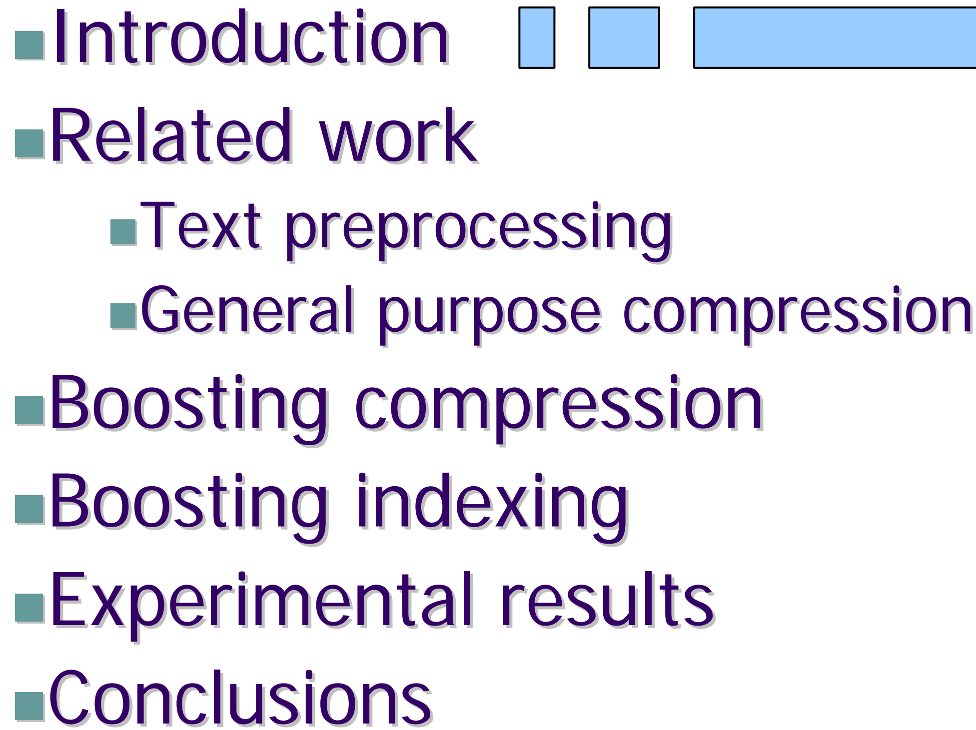
Guión de la exposición

- Motivación
- Índices invertidos
- Wavelet trees
- Wavelet trees sobre códigos densos WTDC
- Arrays de sufijos
- Arrays de sufijos comprimidos
- Otros índices y trabajo futuro
  - ETDC + [compresor | self-index]

Codificaciones orientadas a palabras
Otros Usos:: DCC'08

Fariña, A; Navarro, G. y Parama, J. ***Word-based Statistical Compressors as Natural Language Compression Boosters***. Data compression conference. Snowbird, UT. 2008.

Outline



Introduction

- We show that most of the state-of-the-art compressors (bzip2, those from the Ziv-Lempel family and the predictive PPM-based ones) **improve** their performance if:
  - They compress not the original text, but its compressed representation obtained by a word-based byte-oriented statistical compressor.

- Example:
  1. Using End-Tagged Dense Code (ETDC) as a preprocessing step,
  2. and then applying others (PPM,…)

Introduction

Original Text (T)

To be or not to be …

ETDC, TH Compressor

Byte oriented compression

Compressed Text
ETDC(T), TH(T)

…|127|200|100| 255|189|25|…

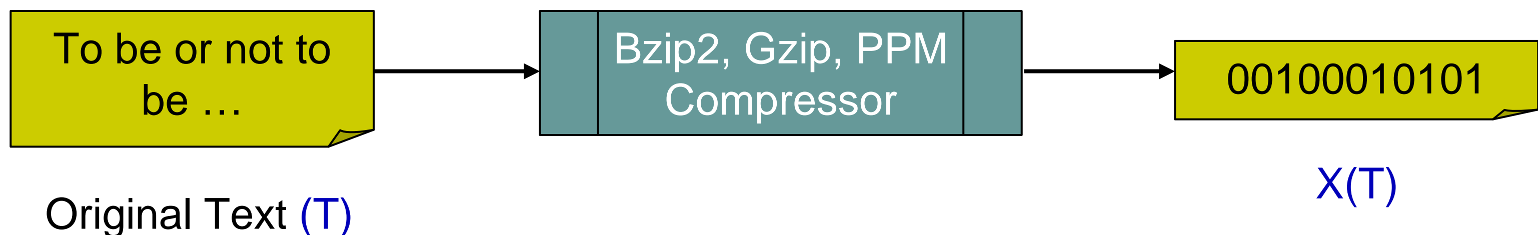
ALTERNATIVE representation of the text

Bzip2, Gzip, PPM Compressor

00100010101

Final Compressed Text
X(ETDC(T))

Better performance (Compression ratio, compression speed and decompression speed) than



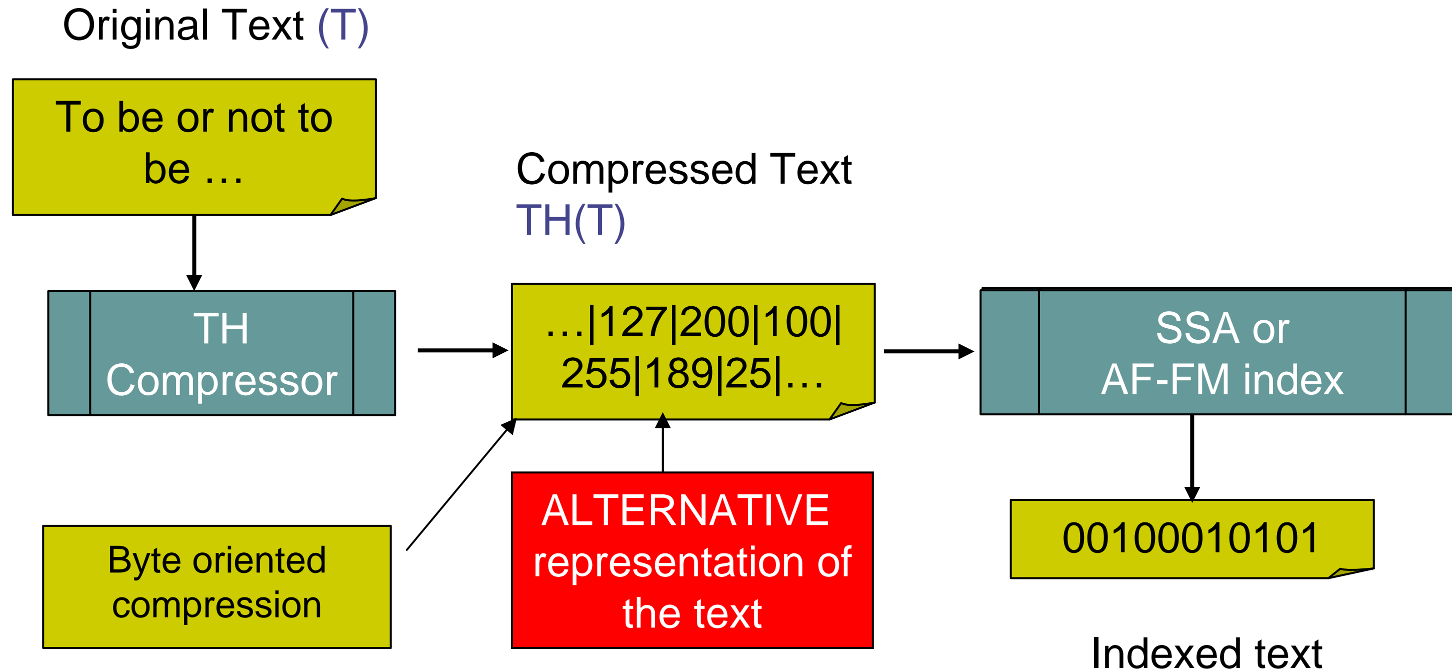
Original Text (T)

X(T)

Introduction

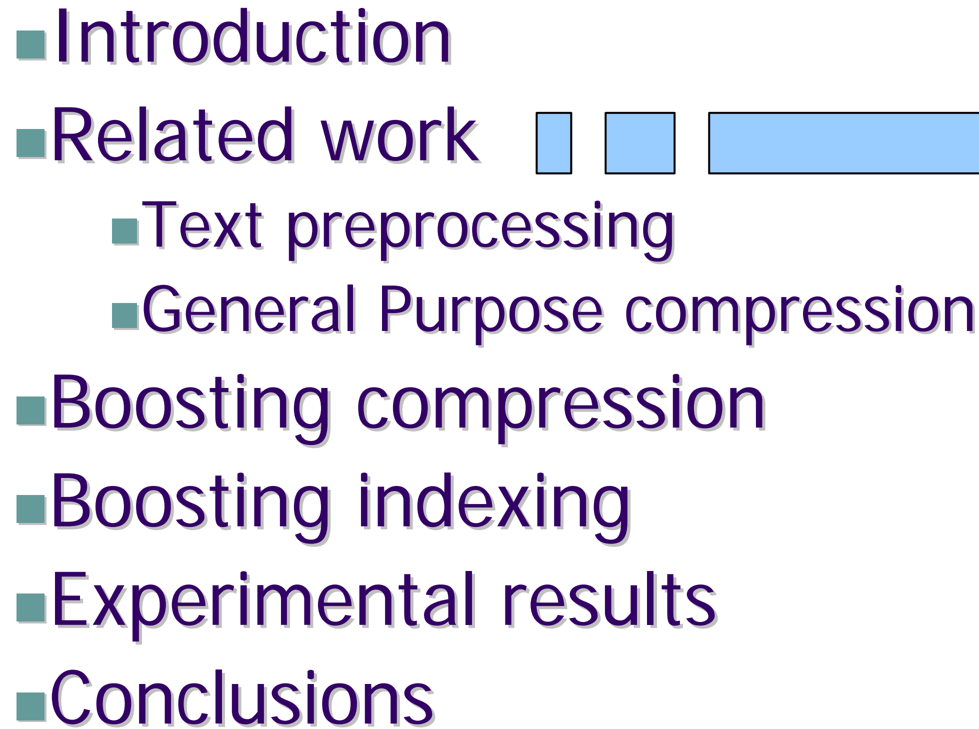
- It also improves text indexing.
  - Text compression has been recently integrated with text indexing.
  - ***Self-Indexes***: It is possible to construct an index which takes space proportional to the compressed text, replaces it, and permits fast indexed searching on it.
  - Examples:
    - Succinct Suffix Array (SSA) and
    - Alphabet-Friendly FM-index (AF-FMindex)

Introduction



A self-index on the preprocessed text is **smaller** and **faster** for searching than if applied directly on T

Outline



Related work

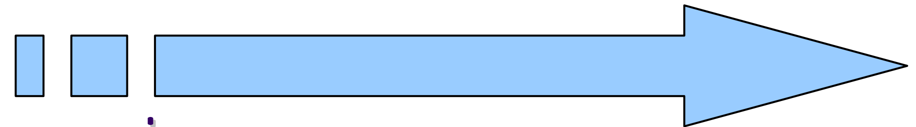
- *There exist several works based on performing some text preprocessing before applying general-purpose compressors*.
  - *Mppm* from *Adiego and de la Fuente*
    - 1st Substitutes each original word with a 2-byte id.
    - 2nd Applies PPM.
  - *Word replacing transformation* (*Skibiński, et al.,* 2005) replace original words by codewords, which index a static dictionary (in addition to other transformations) + ppm.
  - …

Related work

- There are also some works based on building a self-index over compressed text.
  - *WFM-index (Ferragina, 2006) builds a FM-index onto a text compressed with Tagged Huffman.*
  - *A simple alphabet-independent FM-Index (Grabowski, et al.*, 2006) **first** applies a Huffman-compression and **then** a Burrows-Wheeler transform over it. The resulting structure can be regarded as an **FM-index** built over a binary sequence.

Outline

- Introduction
- Related work
  - Text preprocessing
  - General Purpose compression
- Boosting compression
- Boosting indexing
- Experimental results
- Conclusions



Related work
Text preprocessing

Original Text (T)

To be or not to be …

ETDC, TH Compressor

Compressed Text ETDC(T), TH(T)…

…|127|200|100| 255|189|25|…

Bzip2, Gzip, PPM Compressor

00100010101

Final Compressed Text X(ETDC(T))

Semistatic compression

- Statistical **semistatic** compression
  - Association between source symbol ←→ codeword does not change across the text.
  - **Direct search is possible**.
  - ETDC, TH son posibles

Tagged Huffman:

•Worse compression ratio (around 35%)

*•**Suffix-free**!!!*

Outline

- Introduction
- Related work
  - Text preprocessing
  - General Purpose compression

- Boosting compression
- Boosting indexing
- Experimental results
- Conclusions

Related work
General purpose compression

Original Text (T)

To be or not to be …

ETDC, TH Compressor

Compressed Text
ETDC(T), TH(T)…

…|127|200|100|255|189|25|…

Bzip2, Gzip, PPM Compressor

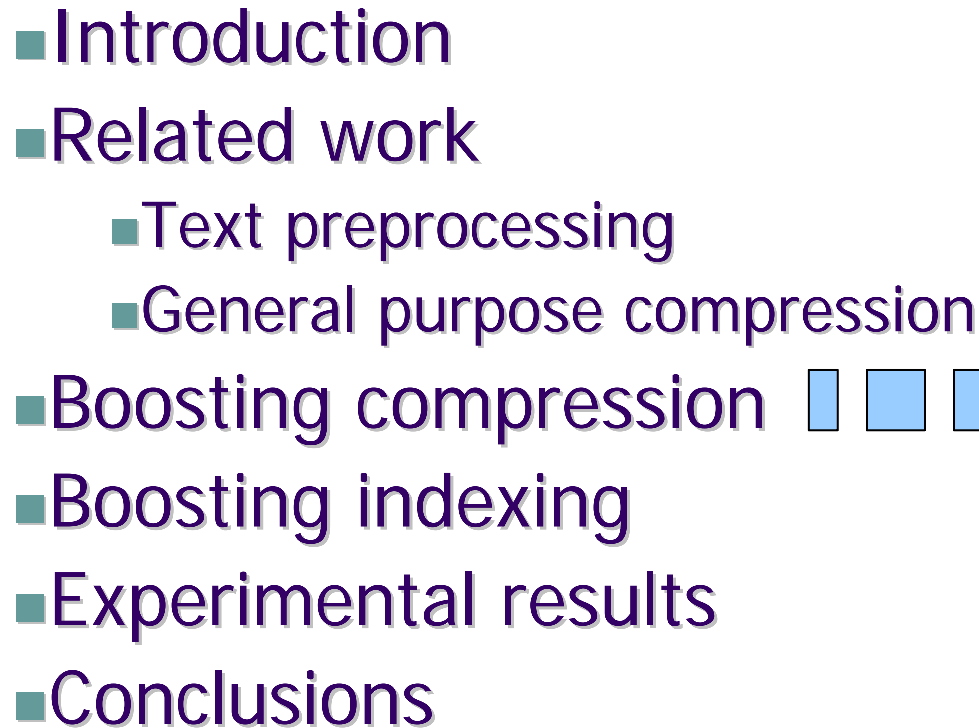
00100010101

Final Compressed Text
X(ETDC(T))

General purpose compression

- As a PPM compressor we chose ppmdi.
  - Uses a k-order modeler and a arithmetic encoder.
- Bzip2
  - Combines BWT, move-to-front, RLE, Huffman.
- Ziv-Lempel
  - Gzip

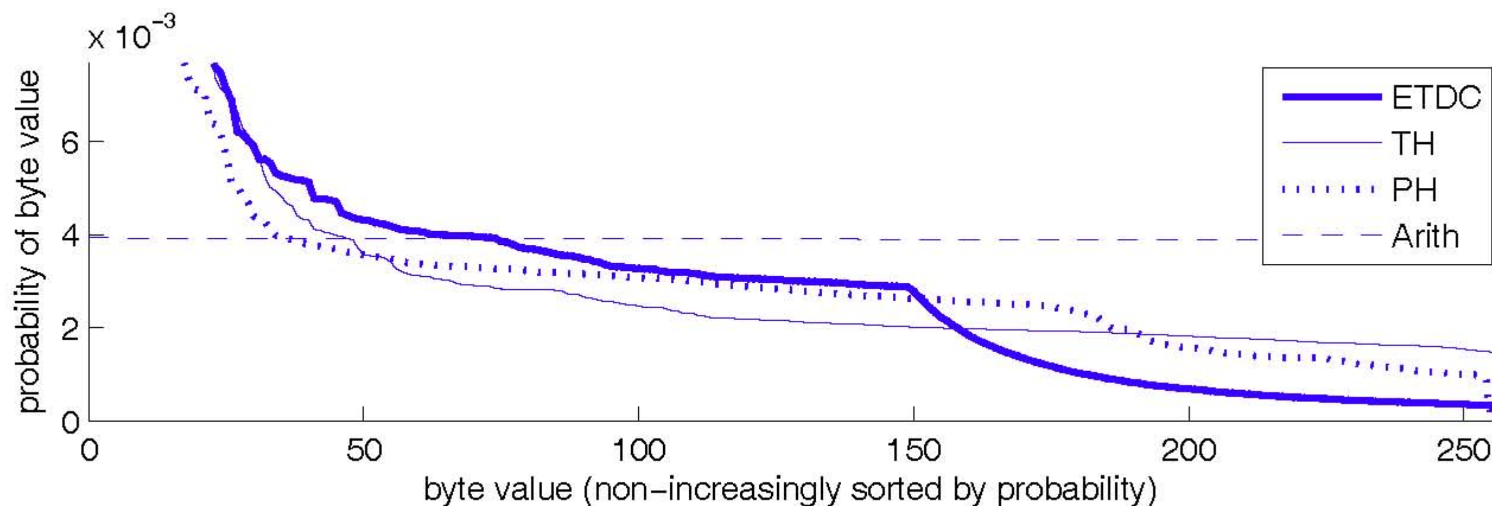
Outline



- Introduction
- Related work
  - Text preprocessing
  - General purpose compression
- Boosting compression
- Boosting indexing
- Experimental results
- Conclusions

Boosting compression

- The byte values obtained by compressing a text *T* with a word-based byte-oriented compressor shows that their frequencies are far from uniform.
- The output of a word-based **arithmetic** bit-oriented compressor displays a rather homogeneous distribution.



Boosting compression

- This idea led us to consider that the compressed file *ETDC(T)* (or *TH(T)*) was **still compressible** with a char-based bit-oriented compressor.
- However, this could not be a zero-order compressor, because the zero-order entropy (H_0) of *ETDC(T)* is too high (around 7 *bpc*).
- Instead, a deeper study of *k*-order entropy (H_k) of both *T* and *ETDC(T)* exposed some interesting properties of *ETDC*.
  - A *k*-order modeler gathers statistics of each symbol c_i by looking at the *k* symbols that precede c_i

Boosting compression

Text approx.50 Mbytes

Plain Text						***Text compressed with ETDC***					
k	***H_k***	***contexts***	***k***	***H_k***	***contexts***	***k***	***H_k***	***contexts***	***K***	***H_k***	***contexts***
0	4.888	1	8	0.972	6,345,025	0	7.137	1	8	0.132	12,531,512
1	3.591	96	9	0.837	9,312,075	1	6.190	256	9	0.099	12,854,938
2	**2.777**	**4,197**	**10**	**0.711**	**12,647,531**	2	4.642	46,027	10	0.082	13,080,690
3	2.098	51,689	11	0.595	16,133,250	3	2.601	1,853,531	11	0.072	13,252,088
4	1.668	299,677	12	0.493	19,598,218	4	1.190	6,191,411	12	0.061	13,401,719
5	**1.430**	**951,177**	13	0.406	22,900,151	5	0.566	9,396,976	13	0.056	13,531,668
6	1.264	2,133,567	33	0.025	43,852,665	6	0.308	11,107,361	49	0.001	14,939,845
7	1.118	3,931,575	50	0.011	46,075,896	7	0.187	12,015,748	50	0.001	14,946,730

A low-order modeler is usually unable to capture the correlations between consecutive characters in the text

By switching to higher-order models better statistics can be obtained, but the number of different contexts increases, consuming more space.

The average length of a word is around 5 bytes in English texts, but the variance is relatively high. In general, a high-order modeler needs to use *k* around 10 to capture the correlation between 2 consecutive words.

Boosting compression

Text approx.50 Mbytes

Plain Text						Text compressed with ETDC					
k	**H_k**	**contexts**	**k**	**H_k**	**contexts**	**k**	**H_k**	**contexts**	**K**	**H_k**	**contexts**
0	4.888	1	8	0.972	6,345,025	0	7.137	1	8	0.132	12,531,512
1	3.591	96	9	0.837	9,312,075	1	6.190	256	9	0.099	12,854,938
2	2.777	4,197	10	0.711	12,647,531	2	4.642	46,027	10	0.082	13,080,690
3	2.098	51,689	11	0.595	16,133,250	**3**	**2.601**	**1,853,531**	11	0.072	13,252,088
4	1.668	299,677	12	0.493	19,598,218	4	1.190	6,191,411	12	0.061	13,401,719
5	1.430	951,177	13	0.406	22,900,151	5	0.566	9,396,976	13	0.056	13,531,668
6	**1.264**	**2,133,567**	33	0.025	43,852,665	6	0.308	11,107,361	49	0.001	14,939,845
7	1.118	3,931,575	50	0.011	46,075,896	7	0.187	12,015,748	50	0.001	14,946,730

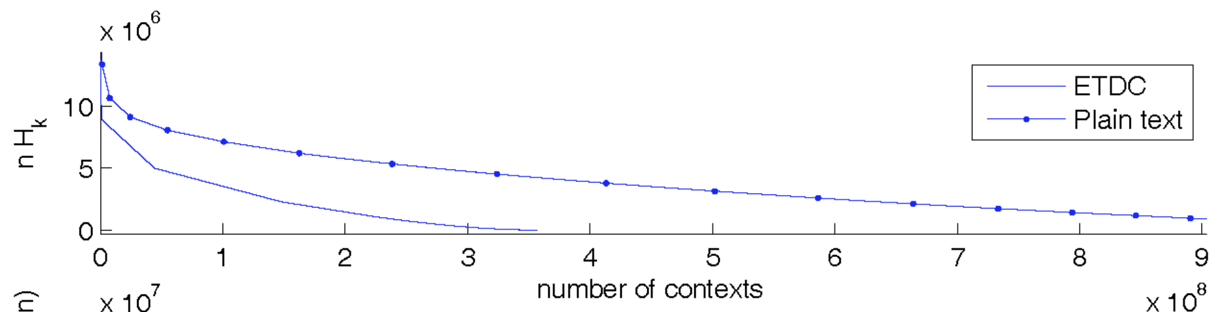
The average code length in ETDC is less than **2 bytes**, and the variance is low, as codes rarely contain more than 3 bytes.

Hence a *k-modeler* can capture correlations between consecutive words with a ***much smaller K***.

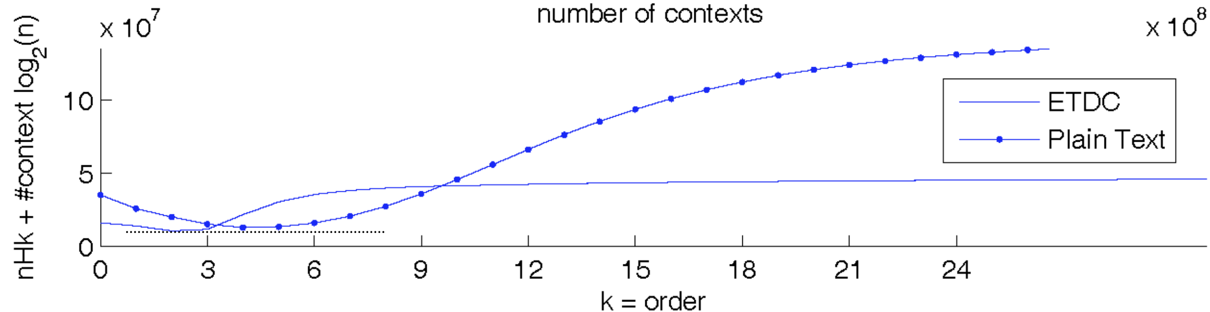
Boosting compression

- However H_k values are not directly comparable.
  - ETDC (T) has approx. 1/3 of the symbols of T.
  - Compressors do not use a fixed k, but rather administer in the best way they can a given amount of memory to store contexts.
  - The correct comparison is between the entropy achieved as a function of the number of contexts necessary to achieve it.

Size of the compressed text



Size of the compressed text + estimation on size for the contexts



Boosting compression

Laboratorio de Bases de Datos

- However H_k values are not dire[ctly comparable]
  - ETDC (T) has approx. 1/3 of the s…
  - Compressors do not use a fixed k… the best way they can a given am… conte…
  - The … as a … achie…

Although the difference is small, it is obtained with a smaller k. This permits faster and less sophisticated modelers

x 10^7

nHk + #context $\log_2(n)$

10
5
0

0 3 6 9

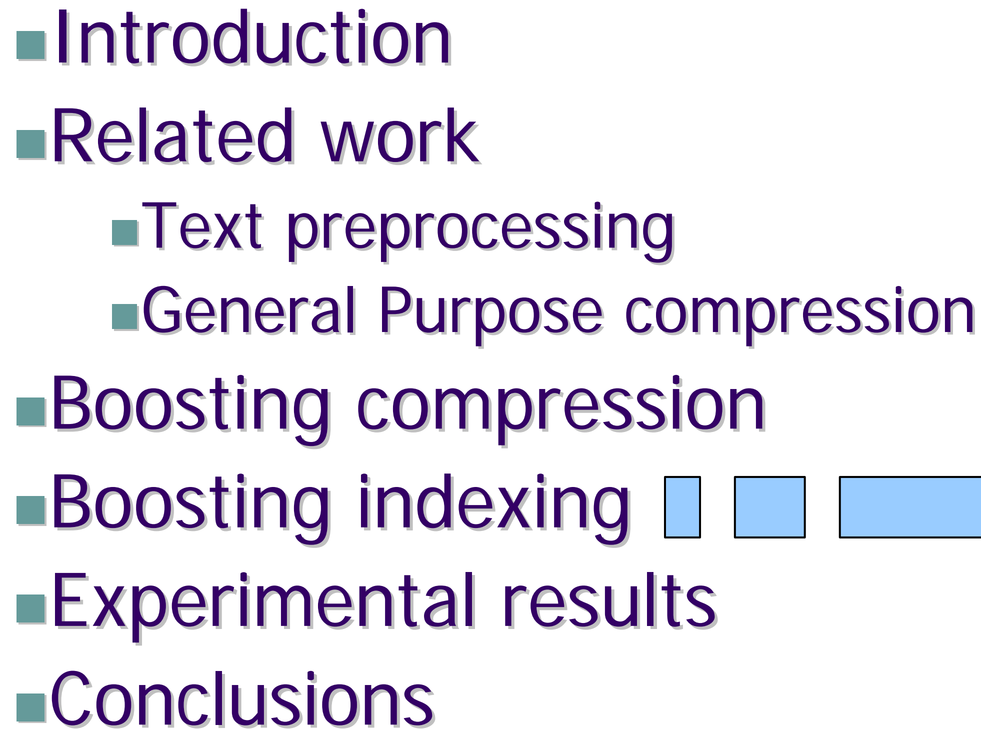
K = Order

ETDC
Plain Text

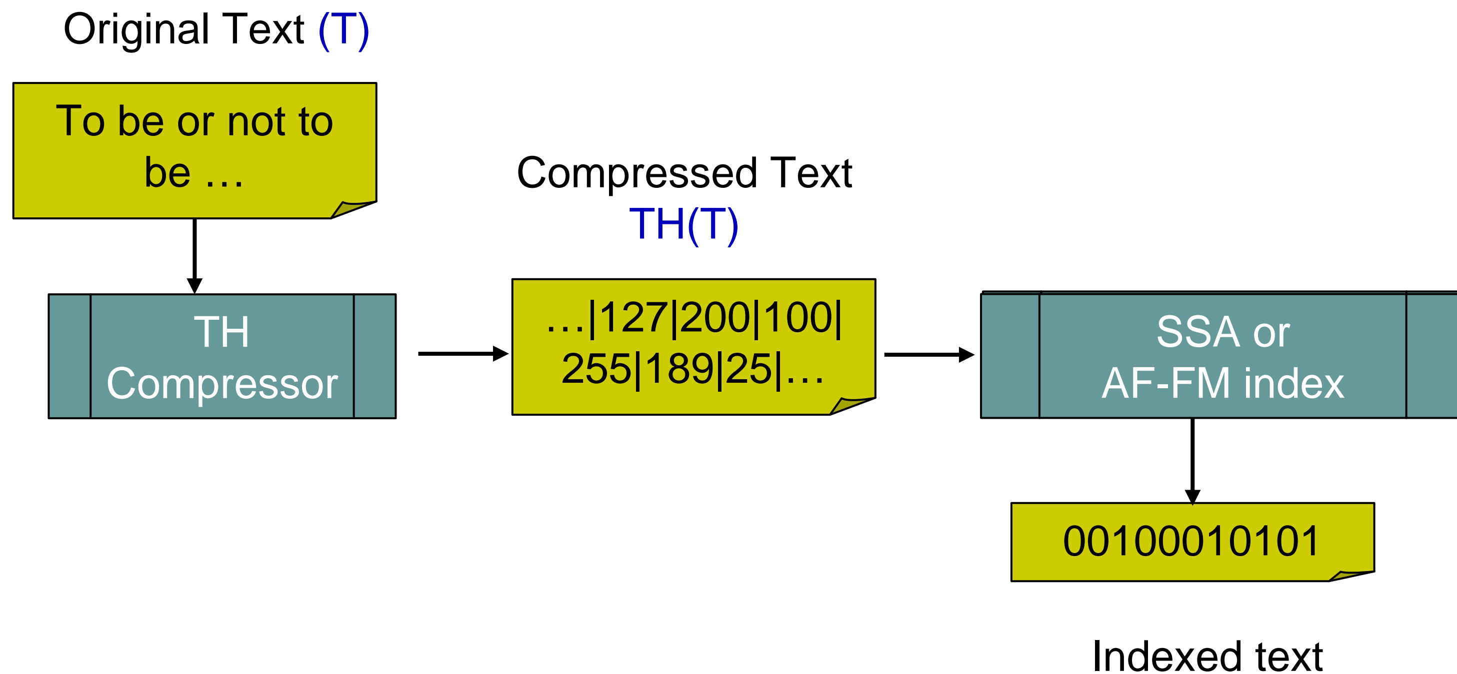
Size of the compressed text

Size of the compressed tex… estimation on s… for the contex…

Outline



Boosting indexing



Boosting indexing

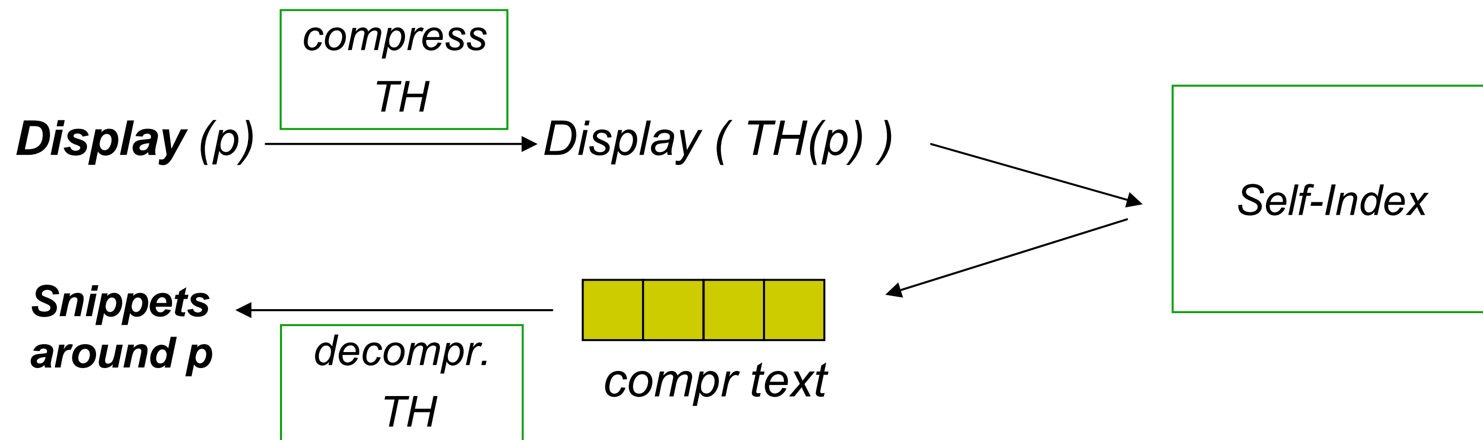
- SSA (Succinct Suffix Array)
  - V. Mäkinen & G. Navarro.
  - Obtains a size related to H_0
- AF-FMindex (Alphabet-Friendly FM-index)
  - P. Ferragina, G. Manzini, V. Mäkinen & G. Navarro
  - Compression approaches H_k.

• We expect AF-FMindex to be successful in detecting high-order correlations in *TH(T)*, where a **smaller *k*** would be sufficient to succeed compared to that built on *T.*

• Important because the AF-FMindex is **limited in practice** to obtain **entropies of relatively low *k value*.**

Boosting indexing

- 'Self-indexes' are able to:
  - *Count* the number of ocurrences of a pattern *p* in *O(|p|)* steps.
  - *Locate* the position of a suffix in the text.
  - Recover the original text (display / extract).



TH generates *suffix-free codes* ➔ no false matchings occur

Boosting indexing

- 'Self-indexes' are able to:
  - *Count* the number of ocurrences of a pattern *p* in *O(|p|)* steps.
  - *Locate* the position of a suffix in the text.
  - Recover the original text.

- We chose *TH* as the base compressor because it generates *suffix-free codes*.
  - This permit to compress the searched pattern *p* and then search for its compressed form directly.
  - As those self indexes use a **terminator** ($) for the indexed text, we modified *TH* to ensure that at least 1 byte value does not appear in the compressed text.

Outline

- Introduction
- Related work
  - Text preprocessing
  - General Purpose compression
- Boosting compression
- Boosting indexing
- Experimental results
- Conclusions

Experimental results

CORPUS	size (bytes)	Num. words	Nº different words
CALGARY	2,131,045	528,611	30,995
FT91	14,749,355	3,135,383	75,681
CR	51,085,545	10,230,907	117,713
FT92	175,449,235	36,803,204	284,892
ZIFF	185,220,215	40,866,492	237,622
FT93	197,586,294	42,063,804	291,427
FT94	203,783,923	43,335,126	295,018
AP	250,714,271	53,349,620	269,141
ALL FT	591,568,807	124,971,944	577,352
ALL	1,080,719,883	229,596,845	886,190

- Intel Pentium-IV 3 Ghz 4Gb RAM.
  - Debian GNU/Linux (kernel 2.4.27)
  - gcc 3.3.5 and optimization –O9
  - Time measures **CPU user-time**

Experimental results

Compression ratio

Compression ratio Corpus ALL

Compressor	Compression ratio
ETDC	33,66%
gzip	35,09%
bzip2	25,98%
ppmdi -6	24,21%
ppmdi -9	22,34%
mppm -9	20,83%
ETDC+gzip	24,35%
ETDC+bzip2	22,18%
ETDC+ppmdi -6	20,90%
ETDC+ppmdi -9	19,98%

Y axis: 0,00% – 40,00%

X

ETDC+X

Experimental results

Compression time

Compression time Corpus ALL

	Compressor	Time
	ETDC	75,03
X	gzip	157,91
X	bzip2	336,81
X	ppmdi -6	599,15
X	ppmdi -9	636,57
	mppm -9	1182,47
ETDC+X	ETDC+gzip	118,11
ETDC+X	ETDC+bzip2	216,17
ETDC+X	ETDC+ppmdi -6	448,11
ETDC+X	ETDC+ppmdi -9	470,01

In seconds

Experimental results

Decompression time

Decompression time Corpus ALL-FT

Group	Method	Time
	ETDC	7,59
X	gzip	11,47
X	bzip2	84,53
X	ppmdi -6	344,01
X	ppmdi -9	345,05
	mppm -9	345,82
ETDC+X	ETDC+gzip	11,25
ETDC+X	ETDC+bzip2	48,55
ETDC+X	ETDC+ppmdi -6	217,33
ETDC+X	ETDC+ppmdi -9	222,05

Y axis: 0,00 – 50,00 – 100,00 – 150,00 – 200,00 – 250,00 – 300,00 – 350,00 – 400,00

In seconds

Experimental results

Compression ratio

- Comparison against other ppm-based algorithms that use a high value of k.

- ppmd (k=16)

ppmd	ETDC+ppmd
17.89%	*17.00%*

- Monstruous ppm (ppmd var J) (k=128)

ppm-monst	ETDC + ppm-monst
15.76%	*15.83%*

All the co-occurrences of the symbols in the text have been detected.

Experimental results

Indexing

- We used the corpus CR (aprox 50 Mbytes)

Size of index: Compression ratio (%)

Rank Factor 16	Sample Rate			
	16	32	64	1024
TH + affm	49,95%	41,84%	37,78%	34,73%
Plain + affm	104,83%	79,83%	67,33%	57,96%
TH + ssa	47,94%	43,88%	41,86%	40,33%
Plain + ssa	111,69%	99,19%	92,94%	88,25%
TH	34,31%			

Experimental results
Indexing

- By setting SR=1024 and RF=64…
  - Less space *(but slower indexes at searches)*
    - the AF-FMindex occupies less than the text compressed with TH

Compression ratio	
TH+affm	32.71%
Plain+affm	53.59%
TH+ssa	38.37%
Plain+ssa	83.62%
TH	34.31%

Experimental results

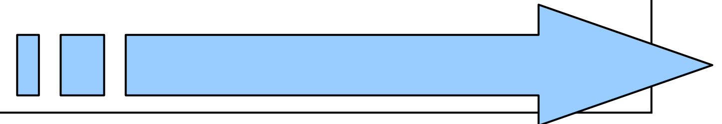
Indexing

- For each SR value there is a line depending on the RF values
- We measured time in ms (for **locate**).

corpus CR (aprox 50 Mbytes)

Outline

- Introduction
- Related work
  - Text preprocessing
  - General Purpose compression
- Boosting compression
- Boosting indexing
- Experimental results
- Conclusions



Conclusions

- By preprocessing a text T with either ETDC or TH:
  - We obtain a compressed text of around 30% of size(T).
  - Still compressible and indexable.
- By compressing in a second step with PPM, gzip or bzip2, we improve: compression ratio, compression speed, and decompression speed.
  - **ETDC+gzip**: very fast and good compression ratio (<bzip2)
  - ETDC+bzip2 compresses a little bit more, at the expense of a lower speed.
  - **ETDC+PPM**: the best compression (but still slow).

Conclusions

- If we apply the AF-FMindex and SSA over text compressed with TH and:
  - we set the **index** parameters in order to obtain a structure of the **same size** as if we indexed the plain text, we obtain two indexes that are **much faster** than the traditional ones.
  - If we instead set the parameters to obtain two indexes with the **same search speed**, the index over the compressed text will **occupy around 30% less** than in the case of the plain text.