Induced Suffix Sorting for String Collections

Felipe A. Louza*, Simon Gog† and Guilherme P. Telles*

*Institute of Computing
University of Campinas, Brazil

†Institute of Theoretical Informatics
Karlsruhe Institute of Technology, Germany

March 30, 2016

Suffix sorting

- Suffix sorting is important in text indexing and in data compression.
  - It is at the core of many applications.
  - It is easy to obtain the BWT from the suffix array.

Some definitions

- Alphabet Σ, $|\Sigma| = \sigma$, is an ordered set of characters.
- We use $T[i]$ for a character of a string T and $T[i, j]$ for a substring, $1 \leq i \leq j \leq n$.
- Strings are terminated by an end-marker \$ which is the smallest character.
- The suffix array of a string T with length n, SA, is an array of integers in the range $[1, n]$ that gives the lexicographic order of all suffixes of T.

$T =$

1	2	3	4	5	6	7
b	a	n	a	n	a	\$

all suffixes		sorted suffixes		i	SA[i]
banana\$		\$		1	7
anana\$		a\$		2	6
nana\$	*sort* →	ana\$	*suffix array* →	3	4
ana\$		anana\$		4	2
na\$		banana\$		5	1
a\$		na\$		6	5
\$		nana\$		7	3

Suffix arrays construction

- Many suffix array construction algorithms (SACAs) were introduced in the past 25 years.
- This work closely relates to
  - SAIS (2009): linear time, $n \log n + n + O(1)$ bits of memory, fast in practice.
  - SACA-K (2013): linear time, σ words for constant alphabets, fast in practice.

Generalized suffix array

- Let $\mathcal{T} = T_1, T_2, \ldots, T_d$ be a collection of d strings over Σ.
- The generalized suffix array of $\mathcal{T}$ is the suffix array of the concatenation T^{cat} of the strings in $\mathcal{T}$.

Two concatenation alternatives

- Multiple end-markers: $T^{cat} = T_1 \cdot \$_1 \cdot T_2 \cdot \$_2 \cdots T_d \cdot \$_d \cdot \#$
- Single end-marker: $T^{cat} = T_1 \cdot \$ \cdot T_2 \cdot \$ \cdots T_d \cdot \$ \cdot \#$

- End-markers $\# < \$ < \$_1 < \$_2 < \ldots < \$_d$ are not in Σ and are smaller than any symbol in Σ.

Multiple end-markers

$$T^{cat} = T_1 \cdot \$_1 \cdot T_2 \cdot \$_2 \cdots T_d \cdot \$_d \cdot \#$$

- It Increases the alphabet size of the resulting string by the number of strings, which may deteriorate the theoretical bounds and the practical behavior of many SACAs.

Single end-markers

$$T^{cat} = T_1 \cdot \$ \cdot T_2 \cdot \$ \cdots T_d \cdot \$ \cdot \#$$

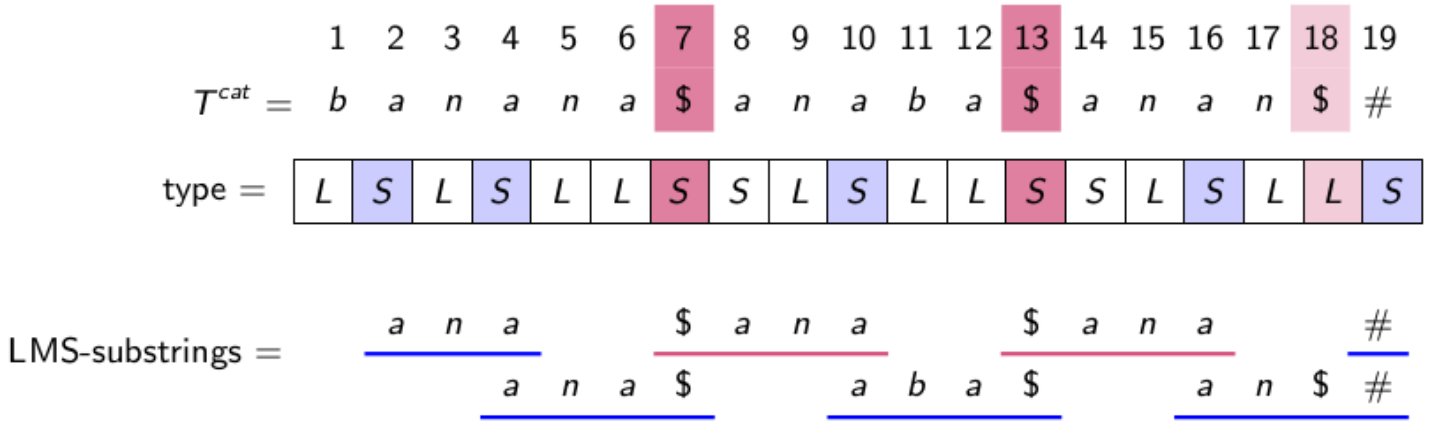
- Ties between suffixes will not be broken by the string rank by many SACAs.
- Suffix comparison may continue after \$, but their symbols shouldn't be compared any further.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$T^{cat} =$	*b*	*a*	*n*	*a*	*n*	*a*	\$	*a*	*n*	*a*	*b*	*a*	\$	*a*	*n*	*a*	*n*	\$	#

Some more definitions and facts

- A suffix $T[i, n]$ is S-type (smaller) if $T[i, n] < T[i + 1, n]$. Otherwise $T[i, n]$ is L-type (larger).
- A suffix $T[i, n]$ is LMS-type (LeftMost S-type) if $T[i, n]$ is S-type and $T[i - 1, n]$ is L-type. The last suffix $T[n, n]$ is LMS-type.
- An LMS-substring is a substring $T[i, j]$ such that $T[i, n]$ and $T[j, n]$ are consecutive LMS-type suffixes. The last suffix $T[n, n]$ is an LMS-substring.
- In a suffix array, the suffixes with the same starting character appear consecutively.
- The blocks of the partition of a suffix array w.r.t the first character are called buckets.

A figure


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
T^{cat} = b a n a n a $ a n a b a $ a n a n $ #
type = L S L S L L S S L S L L S S L S L L S
LMS-substrings =
a n a $ a n a $ a b a $ a n a n $ #

SAIS

SAIS(in T, out SA)

Let $type$ be an array of n booleans
Let T^1 be an array of n integers
Let P be an array of n integers
Let BKT be an array of σ integers
1 Scan T to classify all the suffixes as L or S and store in $type$
2 Scan $type$ once to map all the LMS-substrings in T into P
3 Induced sort all the LMS-substrings using P and BKT
4 Name each LMS-substring in S by its rank and build a new string T^1
5 **if** each character in T^1 is unique
6 Directly compute SA^1 from T^1
7 **else**
8 SAIS(T^1, SA^1)
9 Induced sort SA from SA^1
10 **return**

SACA-K

- SACA-K follows the same framework of SAIS.
- Evaluates suffix types on-the-fly.
- Overlays and reuses memory, and uses a bucket array only at the first recursion level.

Our work

- Shows how to modify SAIS and SACA-K to receive the single end-marker concatenation
  - guaranteeing the relative order among suffixes that are equal up to \$,
  - preserving the theoretical bounds of the original algorithms and
  - avoiding some unnecessary substring sorting.
- The algorithms are named gSAIS and gSACA-K.
- Presents experimental results for different string collections.

Modifications

- Modifications to SAIS and SACA-K are concentrated in the LMS-substring induced sorting.
- They affect only the top recursion level.
- Worst-case analysis is unchanged.

Datasets

Collection	size (GB)	d	n	n/d
pages	3.74	1,000	4,019,585,128	4,019,585
revision	0.39	20,433	419,437,293	20,527
influenza	0.56	394,217	597,471,768	1,516
wikipedia	8.32	3,903,703	8,933,518,792	2,288
reads	2.87	32,621,862	3,082,739,100	94
proteins	15.77	50,825,784	16,931,428,229	333

pages: is a repetitive collection from a snapshot of Wikipedia in Finnish. Each document is composed by one page and its revisions.

revision: is the same as pages, but each revision is a separate document.

influenza: is a repetitive collection of the genomes of influenza viruses.

wikipedia: is a collection of pages from a snapshot of Wikipedia in English.

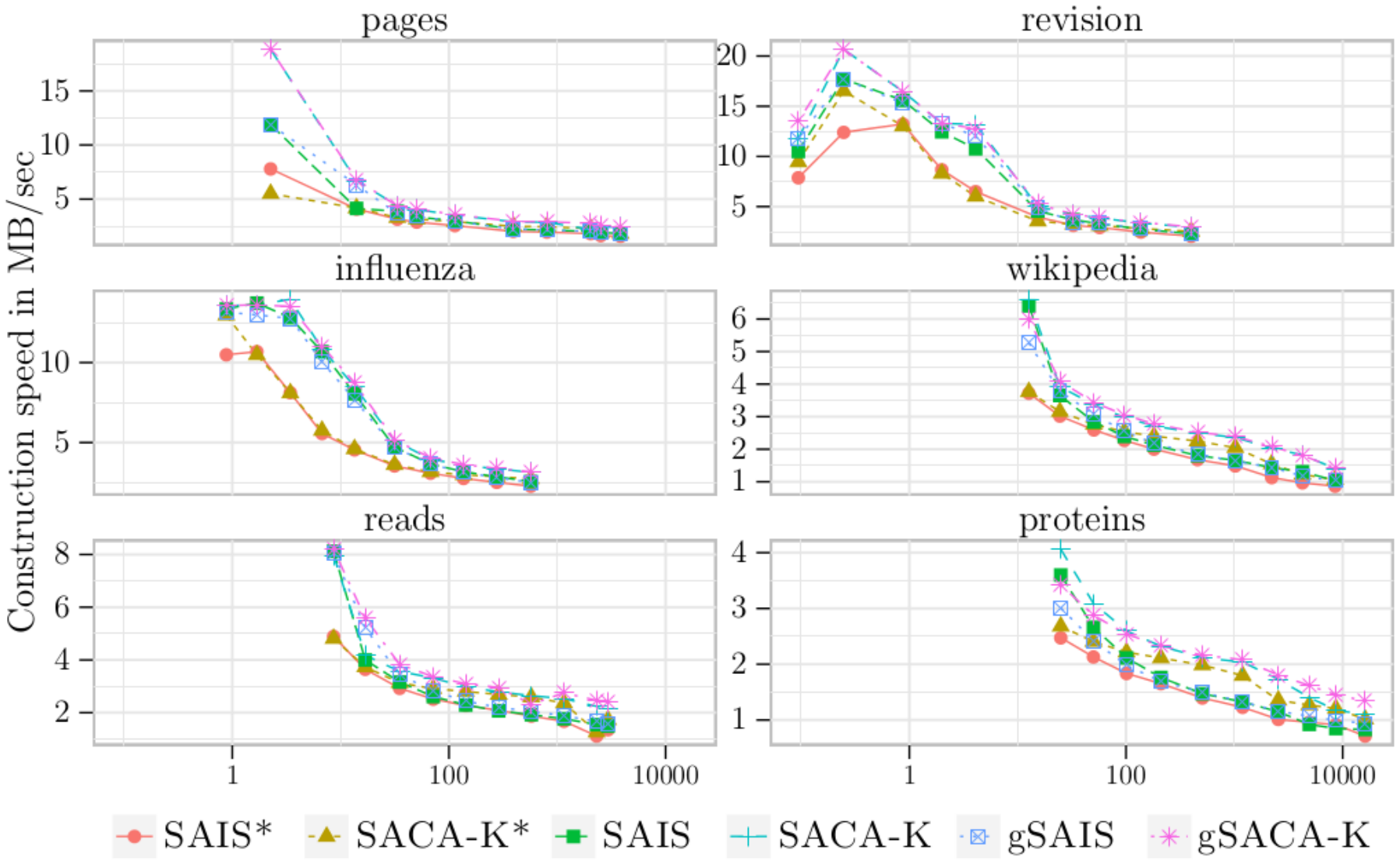
reads: is a collection of reads from Human Chromosome 14 (library 1).

proteins: is a collection of protein sequences from Uniprot/TrEMBL protein database release 2015_09.

Implementations

- The algorithms were implemented in ANSI C based on the source codes of SAIS and SACA-K downloaded from `https://code.google.com/p/ge-nong/`.
- The source code of our algorithms and detailed experimental results are available at `https://github.com/felipelouza/gsa-is`.
- We used 32-bits integers when $n < 2^{31}$, otherwise we used 64-bits integers. SAIS, SACA-K, gSAIS and gSACA-K use 1 byte per symbol and SAIS* and SACA-K* use 4 bytes.
- The collection size was increased (almost) as a ratio 2 geometric progression.

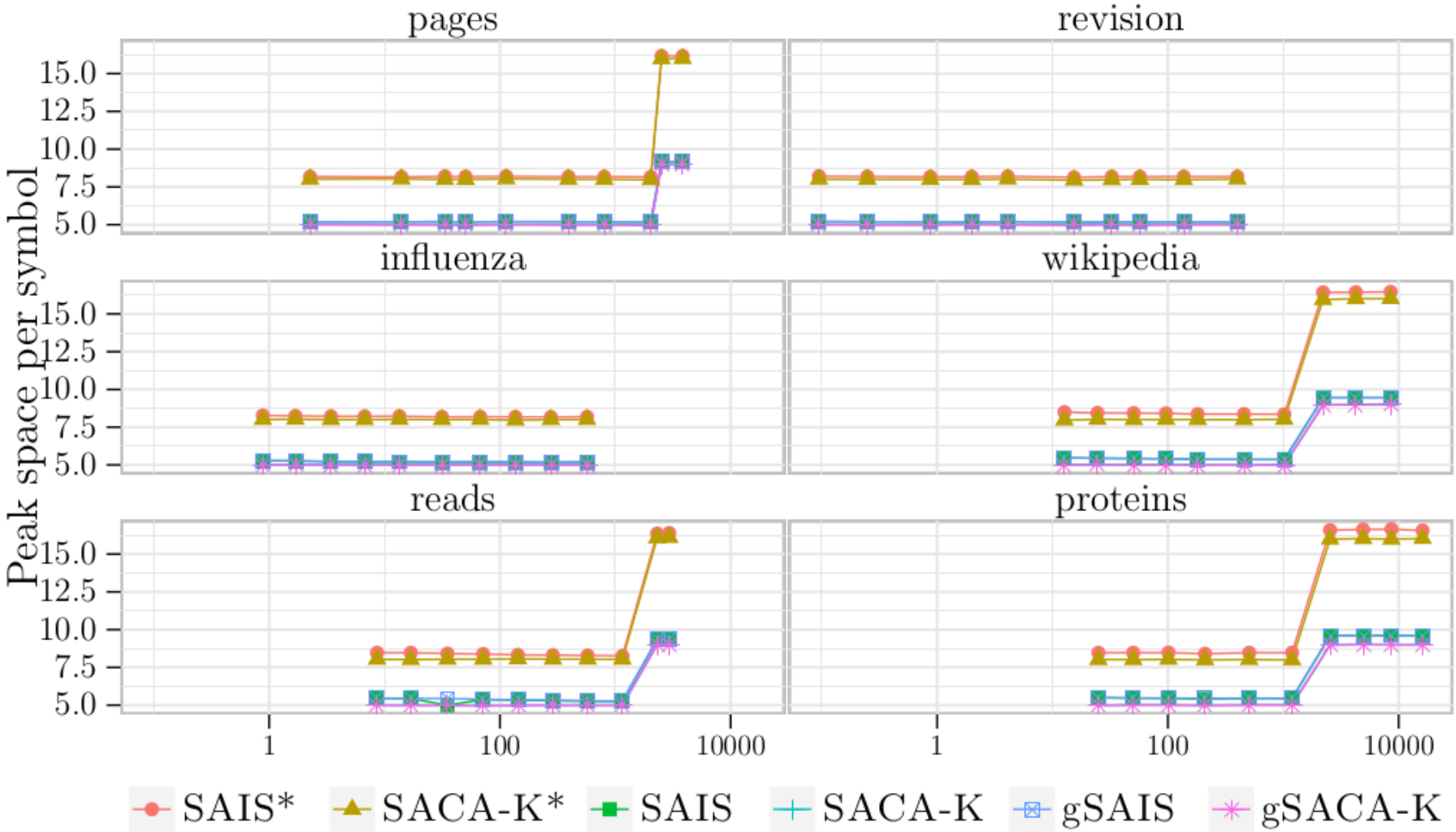
Construction speed



Construction speed

- gSACA-K was almost always the fastest algorithm, specially when the number of strings is large and it avoids some sorting at top recursion level
  - proteins (up to 18.4% faster)
  - reads (up to 10.2% faster)
- The speeds of SACA-K and gSACA-K are similar, but the output of SACA-K (and SAIS) does not guarantee the relative order among suffixes that are equal up to \$.

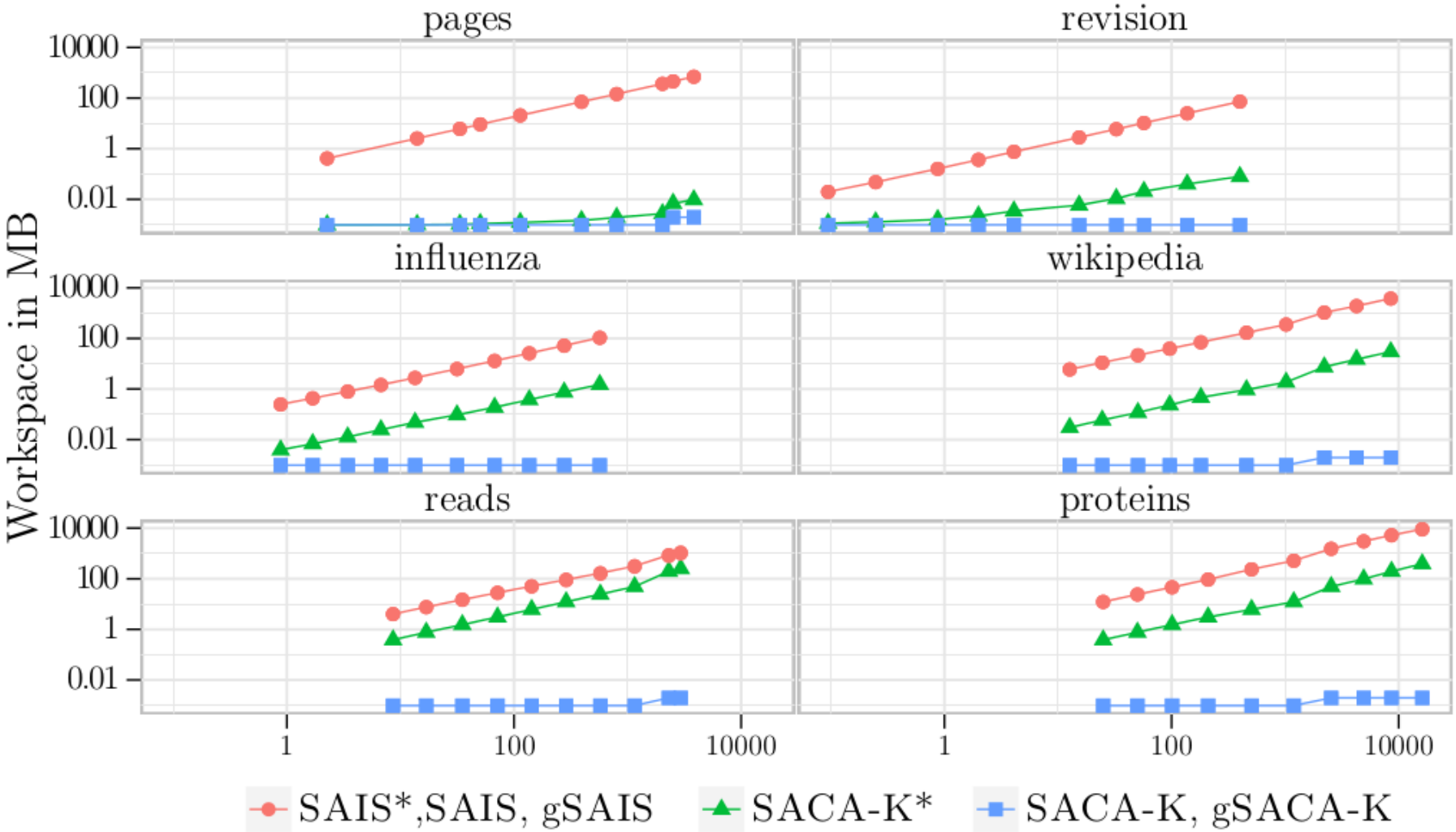
Peak memory


pages
revision
influenza
wikipedia
reads
proteins
Peak space per symbol
15.0
12.5
10.0
7.5
5.0
1
100
10000
SAIS*
SACA-K*
SAIS
SACA-K
gSAIS
gSACA-K

Peak memory

- SAIS^* and $\mathrm{SACA\text{-}K}^*$ spent more memory since T^{cat} is stored in an array of integers, once the d different separators do not fit into the byte alphabet of the input.
- The augmented alphabet also increases the space used by the bucket array BKT.
- Peaks in the graphs are the transitions from 32 to 64 bits.

Workspace


pages
revision
influenza
wikipedia
reads
proteins
Workspace in MB
10000
100
1
0.01
1
100
10000
SAIS*,SAIS, gSAIS
SACA-K*
SACA-K, gSACA-K

Workspace

- Workspace is peak space minus the space used by T^{cat} and by SA.
- The smallest workspaces were by $\mathrm{SACA\text{-}K}$ and $\mathrm{gSACA\text{-}K}$, which are constant for constant alphabets, having $1{,}024$ bytes when $n < 2^{31}$ and $2{,}048$ bytes otherwise.
- The workspace of SAIS^*, SAIS and gSAIS are linearly dependent on the length of T^{cat}
- The workspace of $\mathrm{SACA\text{-}K}^*$ is linearly dependent on the number of strings.

Final remarks

- Overall, gSACA-K's time-space trade-off is the best optimal compared to all the other algorithms in the experiments.
- gSACA-K and gSAIS output a SA where the relative order among suffixes that are equal up to the separator is preserved.

Thank you!

The road not taken

Induced sorting SA from SA^1

1. Scan SA^1 from right to left, and insert each LMS-type suffix of T into the current tail of its bucket in SA.
2. Scan SA from left to right and for each suffix $T[SA[i], n]$ if $T[SA[i]-1, n]$ is L-type, insert $SA[i]-1$ into the current head of its bucket.
3. Scan SA from right to left and for each suffix $T[SA[i], n]$ if $T[SA[i]-1, n]$ is S-type, insert $SA[i]-1$ into the current tail of its bucket.

At the end, the suffixes are sorted in SA.

Induced sorting LMS substrings

1. Scan T from right to left, and insert each corresponding LMS-type suffix of T into the current tail of its c-bucket in SA.
2. Scan SA from left to right and for each suffix $T[SA[i], n]$ if $T[SA[i]-1, n]$ is L-type, insert $SA[i]-1$ into the current head of its bucket.
3. Scan SA from right to left and for each suffix $T[SA[i], n]$ if $T[SA[i]-1, n]$ is S-type, insert $SA[i]-1$ into the current tail of its bucket.

At the end, the LMS-substrings are sorted and stored in their buckets in SA.

Modifications to LMS substrings induced sorting

- In T^{cat} each $\$$ will be an LMS-type suffix, except for the last one.
- Such LMS-type suffixes will generate LMS-substrings that will be sorted by SAIS and $\mathrm{SACA\text{-}K}$ but if two suffixes are equal up to a $\$$ then their symbols shouldn't be compared any further and the order should be given by the string rank in T^{cat}.
- In Step 1, when scanning $T^{cat}[1, n]$ from right to left to bucket-sort all LMS-type suffixes, we do not insert any LMS-type suffix $T^{cat}[j, n]$ in its bucket if the next LMS-type suffix $T^{cat}[i, n]$ to the left starts with a $\$$, for $1 < i < j \leq n$.
- Such positions j are exactly those that will induce the order of the LMS-substrings starting at positions i.

Modifications to LMS substrings induced sorting

- After sorting LMS-substrings we scan $T^{cat}[1, n]$ from left to right, and the LMS-type suffixes starting with \$ are inserted directly into the head of \$-bucket.
- At the end, all LMS-substrings starting with \$ are also sorted and stored into the \$-bucket in SA.

Modifications to LMS substrings induced sorting

- In Step 2, the LMS-type suffixes starting with \$ are inserted into the \$-bucket from $BKT[\$] - 1$, such that $BKT[\$]$ is the tail of the \$-bucket.
- The last position of the \$-bucket is reserved for the last suffix starting with \$, $T^{cat}[n-1, n]$, which is L-type and would otherwise induce $T^{cat}[n-1, n]$ into the head of the \$-bucket.
- We do not induce any L- or S-type suffix $T^{cat}[SA[i] - 1, n]$ if $T^{cat}[SA[i] - 1] = \$$ in Steps 3 and 4.
- This will preserve the relative order among suffixes starting with \$

- Let P be an array holding the indexes of each LMS-substring of T in appearence order.
- The relative order of LMS-suffixes T_i^1 and T_j^1 is the same of $T_{P[i]}^1$ and $T_{P[j]}^1$.

Burrows-Wheeler transform

- A reversible transformation that produces a permutation of a string S which tends to group the occurrences of a symbol into blocks.
- To build the BWT sort all the cyclic rotations of S and get the last column.

i	cyclic rotations	sorted rotations	BWT
0	BANANA$	$BANANA	A
1	$BANANA	A$BANAN	N
2	A$BANAN	ANA$BAN	N
3	NA$BANA	ANANA$B	B
4	ANA$BAN	BANANA$	$
5	NANA$BA	NA$BANA	A
6	ANANA$B	NANA$BA	A

Suffix Array

- An array of integers that stores the order of every suffix of a string in lexicographic order.

	0	1	2	3	4	5	6
T	B	A	N	A	N	A	$

i	$SA[i]$	$T[SA[i], n-1]$
0	6	$
1	5	A$
2	3	ANA$
3	1	ANANA$
4	0	BANANA$
5	4	NA$
6	2	NANA$

lcp, LCP array

- The lcp between two strings S and T is the length of their largest common prefix.
- The LCP array stores the lcp value of consecutive suffixes in the suffix array.

i	$SA[i]$	$LCP[i]$	$T[SA[i], n-1]$
0	6	0	\$
1	5	0	A\$
2	3	1	ANA\$
3	1	3	ANANA\$
4	0	0	BANANA\$
5	4	0	NA\$
6	2	2	NANA\$

Relations

i		F L	SA	LCP	BWT	$T[SA[i], n-1]$
0	BANANA$	$BANANA	6	0	A	$
1	$BANANA	A$BANAN	5	0	N	A$
2	A$BANAN	ANA$BAN	3	1	N	ANA$
3	NA$BANA	ANANA$B	1	3	B	ANANA$
4	ANA$BAN	BANANA$	0	0	$	BANANA$
5	NANA$BA	NA$BANA	4	0	A	NA$
6	ANANA$B	NANA$BA	2	2	A	NANA$

- $L[i] = BWT[i]$ and $F[i] = T[SA[i]]$.
- The j-th symbol in the BWT is the symbol that precedes the j-th suffix (in order).
  The 0-th symbol in the BWT precedes $T[n-1]$.
- $BWT[i] = T[SA[i]-1]$ if $SA[i] \neq 0$ or $BWT[i] = \$$ if $SA[i] = 0$.