A Grammar Compression Algorithm based on Induced Suffix Sorting

D. S. N. Nunes [1,2] F. Louza [3] S. Gog[4] M. Ayala-Rincón [2]
G. Navarro [5]

[1]Federal Institute of Education, Science and Technology of Brasília, Brazil

[2]Department of Computer Science, University of Brasília, Brazil

[3]Department of Computing and Mathematics, University of São Paulo, Brazil

[4]Institute of Theoretical Informatics, Karlsruhe Institute of Technology, Germany

[5]Department of Computer Science, University of Chile, Chile

28th March 2018
Data Compression Conference
Snowbird, Utah, U.S.

Summary

1. Introduction
2. GCIS
3. Results
4. Final Considerations

Summary

1. Introduction

Introduction

- Lossless data compression reduce space requirement by identifying and eliminating redundancy.
- Useful in practice: reduce resources required to store and transmit data.
- Space/Time trade-off.

Introduction

- The Suffix Array Data Structure is used extensively in Stringology.
- Key to solve several text-related problems in efficient or optimal time, using a small footprint of memory, if compared to other data structures.
  - Exact Pattern Matching;
  - Approximate Pattern Matching;
  - LZ factorization;
  - Finding repeats.

Suffix Array

i	$A[i]$	$T_{A[i]}$
0	22	0
1	21	a0
2	18	aana0
3	13	aananaana0
4	8	aananaananaana0
5	3	aananaananaananaana0
6	19	ana0
7	16	anaana0
8	11	anaananaana0
9	6	anaananaananaana0
10	1	anaananaananaananaana0
11	14	ananaana0
12	9	ananaananaana0
13	4	ananaananaananaana0
14	0	banaananaananaananaana0
15	20	na0
16	17	naana0
17	12	naananaana0
18	7	naananaananaana0
19	2	naananaananaananaana0
20	15	nanaana0
21	10	nanaananaana0
22	5	nanaananaananaana0

Nong's Algorithm

- Nong *et al.* algorithm is capable of sorting the suffixes of a text in linear optimal time.
- Very fast in practice.
- Induces the order of suffixes based in the already calculated order of other suffixes.

Our Contribution

- Develop a novel grammar compression algorithm based on the induced suffix sorting algorithm from Nong *et al.* to compress the original string.
  - Faster in compression than 7-ZIP and RE-PAIR.
  - Lower memory peak under compression than RE-PAIR.

Summary

2 GCIS

GCIS

We adapt SAIS framework from Nong *et al.* to build our grammar:

1. Classify suffixes into "S", "L", or "LMS" types.
2. Sort all LMS-suffixes by its first symbol.
3. Induce:
   1. L-type suffixes from LMS-type suffixes;
   2. S-Type suffixes from L-type suffixes;
4. Rename the LMS-Substrings to obtain T' and create grammar rules;
5. If there are equal renamed factors, solve the problem recursively for T'. Else, store T explicitly.
6. LMS-suffixes are now sorted regarding all symbols. Repeat 3.1 and 3.2 to obtain the suffix array.
7. ~~LMS-suffixes are now sorted regarding all symbols. Repeat 3.1 and 3.2 to obtain the suffix array.~~

GCIS

Example

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
T	b	a	n	a	a	n	a	n	a	a	n	a	n	a	a	n	a	n	a	a	n	a	0

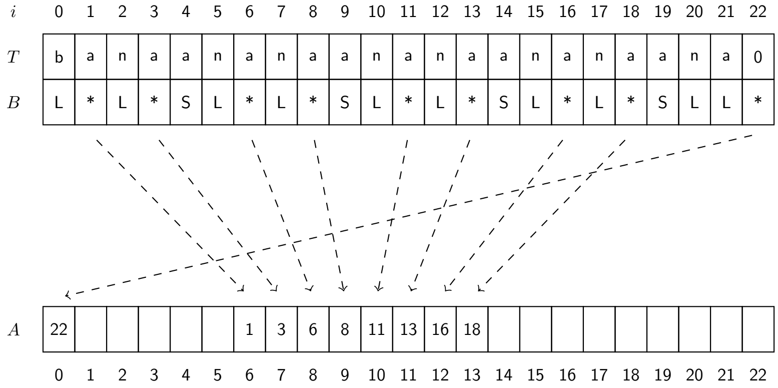
GCIS

Classification

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
T	b	a	n	a	a	n	a	n	a	a	n	a	n	a	a	n	a	n	a	a	n	a	0
B	L	*	L	*	S	L	*	L	*	S	L	*	L	*	S	L	*	L	*	S	L	L	*

GCIS

Radixsort

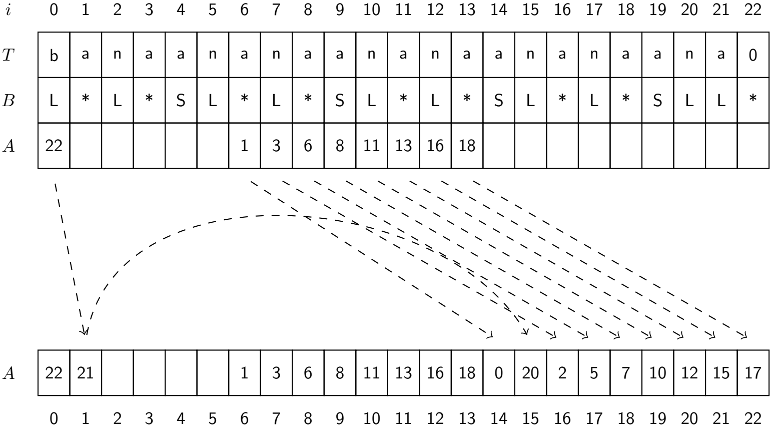


i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
T	b	a	n	a	a	n	a	n	a	a	n	a	n	a	a	n	a	n	a	a	n	a	0
B	L	*	L	*	S	L	*	L	*	S	L	*	L	*	S	L	*	L	*	S	L	L	*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
A	22						1	3	6	8	11	13	16	18									

GCIS

Inducing L-Type Suffixes

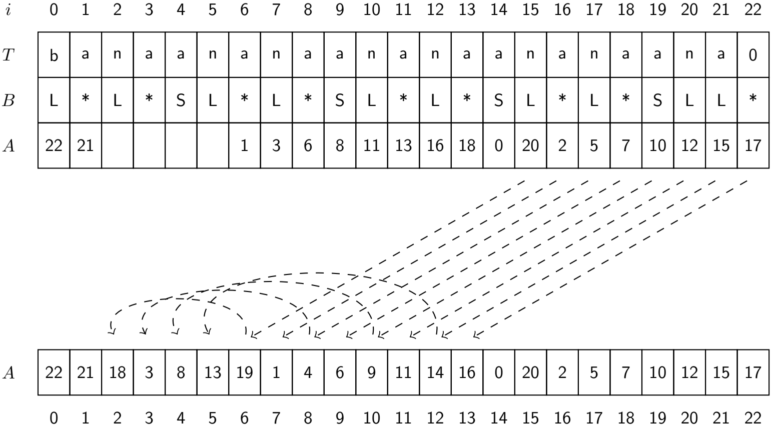


i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
T	b	a	n	a	a	n	a	n	a	a	n	a	n	a	a	n	a	n	a	a	n	a	0
B	L	*	L	*	S	L	*	L	*	S	L	*	L	*	S	L	*	L	*	S	L	L	*
A	22						1	3	6	8	11	13	16	18									

A	22	21					1	3	6	8	11	13	16	18	0	20	2	5	7	10	12	15	17
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

GCIS

Inducing S-Type Suffixes



i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
T	b	a	n	a	a	n	a	n	a	a	n	a	n	a	a	n	a	n	a	a	n	a	0
B	L	*	L	*	S	L	*	L	*	S	L	*	L	*	S	L	*	L	*	S	L	L	*
A	22	21					1	3	6	8	11	13	16	18	0	20	2	5	7	10	12	15	17

A	22	21	18	3	8	13	19	1	4	6	9	11	14	16	0	20	2	5	7	10	12	15	17
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

GCIS

Definition (LMS-Substring)

A LMS-Substring is either the sentinel symbol 0 or a substring $T[i, j-1]$ with both $T[i]$ and $T[j]$ being of type LMS and there is no other LMS symbols for $i \neq j$. We denote this substring by $sub(i)$.

- After inducing L and S-type suffixes, all LMS-substrings are sorted.
- All LMS-substrings are renamed according to their order.
- A pairwise comparison should be done to check if the LMS-substrings are equal.
- T is replaced by the renamed factors, giving place to T'.

GCIS

Renaming

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
T	b	a	n	a	a	n	a	n	a	a	n	a	n	a	a	n	a	n	a	a	n	a	0
B	L	*	L	*	S	L	*	L	*	S	L	*	L	*	S	L	*	L	*	S	L	L	*
A	22	21	18	3	8	13	19	1	4	6	9	11	14	16	0	20	2	5	7	10	12	15	17
	*		*	*	*	*		*		*		*		*									

$$\begin{array}{ll} sub(22) = 0 \mapsto 0 & sub(1) = an \mapsto 3 \\ sub(18) = aana \mapsto 1 & sub(6) = an \mapsto 3 \\ sub(3) = aan \mapsto 2 & sub(11) = an \mapsto 3 \\ sub(8) = aan \mapsto 2 & sub(16) = an \mapsto 3 \\ sub(13) = aan \mapsto 2 & \end{array}$$

$$T' = 323232310$$

Creating Rules

- For every unique LMS-substring S a rule of the form $X \to S$ is created.
- In order to obtain a good compression ratio, certain properties shared between the LMS-substrings shall be explored.
- The first property is that the LMS-substrings are sorted!
- We can represent them by a pair (ℓ, s):
    - ℓ: the common prefix length shared with the previous LMS-substring.
    - s the remaining suffix not encoded by ℓ.
- A special rule is created to store the first symbols of T not covered by a LMS-substring.

Encoding the Factors

For $T = banaananaananaananaana0$

- $0 \rightarrow (0,\text{'}0\text{'})$; //'0';
- $1 \rightarrow (0,\text{'}aana\text{'})$; // 'aana'
- $2 \rightarrow (3,\text{''})$; // 'aan'
- $3 \rightarrow (1,\text{'}n\text{'})$; // 'an'
- $Tail \rightarrow b$

Encoding the Factors

- The ℓ values can be encoded succinctly by using `Simple-8b`
- Tries to fit as many fixed-width integer as possible in a 64-bit word.
- With a single memory access, many values are retrieved.

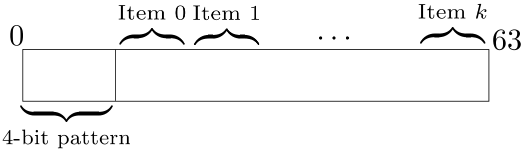


Table: `Simple8b` possible arrangements.

Selector value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Item width	0	0	1	2	3	4	5	6	7	8	10	12	15	20	30	60
Group Size	240	120	60	30	20	15	12	10	8	7	6	5	4	3	2	1
Wasted bits	60	60	0	0	0	0	0	0	4	4	0	0	0	0	0	0

Encoding the Factors

- The s parts are concatenated in a single array of integers with fixed width of $\lceil \lg(|\Sigma|) \rceil$ bits.

$$V = 0aanan$$

- A support bitmap encoding the length of each s part is created.

$$BV = 0100001101$$

- Length of suffix $i = \textsc{select}_1(BV, i+1) - \textsc{select}_1(BV, i) - 1$
- Start of suffix i in V $= \textsc{select}_1(BV, i) - i + 1$.
- To improve space usage, the bitmap is encoded using Elias-Fano.

Decoding

- We begin by the reduced string.
- At each recursion level, we read $T[i] = X$ and replace it by its the correct LMS-substring $X \rightarrow S$.
- To make the decoding faster, all rules from the current level are decompressed previously.

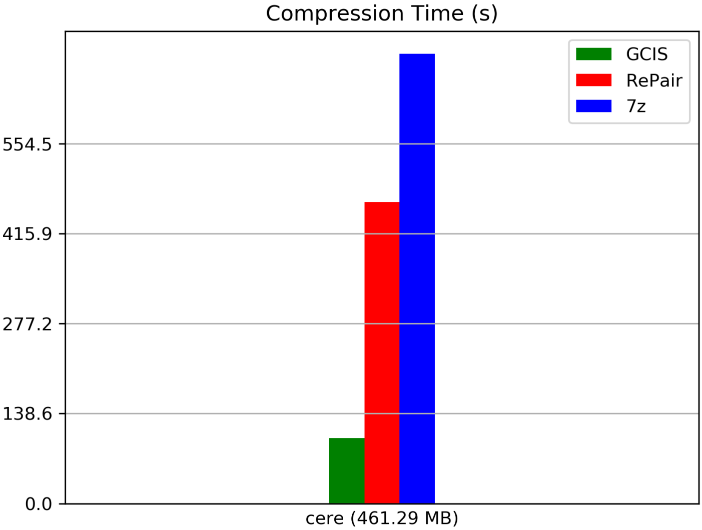
Summary

3. Results

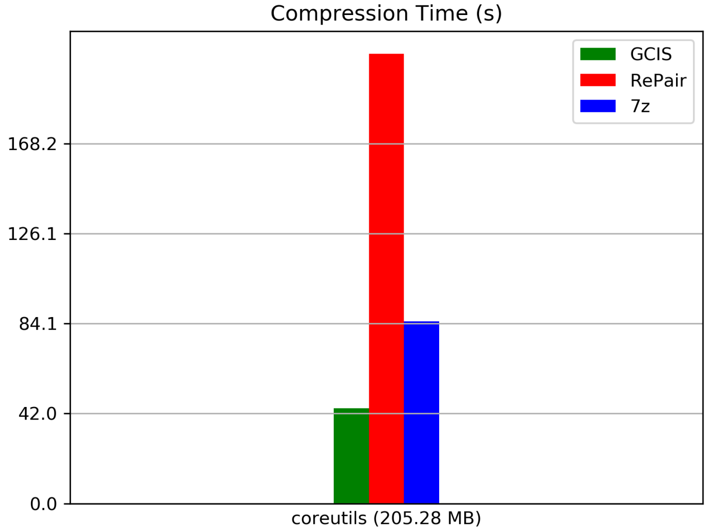
Results

- In order to visualize the potential of the proposed grammar compressor (GCIS), experiments were done with the repetitive Pizza-Chili *corpus*, which was populated with texts of different nature.
- GCIS was compared against popular compressors suited to repetitive sequences: RE-PAIR and 7-ZIP.
- Three subjects were evaluated:
  - Compression ratio: compressedSize/Size.
  - Compression time (s).
  - Decompression time (s).

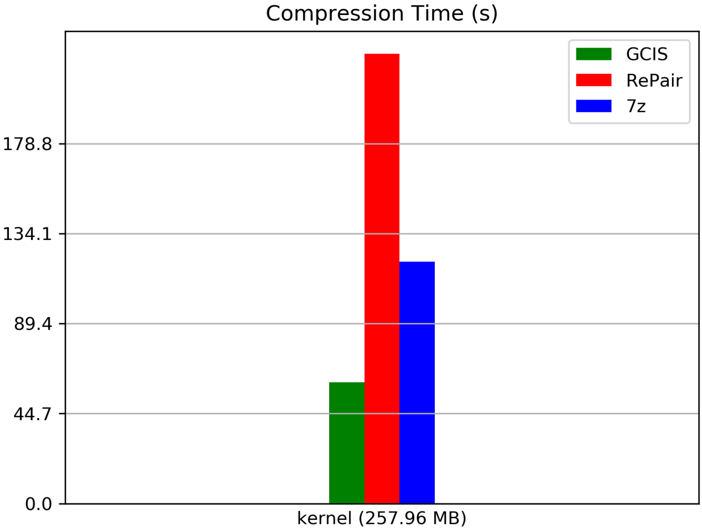
Compression Time



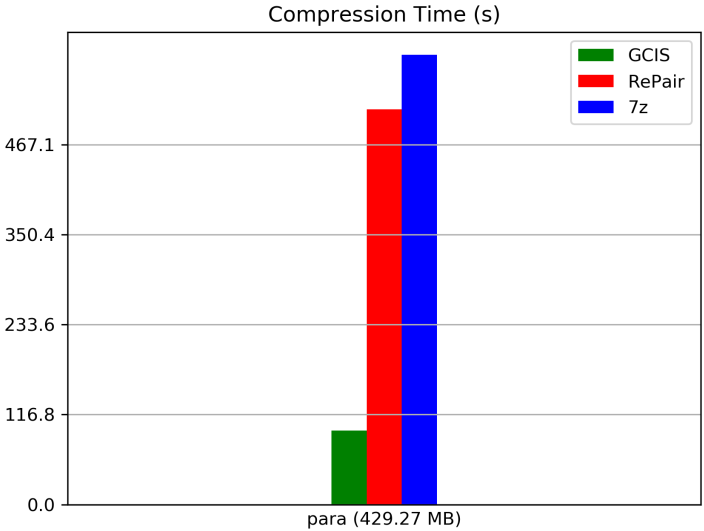
Compression Time



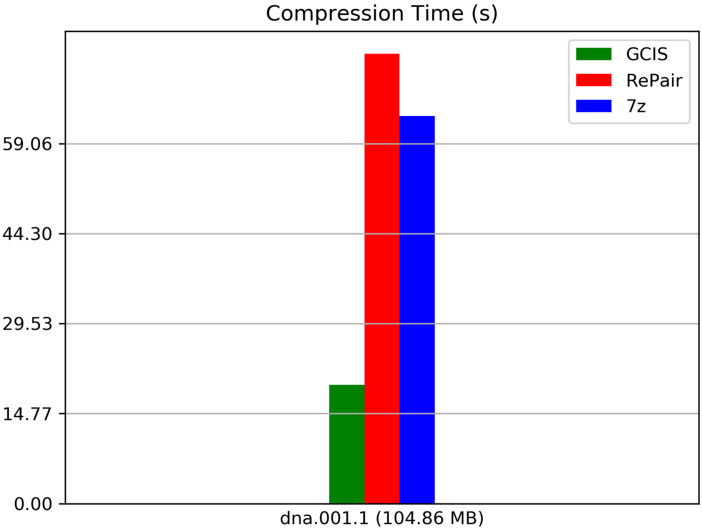
Compression Time



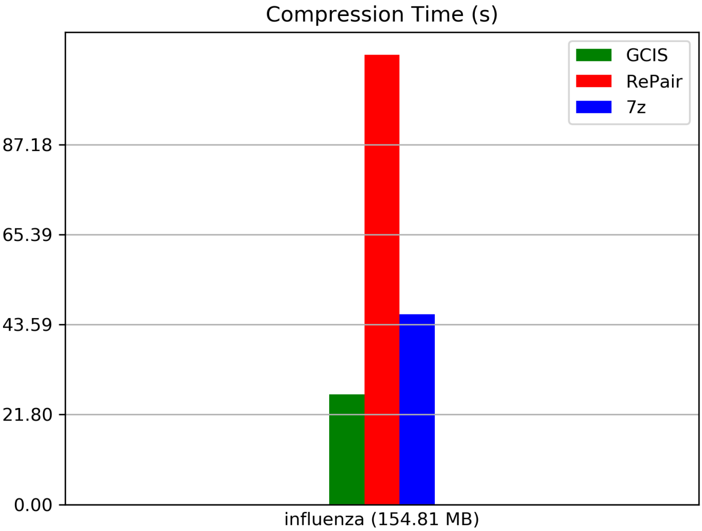
Compression Time



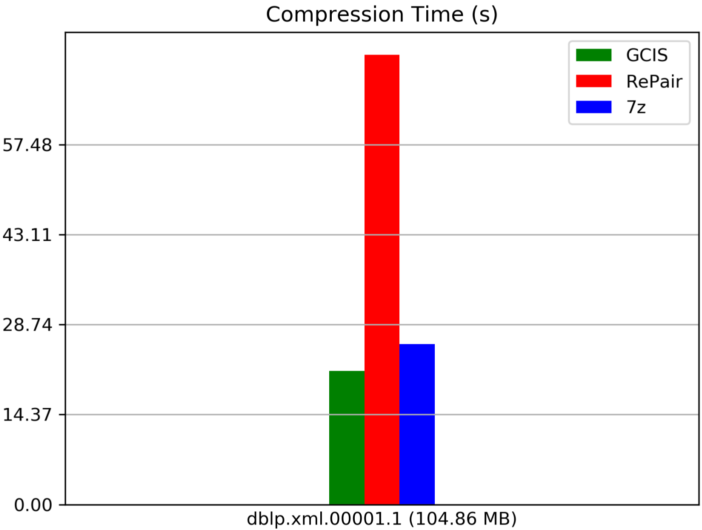
Compression Time



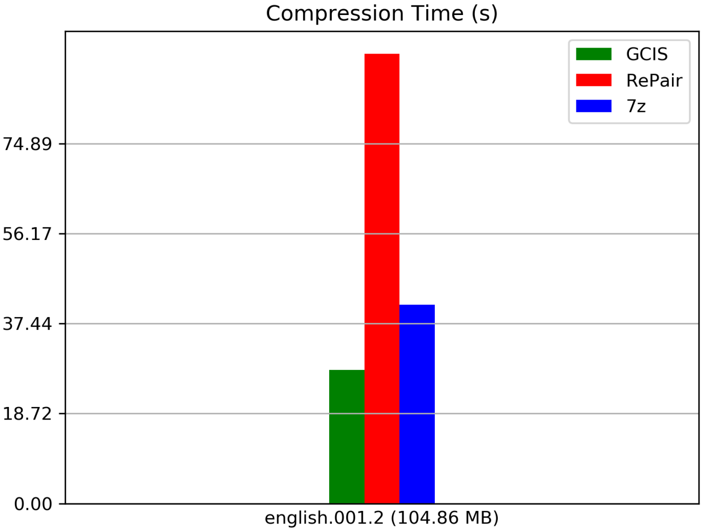
Compression Time



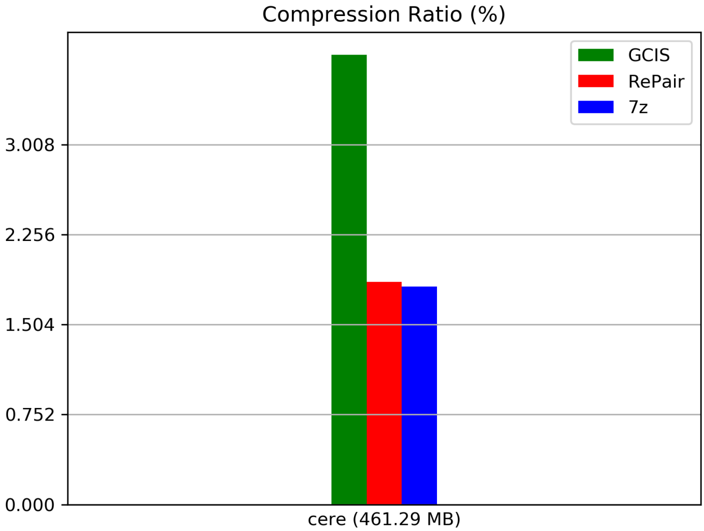
Compression Time



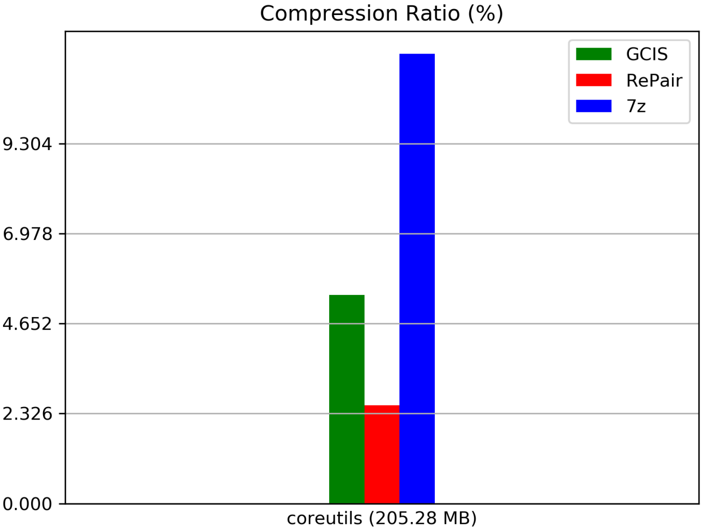
Compression Time



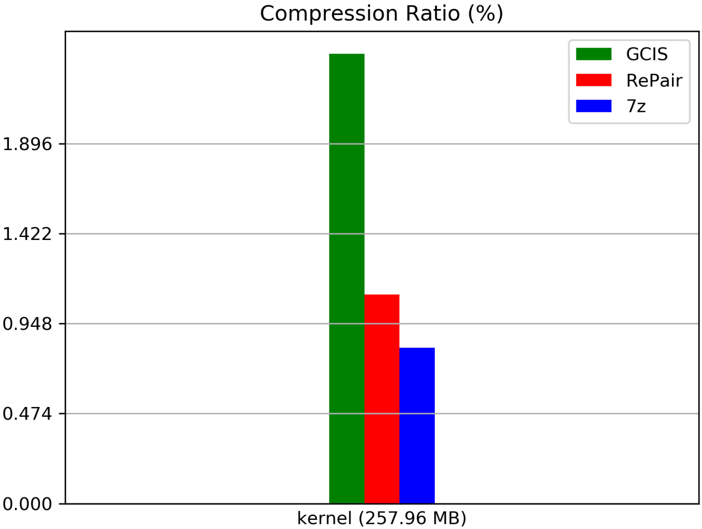
Compression Ratio



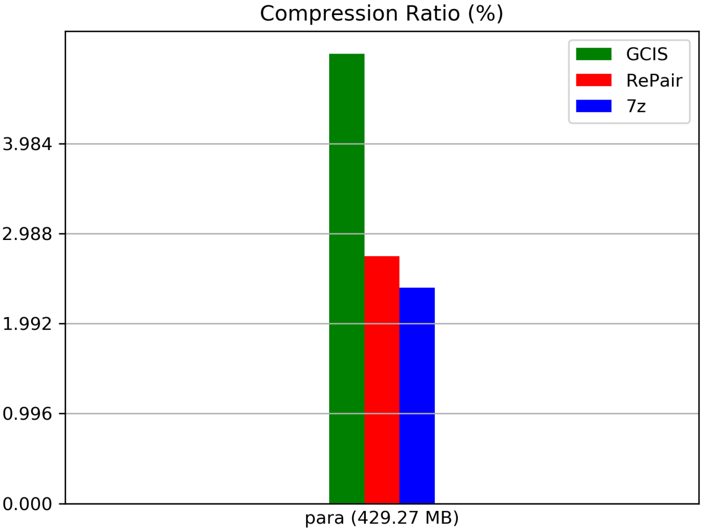
Compression Ratio



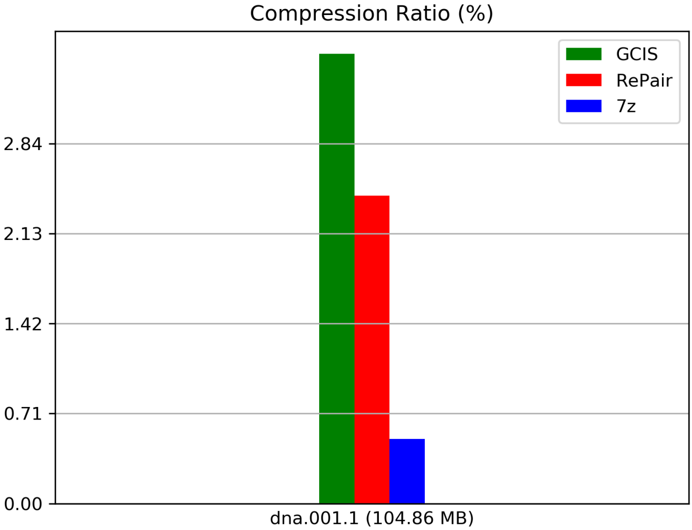
Compression Ratio



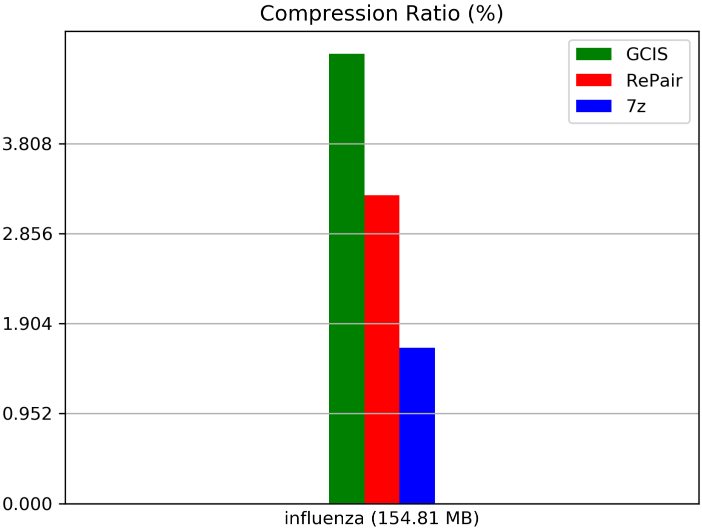
Compression Ratio



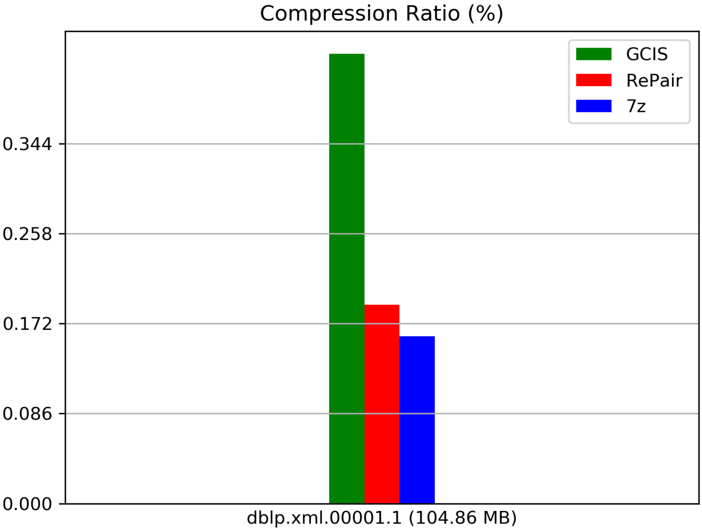
Compression Ratio



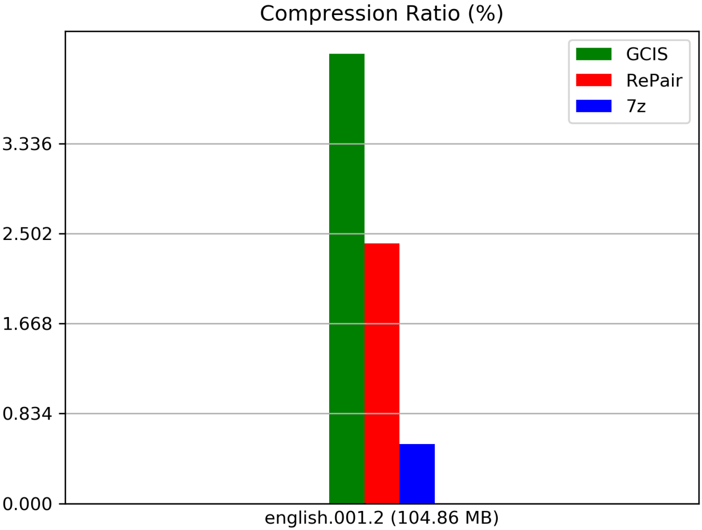
Compression Ratio



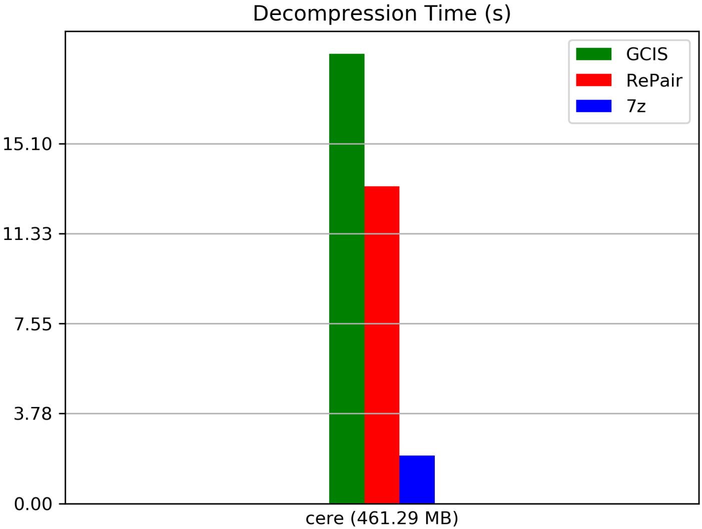
Compression Ratio



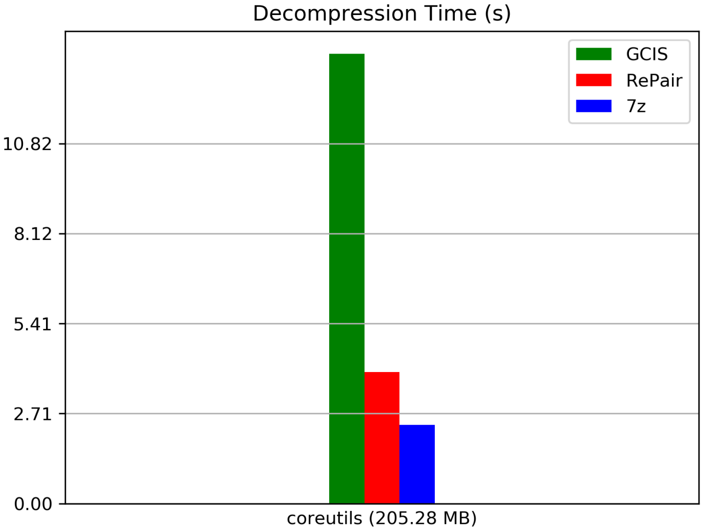
Compression Ratio



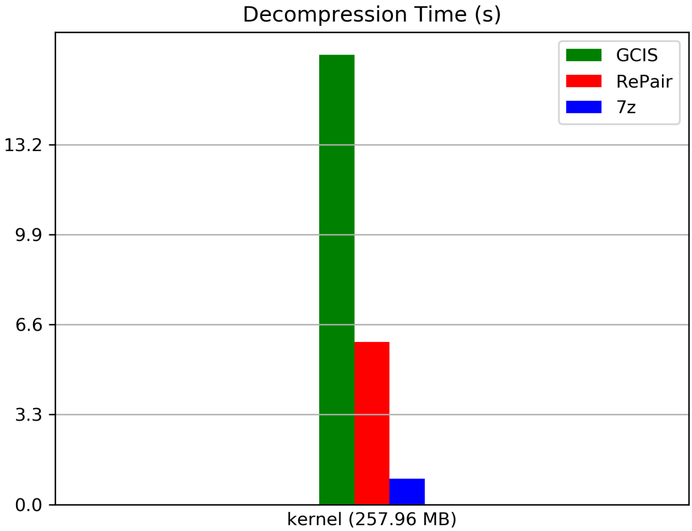
Decompression Time



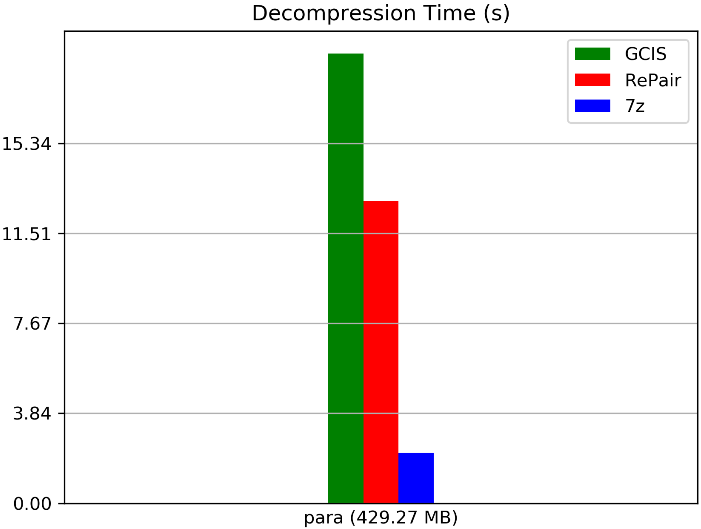
Decompression Time



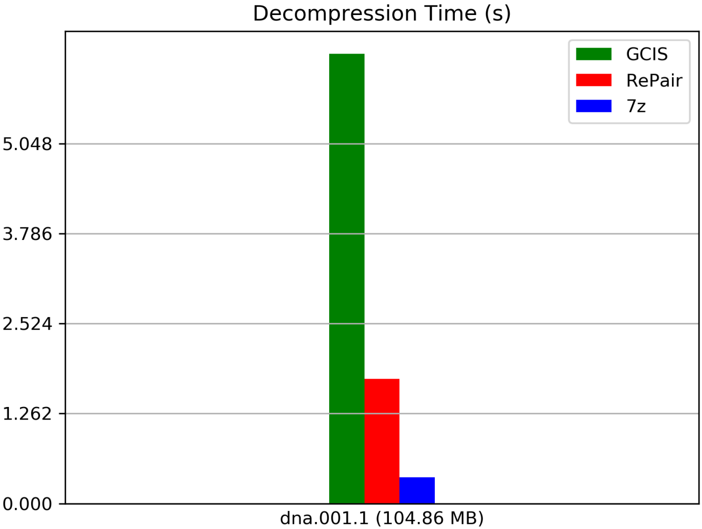
Decompression Time



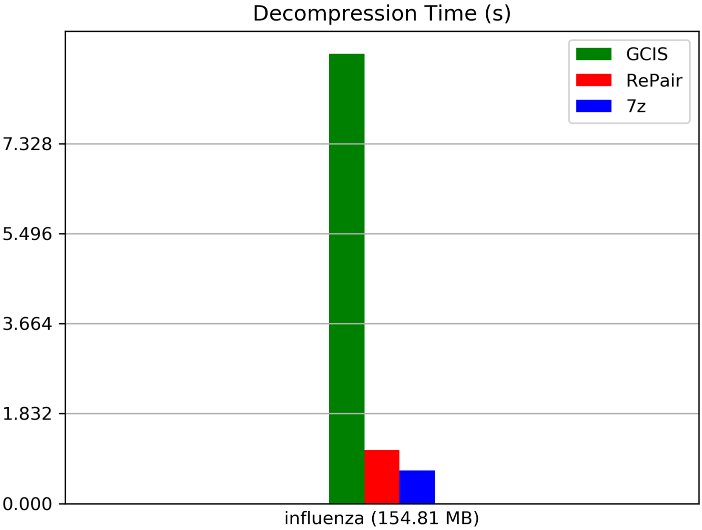
Decompression Time



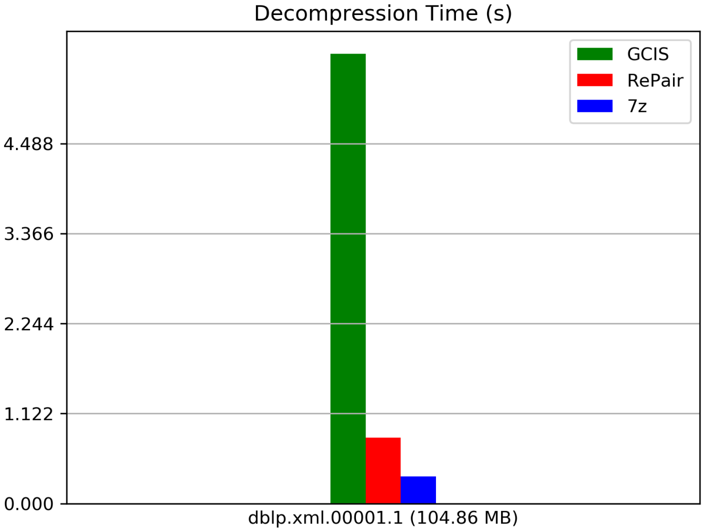
Decompression Time



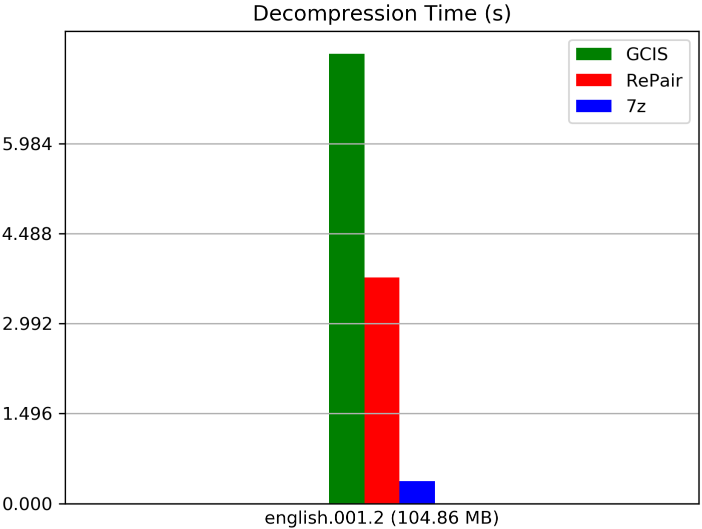
Decompression Time



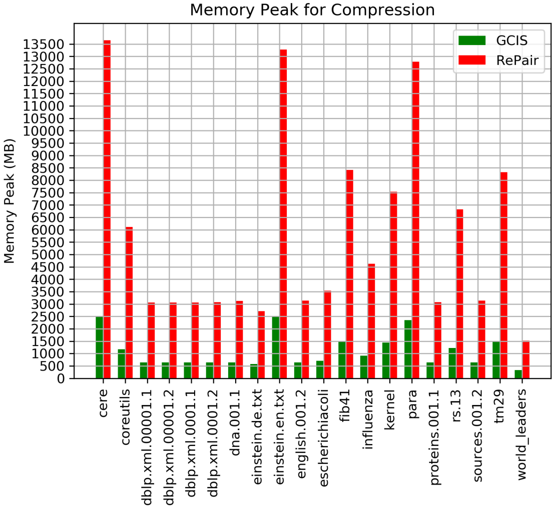
Decompression Time



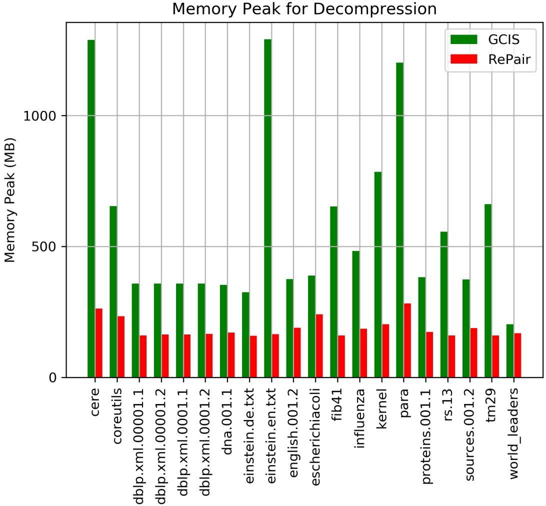
Decompression Time



Peak Memory during Compression



Peak Memory during Decompression



Summary

4 Final Considerations

Final Considerations

- GCIS showed to be a practical alternative to popular compressors due its good compression ratio and time.
- Competitive in compression regarding 7-ZIP and RE-PAIR and much faster than both.
- Slower decode times.
- Worse compression ratio, but comparable to RE-PAIR.
- Lower memory peak than RE-PAIR.

Work in Progress

- It is possible to support extraction of any substring without much space by storing the rule lenghts succinctly.
- RE-PAIR also supports EXTRACT but in a less space-efficient version which requires $(2.x$ to 3 times more space$)$.
- 7-ZIP does not support extraction.
- Develop extraction and compare with the less space-efficient version of RE-PAIR.
- We hope that GCIS will be more space-efficient than RE-PAIR when supporting the EXTRACT operation.

Thank you

Thank you!