	mal$
16	$	2	minimal$
17	$	3	minimize$
18	$	1	minimum$
19	i	3	mize$
20	i	1	mum$
21	i	2	nimal$
22	i	3	nimize$
23	i	1	nimum$
24	m	1	um$
25	i	3	ze$

The BWT contains the previous character for each suffix.

For a repetitive collection, there should be long runs of identical characters.

Given a run in BWT, the preceding suffixes are also adjacent, and the region in D is likely to be identical.

Row	BWT	D	Suffix
1	m	1	$
2	m	2	$
3	m	3	$
4	m	2	al$
5	z	3	e$
6	n	2	imal$
7	n	3	imize$
8	n	1	imum$
9	m	2	inimal$
10	m	3	inimize$
11	m	1	inimum$
12	m	3	ize$
13	a	2	l$
14	u	1	m$
15	i	2	mal$
16	$	2	minimal$
17	$	3	minimize$
18	$	1	minimum$
19	i	3	mize$
20	i	1	mum$
21	i	2	nimal$
22	i	3	nimize$
23	i	1	nimum$
24	m	1	um$
25	i	3	ze$

We split array D into blocks and store the set of document identifiers in each block, using grammar compression.

Sets	Blocks	D	Suffix
1, 2	X	1	$
		2	$
2, 3	X	3	$
		2	al$
2, 3	X	3	e$
		2	imal$
1, 3	X	3	imize$
		1	imum$
2, 3	X	2	inimal$
		3	inimize$
1, 3	X	1	inimum$
		3	ize$
1, 2	X	2	l$
		1	m$
2	X	2	mal$
		2	minimal$
1, 3	X	3	minimize$
		1	minimum$
1, 3	X	3	mize$
		1	mum$
2, 3	X	2	nimal$
		3	nimize$
1	X	1	nimum$
		1	um$
3	X	3	ze$

We split array D into blocks and store the set of document identifiers in each block, using grammar compression.

To answer a docs() query, we take the union of the full blocks in the range, and use brute force for the remaining rows.

Sets	Blocks	D	Suffix
1, 2	X	1	$
		2	$
2, 3	X	3	$
		2	al$
2, 3	X	3	e$
		2	imal$
1, 3	X	3	imize$
		1	imum$
2, 3	X	2	inimal$
		3	inimize$
1, 3	X	1	inimum$
		3	ize$
1, 2	X	2	l$
		1	m$
2	X	2	mal$
		2	minimal$
1, 3	X	3	minimize$
		1	minimum$
1, 3	X	3	mize$
		1	mum$
2, 3	X	2	nimal$
		3	nimize$
1	X	1	nimum$
		1	um$
3	X	3	ze$

We get a better grammar by using suffix tree nodes instead of blocks of fixed size.

Sets	Blocks	D	Suffix
1	X	1	$
2	X	2	$
3	X	3	$
2	X	2	al$
3	X	3	e$
1, 2, 3	X	2	imal$
		3	imize$
		1	imum$
1, 2, 3	X	2	inimal$
		3	inimize$
		1	inimum$
3	X	3	ize$
2	X	2	l$
1	X	1	m$
2	X	2	mal$
1, 2, 3	X	2	minimal$
		3	minimize$
		1	minimum$
		3	mize$
1	X	1	mum$
1, 2, 3	X	2	nimal$
		3	nimize$
		1	nimum$
1	X	1	um$
3	X	3	ze$

We get a better grammar by using suffix tree nodes instead of blocks of fixed size.

Brute force is only needed for intervals contained in a single block.

Sets	Blocks	D	Suffix
1	X	1	$
2	X	2	$
3	X	3	$
2	X	2	al$
3	X	3	e$
1, 2, 3	X	2	imal$
		3	imize$
		1	imum$
1, 2, 3	X	2	inimal$
		3	inimize$
		1	inimum$
3	X	3	ize$
2	X	2	l$
1	X	1	m$
2	X	2	mal$
1, 2, 3	X	2	minimal$
		3	minimize$
		1	minimum$
		3	mize$
1	X	1	mum$
1, 2, 3	X	2	nimal$
		3	nimize$
		1	nimum$
1	X	1	um$
3	X	3	ze$

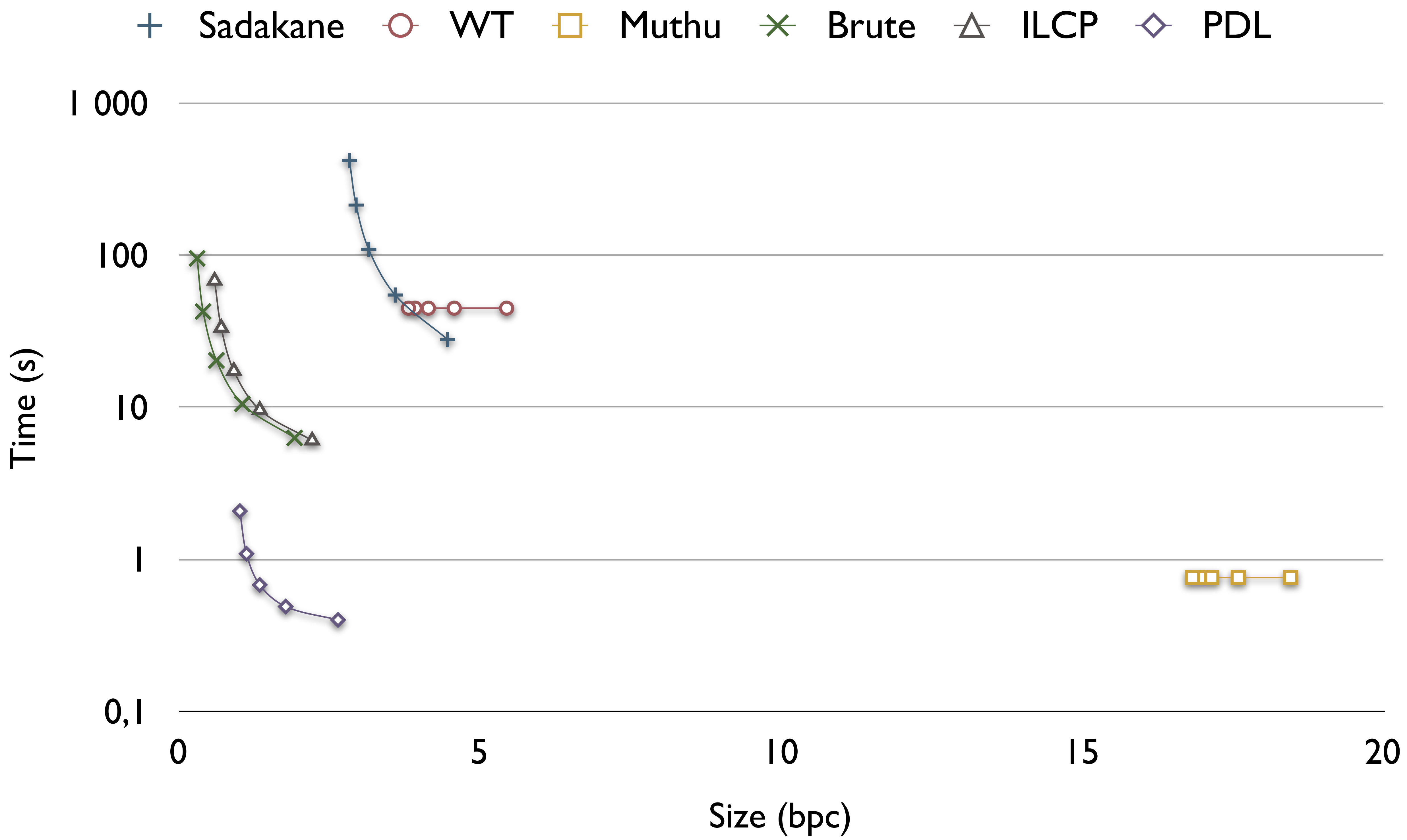
We get a better grammar by using suffix tree nodes instead of blocks of fixed size.

Brute force is only needed for intervals contained in a single block.

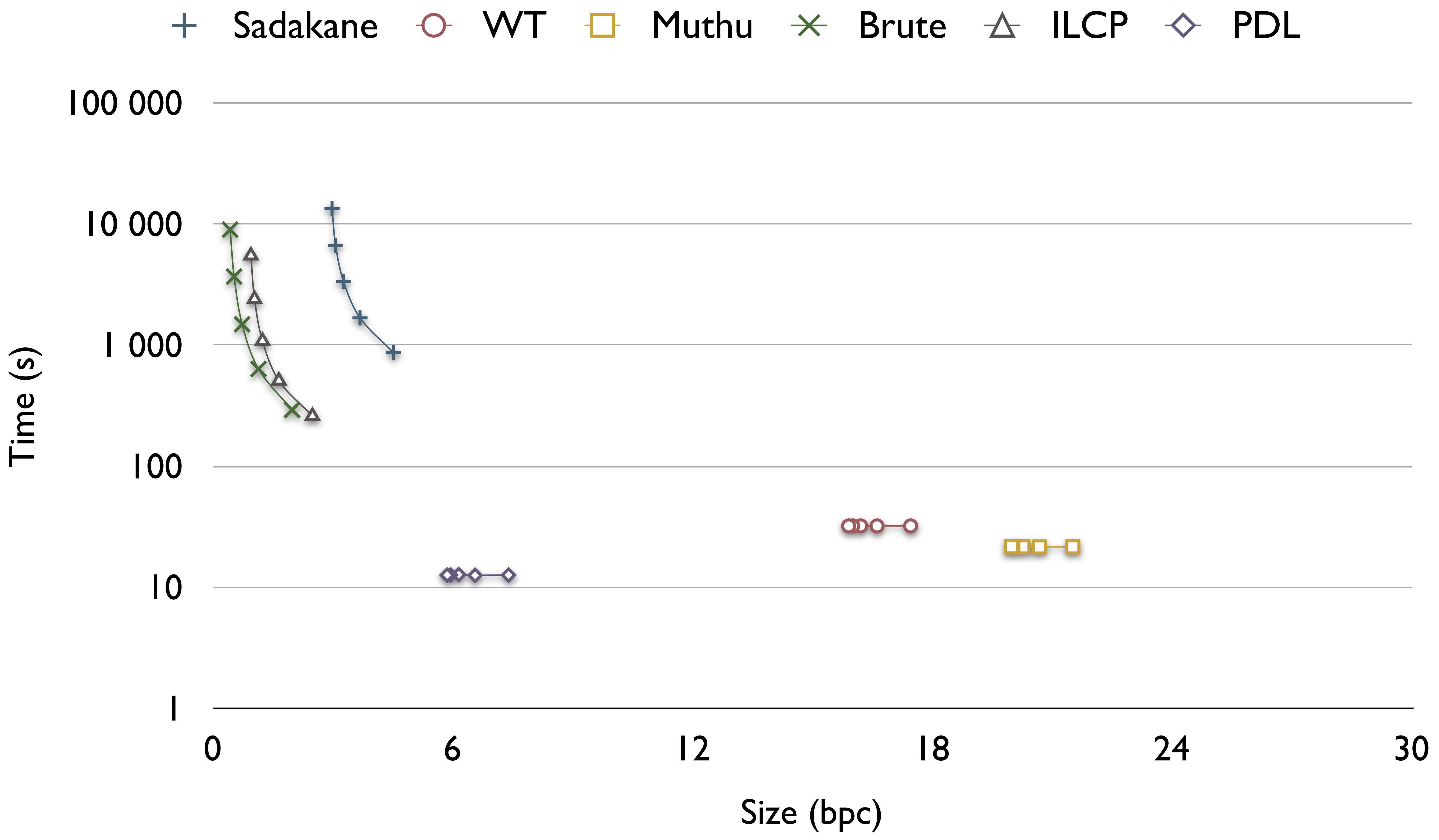
We store the sets for some higher level nodes to make docs() faster on long intervals.

Sets	Blocks	D	Suffix
1	X	1	$
2	X	2	$
3	X	3	$
2	X	2	al$
3	X	3	e$
1, 2, 3	X	2	imal$
		3	imize$
		1	imum$
1, 2, 3	X	2	inimal$
		3	inimize$
		1	inimum$
3	X	3	ize$
2	X	2	l$
1	X	1	m$
2	X	2	mal$
1, 2, 3	X	2	minimal$
		3	minimize$
		1	minimum$
		3	mize$
1	X	1	mum$
1, 2, 3	X	2	nimal$
		3	nimize$
		1	nimum$
1	X	1	um$
3	X	3	ze$

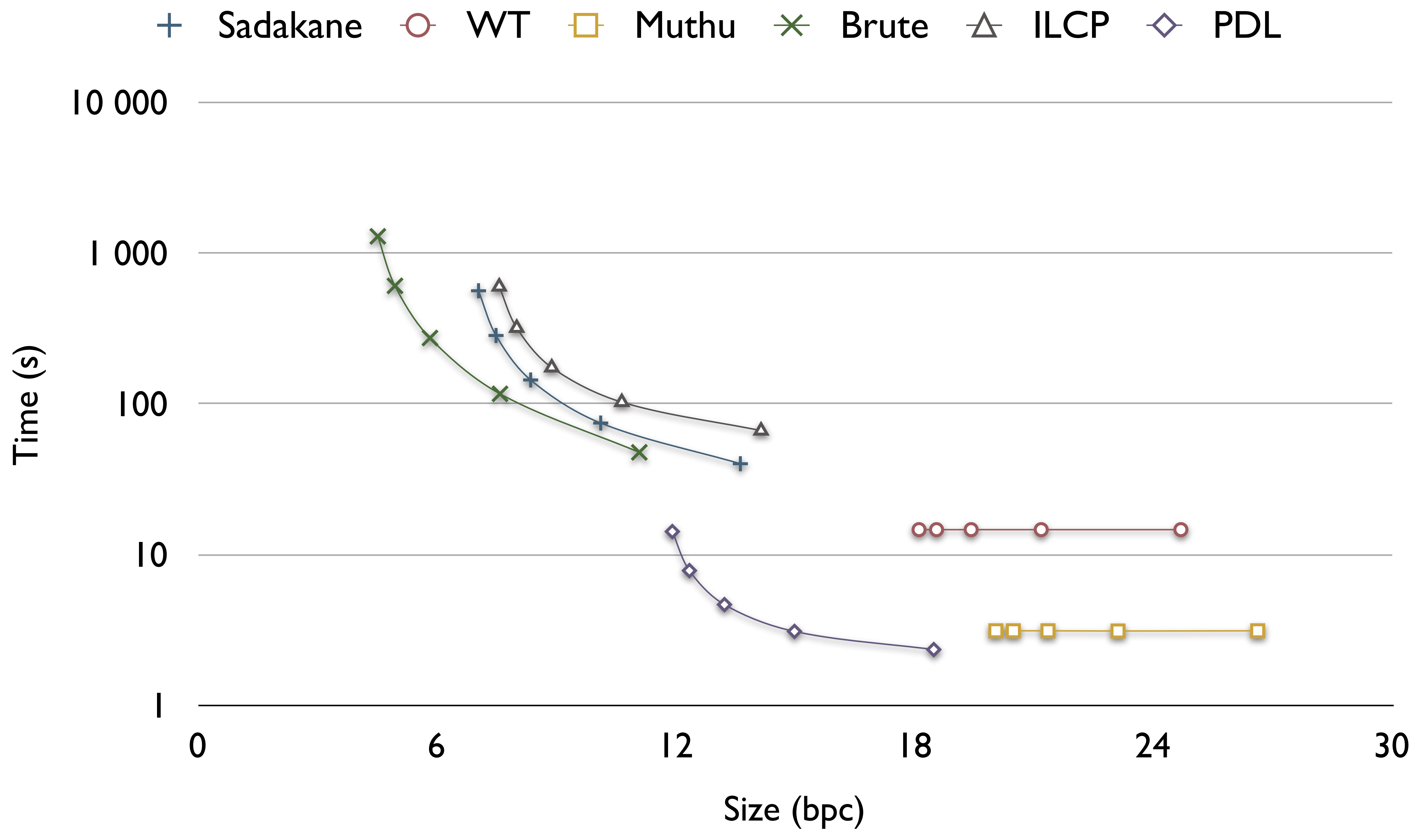
Fiwiki (60 pages, 8834 revisions)
11525 patterns (5.94M occ, 2.84M docc)



Influenza (100k sequences) 1000 patterns (142M occ, 66.2M docc)



Enwiki (7000 pages)
16145 patterns (23.5M occ, 6.35M docc)



Conclusions

- Existing document listing solutions are usually worse than brute force on repetitive collections.
- We proposed two new solutions. ILCP can be marginally better than brute force, while PDL is fast but does not always compress well.
- PDL seems to be competitive on normal collections as well.
- What about other document retrieval queries?

Thank you!