Data Compression Conference 2010

Snowbird, UT. March 25, 2010

A new searchable variable-to-variable compressor

Nieves R. Brisaboa

Antonio Fariña

Juan R. López

Eduardo Rodríguez

Gonzalo Navarro

Outline

- Introduction
  - Variable-to-variable compression
- Our Approach (v2vdc)
  - General structure
    - LCP-based phrase generation & compression
  - Searching
- Experimental results
- Conclusions

Introduction

Fariña et al, DCC 2010

Text compression

- *My huge disk is cheap… Why compression?*
- *Compression* ***reduces not only space****!*
  - *Disk access time (your huge disk is slow)*
  - *Transmission time (networks are even slower)*
  - *Search time (less data to process)*

- *Compression has been integrated into Text Retrieval Systems, improving their performance in all aspects*

Introduction

Fariña et al, DCC 2010

Typical classification of compressors

- *Typically a compressor…*
  - *Splits the source text into source symbols. Each symbol composed of (1+) elements from a **source alphabet** (chars, words, etc).*
  - *Assigns each symbol a codeword, build over a **target alphabet** (bits, bytes, …)*

input \ output	fixed	variable
fixed	--	statistical
variable	Dictionary-based	Variable-2-variable

- *Lempel-Ziv: char-based input (1 char, 2 chars…), fixed output (pointer)*
- *Huffman: (char-based / word-based),variable length codes.*

Introduction

Fariña et al, DCC 2010

Typical classification of compressors

- *Variable-to-fixed compressors*
  - *Compression is achieved by compressing large substrings of the source text with fixed length codewords*

- *Fixed-to-variable compressors (statistical)*
  - *They model the text being compressed so that a source symbol (a char, a word) is assigned a variable-length codeword.*
  - *More frequent symbols ➔ shorter codewords*
    - *Ex: Huffman, dense codes,…*

Introduction

Fariña et al, DCC 2010

Statistical word-based Semistatic compression

- *Importance of **semistatic** compression in Text Databases*
  - ***Two-passes** compressors*
  - *Word-based && byte-oriented*
  - *Reduces their size: around 30% compression ratio*
  - *It is possible to search the compressed text directly*
    - *Searches are up to 8 times faster (online searches)*
  - *Permits direct access and local decompression*
  - *Decompression is only needed for presenting results*
  - *Fast compression (two-passes)*
  - *even faster decompression!!*

- *Ex: Tagged Huffman, **ETDC**, SCDC, RPBC,…*

Outline

- Introduction
  - Variable-to-variable compression
- Our Approach (v2vdc)
  - General structure
    - LCP-based phrase generation & compression
  - Searching
- Experimental results
- Conclusions

Introduction

Fariña et al, DCC 2010

Motivation: Variable-to-variable compression

- *Changing from char-based to word-based compressors*
  - *Improved compression from 60% (classic Huffman) to around 25-30% (word-based bit-oriented Huffman)*
- *Changing from bit-oriented to byte-oriented codewords.*
  - *Improved decoding speed and searches, but…*
  - *Plain Huffman (the optimal code) compresses "only" around 30%*
- *Can a word-based byte-oriented var2var compressor…??*
  - *Improve the compression ratios of Plain Huffman, SCDC ???*
  - *Permit online searching without decompression*
  - *Allow random access and local decompression*

… LET'S COMPRESS PHRASES, NOT ONLY WORDS !!!

Introduction

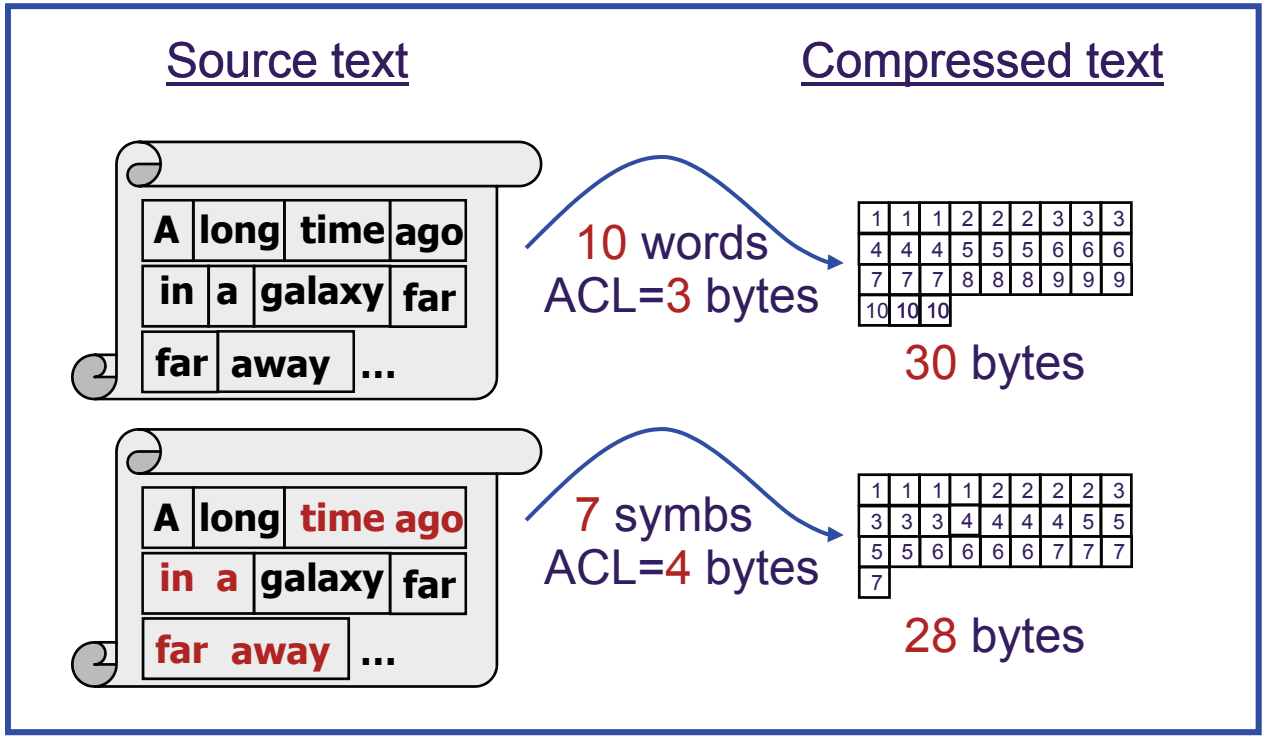
Fariña et al, DCC 2010

Variable-to-variable compression

Example

Input:
Words and pairs

Output:
bytes



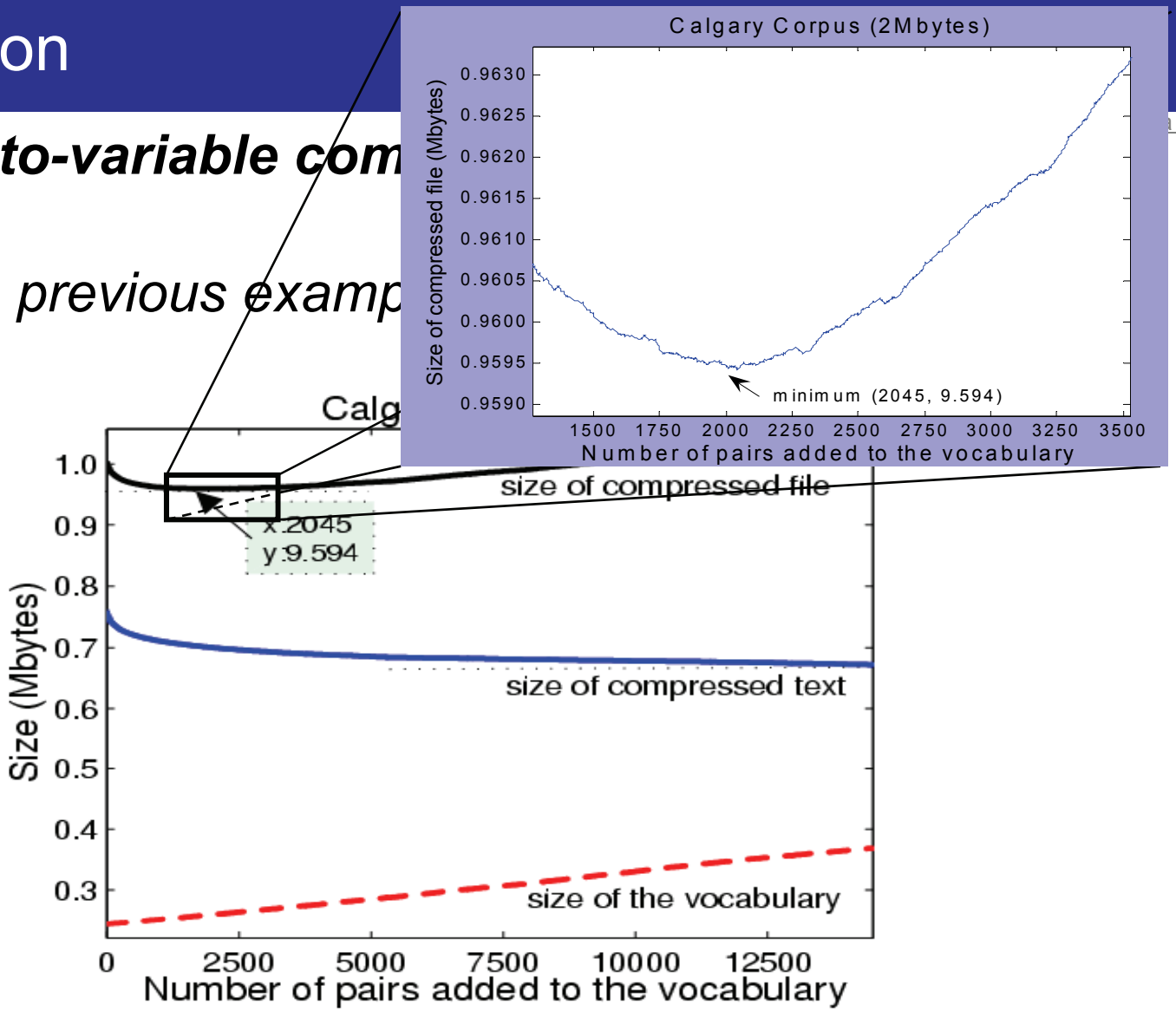
- *Using 1 pair implies that less symbols need to be compressed…*
- *… but the vocabulary size increases*

Introduction

Variable-to-variable com

et al, DCC 2010

- *In the previous examp*



- *Tradeoff "size of compressed text" Vs "size of vocabulary"*

Introduction

Fariña et al, DCC 2010

Using Phrases: Problems to face

- *Choosing good phrases is crucial*
  - *Using some phrases ➔ forbids choosing "subphrases" or "overlapping" ones*

    *ABBCABAACAB ➔ {A; **BBC**; AB; AAC; AB}*

  - *The number of occs of the words in a phrase "decreases" ➔ a longer codeword could be given to them*

    ABBCABAACAB… ➔ {A; BBC; AB; AAC; AB}

- *Need for a compact representation of the phrases*
  - *Storing the phrase VS gain in compression*

Outline

- Introduction
  - Variable-to-variable compression
- Our Approach (v2vdc)
  - General structure
    - LCP-based phrase generation & compression
  - Searching
- Experimental results
- Conclusions

Our Approach (v2vdc)

Fariña et al, DCC 2010

General structure: 2 main phases

- Phrase-based modeler + statistical coder
  - 1st phase:
    - parsing and phrase selection
    - Encoding (ETDC)
  - 2nd phase:
    - Compression

Our Approach (v2vdc)

Fariña et al, DCC 2010

LCP-based phrase generation and compression

- 1st phase: parsing and phrase selection
  - Parsing the text with a word-oriented spaceless model. Each word is given an integer ID, and Sids[] (array of IDs) is set.
  - Creating an Int-Suffix Array over Sids[ ] ➔ LCP
  - Detecting phrases and their frequency (using LCP): Threshold
  - Choosing "good" phrases. Cost function
    - Marking the used phrases from Sids[ ], so that their words are compressed inside the phrase, instead of independently
    - Adding those phrases to the vocabulary of the compressor.
      - » The number of occs of its words is decreased (n_i=n_i - freq phrase)
    - Heuristic
  - Sorting the vocabulary (words+phrases) by frequency
  - Encoding: each symbol is given a codeword

Our Approach (v2vdc)

Fariña et al, DCC 2010

- Heurística para decidir si un par $\alpha\beta$ debe ser añadido …
  - Tamaño texto comprimido antes de añadir par $\alpha\beta$
  - Tamaño texto comprimido después de añadir par $\alpha\beta$
  - Incremento tamaño vocabulario para incluir $\alpha\beta$

$$skip_{bytes} = f_\alpha * |C_\alpha| + f_\beta * |C_\beta|$$

$$add_{bytes} = f_{\alpha\beta} * |C_{\alpha\beta}| + (f_\alpha - f_{\alpha\beta}) * |C'_\alpha| + (f_\beta - f_{\alpha\beta}) * |C'_\beta| + K$$

Our Approach (v2vdc)

Fariña et al, DCC 2010

LCP-based phrase generation and compression

- 2nd phase:
  - Compression
    - Replaces any word/phrase by its codeword.
      - » X^{th} word ➔ Codeword $\mathbf{C_x}$
      - » Y^{th} phrase ABC ➔

        -1st ocurrence is encoded as $\mathbf{C_A,C_B,C_C}$ (and offset is kept)

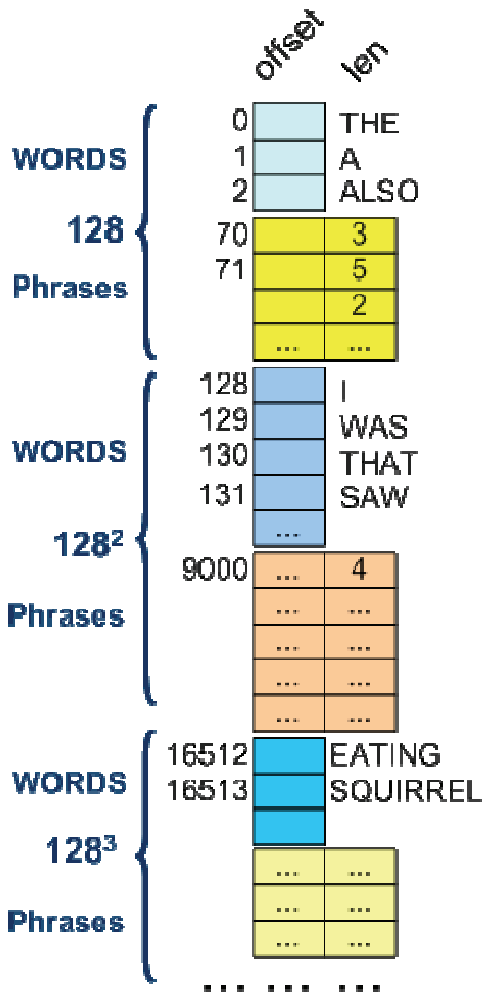
        - Rest of occurrences ➔ Codeword $\mathbf{C_y}$
  - Saving the vocabulary: (sorted by frequency)
    - All the words
    - For each phrase:
      - » The number of words it contains
      - » The offset in the compressed text of its first occurrence

Our Approach (v2vdc)

Fariña et al, DCC 2010

Data structures for compression

I SAW A SQUIRREL [...] THAT SQUIRREL WAS EATING [...] THAT SQUIRREL WAS ALSO [...]

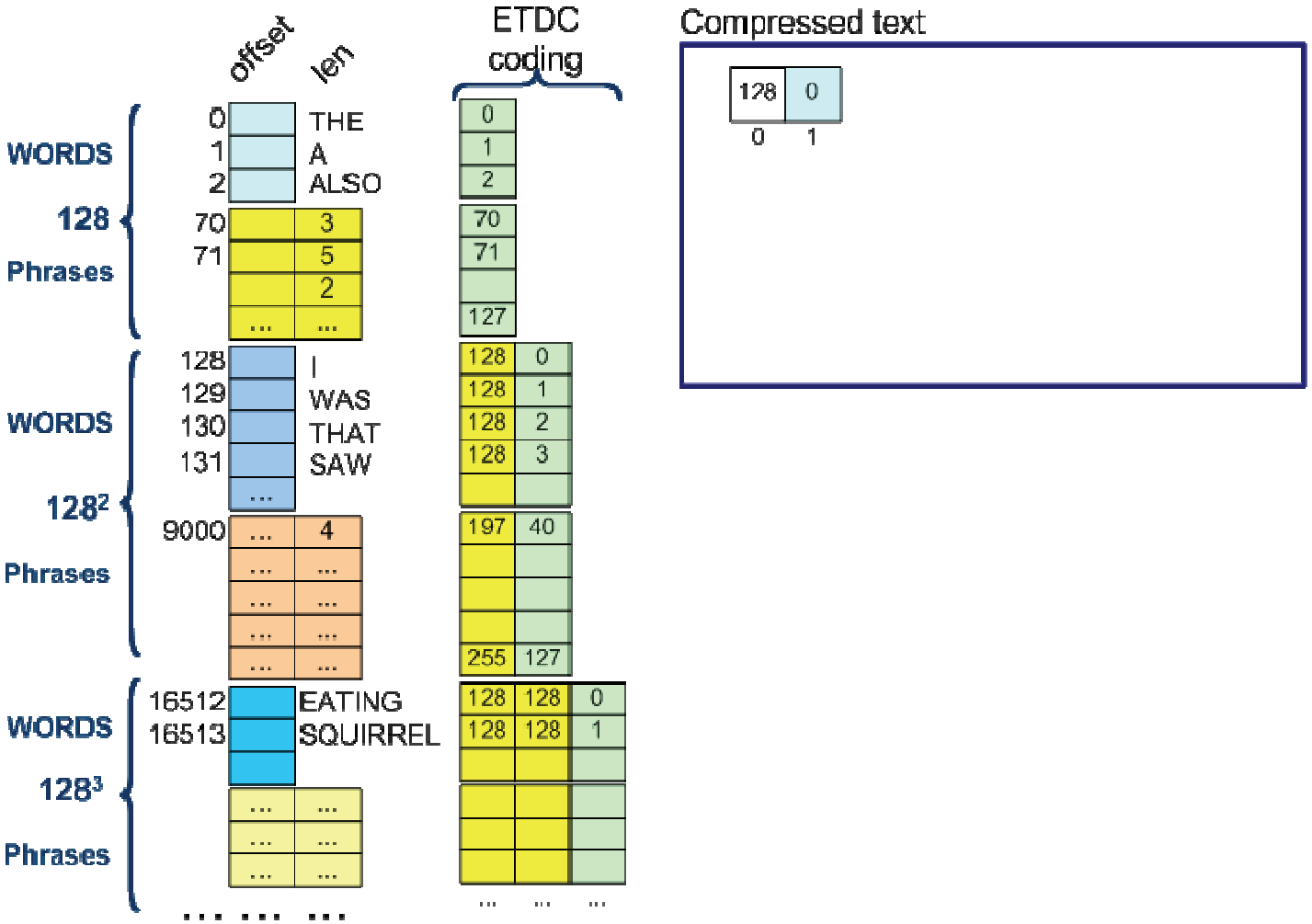


Our Approach (v2vdc)

Fariña et al, DCC 2010

Data structures for compression

I SAW A SQUIRREL [...] THAT SQUIRREL WAS EATING [...] THAT SQUIRREL WAS ALSO [...]

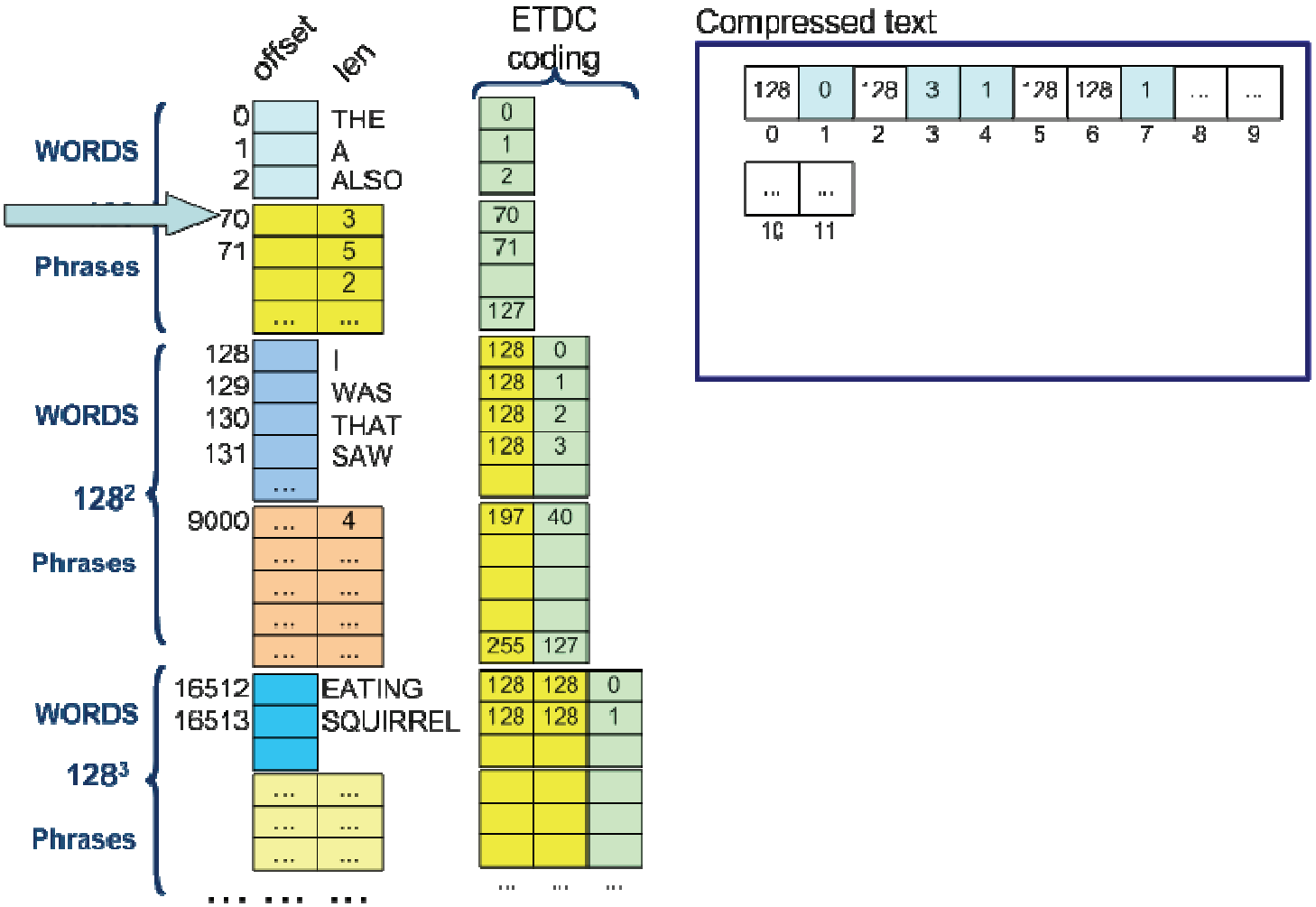


Our Approach (v2vdc)

Fariña et al, DCC 2010

Data structures for compression

I SAW A SQUIRREL [...] THAT SQUIRREL WAS EATING [...] THAT SQUIRREL WAS ALSO [...]

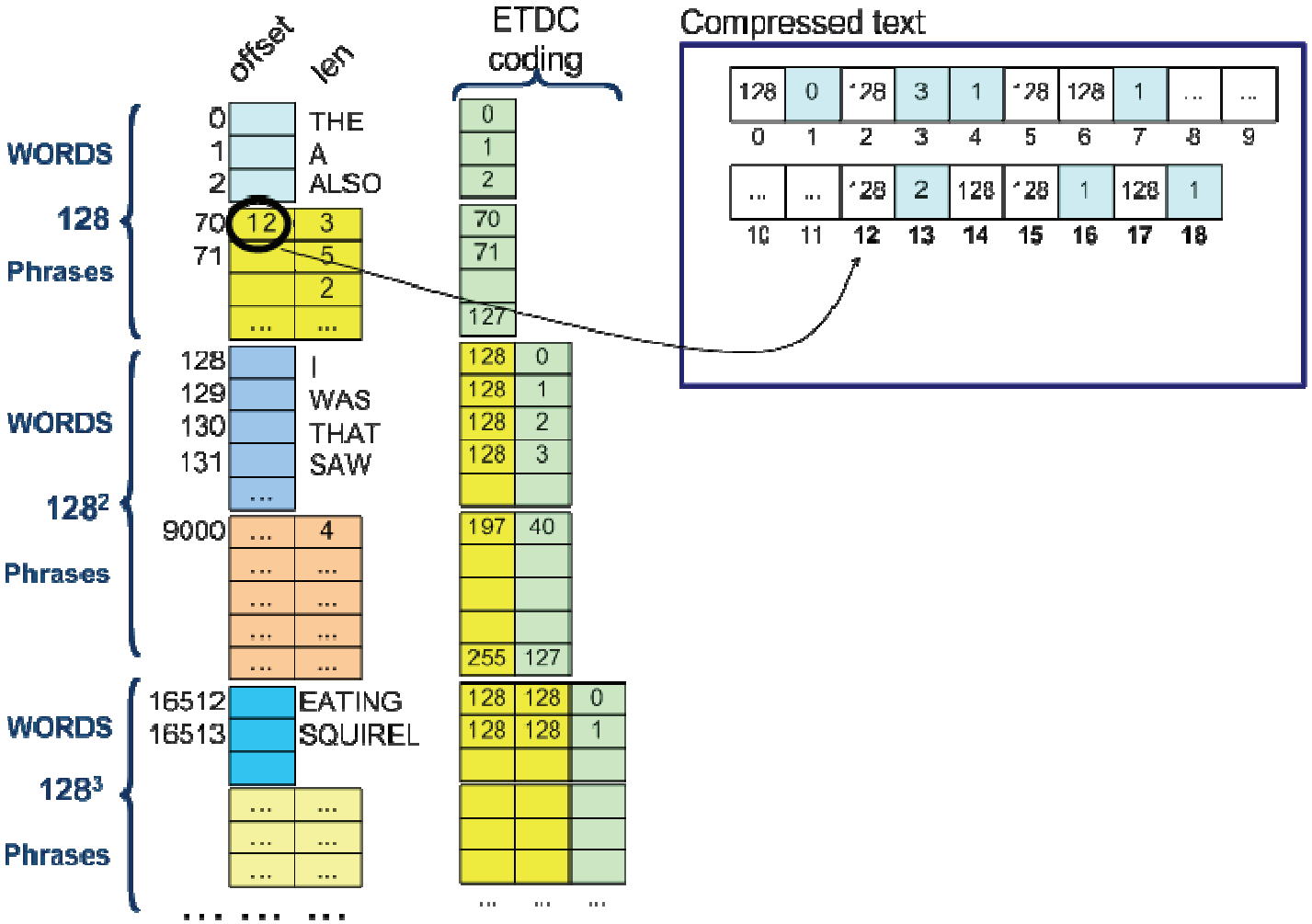


Our Approach (v2vdc)

Fariña et al, DCC 2010

Data structures for compression

I SAW A SQUIRREL [...] THAT SQUIRREL WAS EATING [...] THAT SQUIRREL WAS ALSO [...]

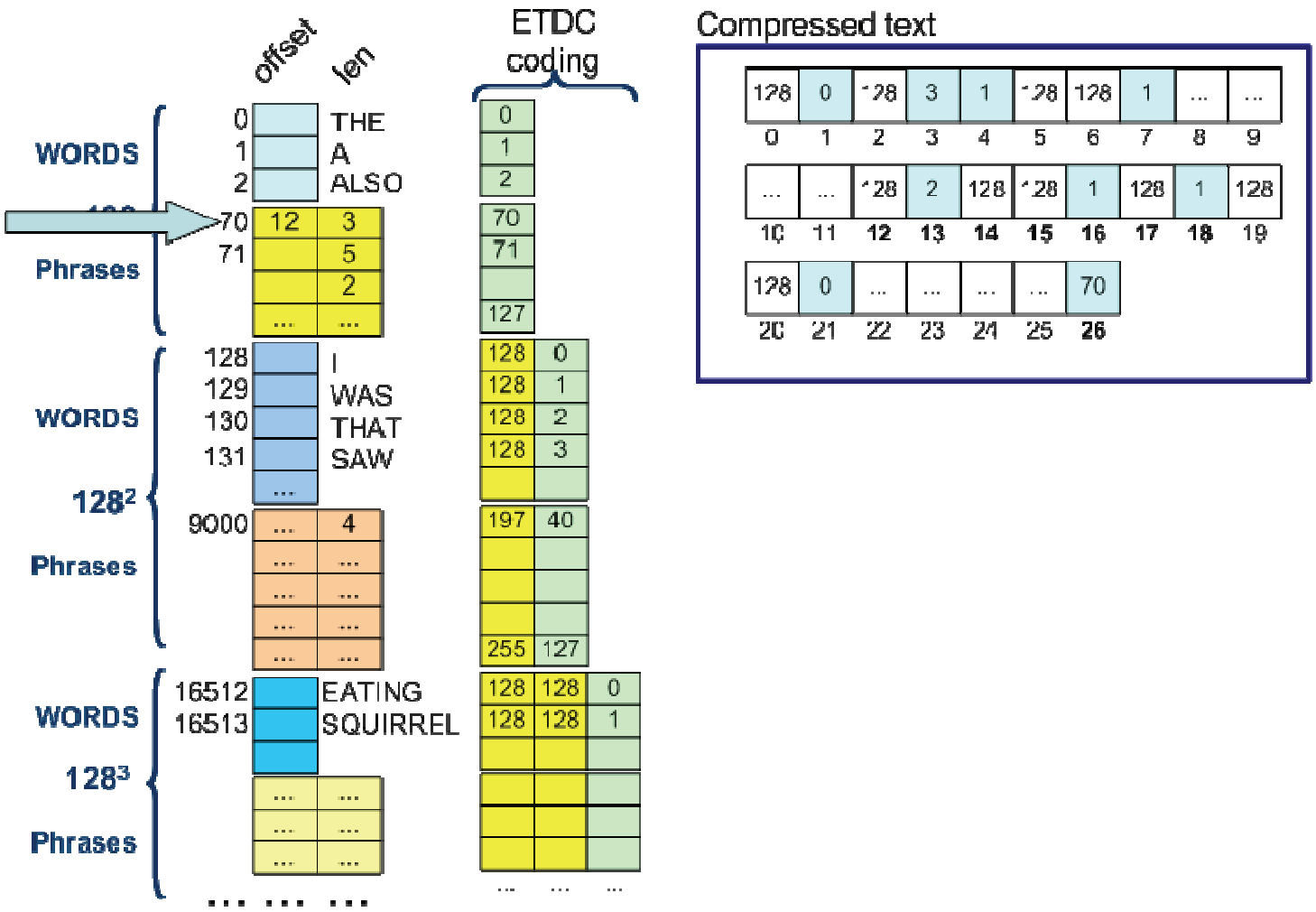


Our Approach (v2vdc)

Fariña et al, DCC 2010

Data structures for compression

I SAW A SQUIRREL [...] THAT SQUIRREL WAS EATING [...] THAT SQUIRREL WAS ALSO [...]

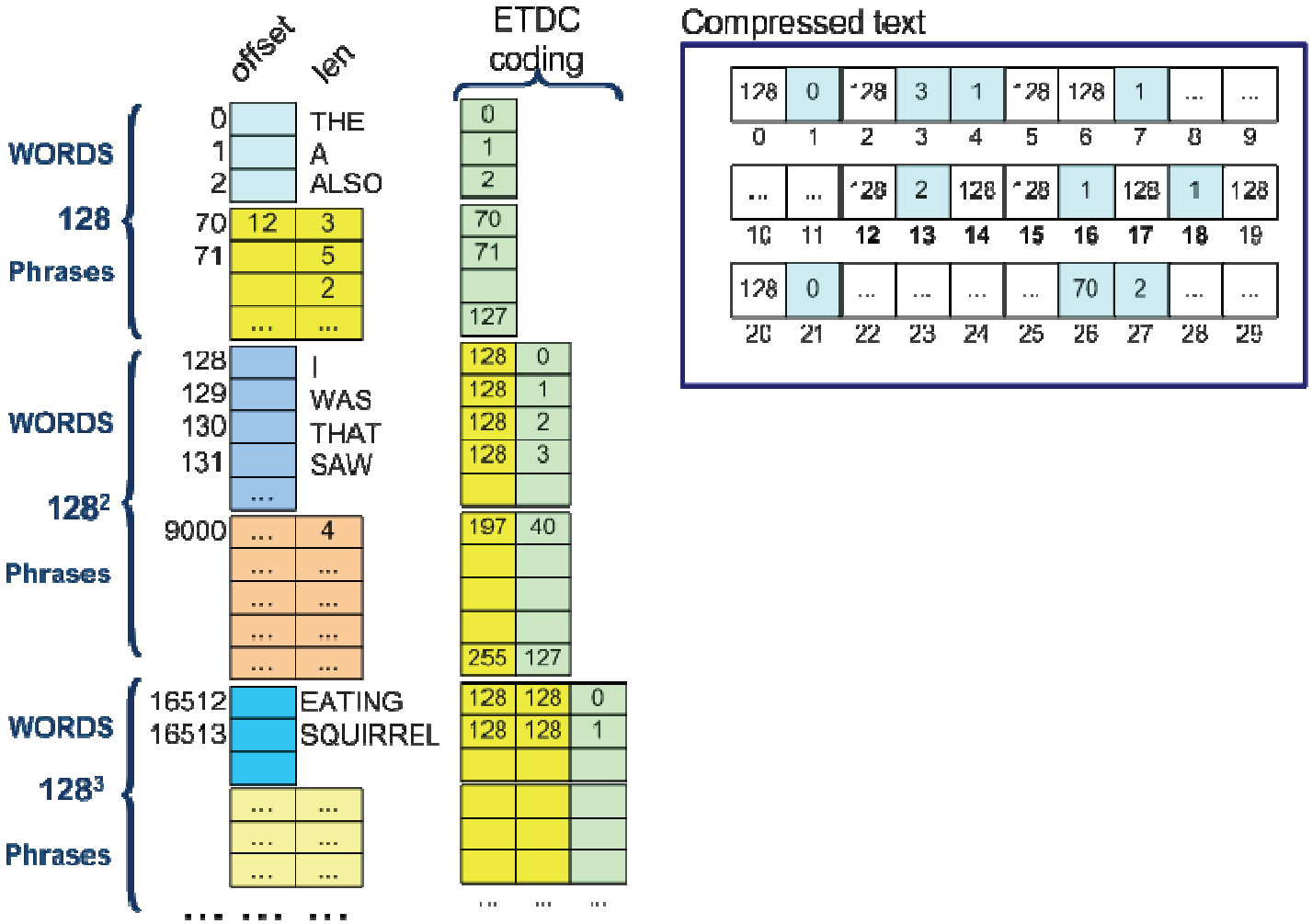


Our Approach (v2vdc)

Fariña et al, DCC 2010

Data structures for compression

I SAW A SQUIRREL [...] THAT SQUIRREL WAS EATING [...] THAT SQUIRREL WAS ALSO [...]

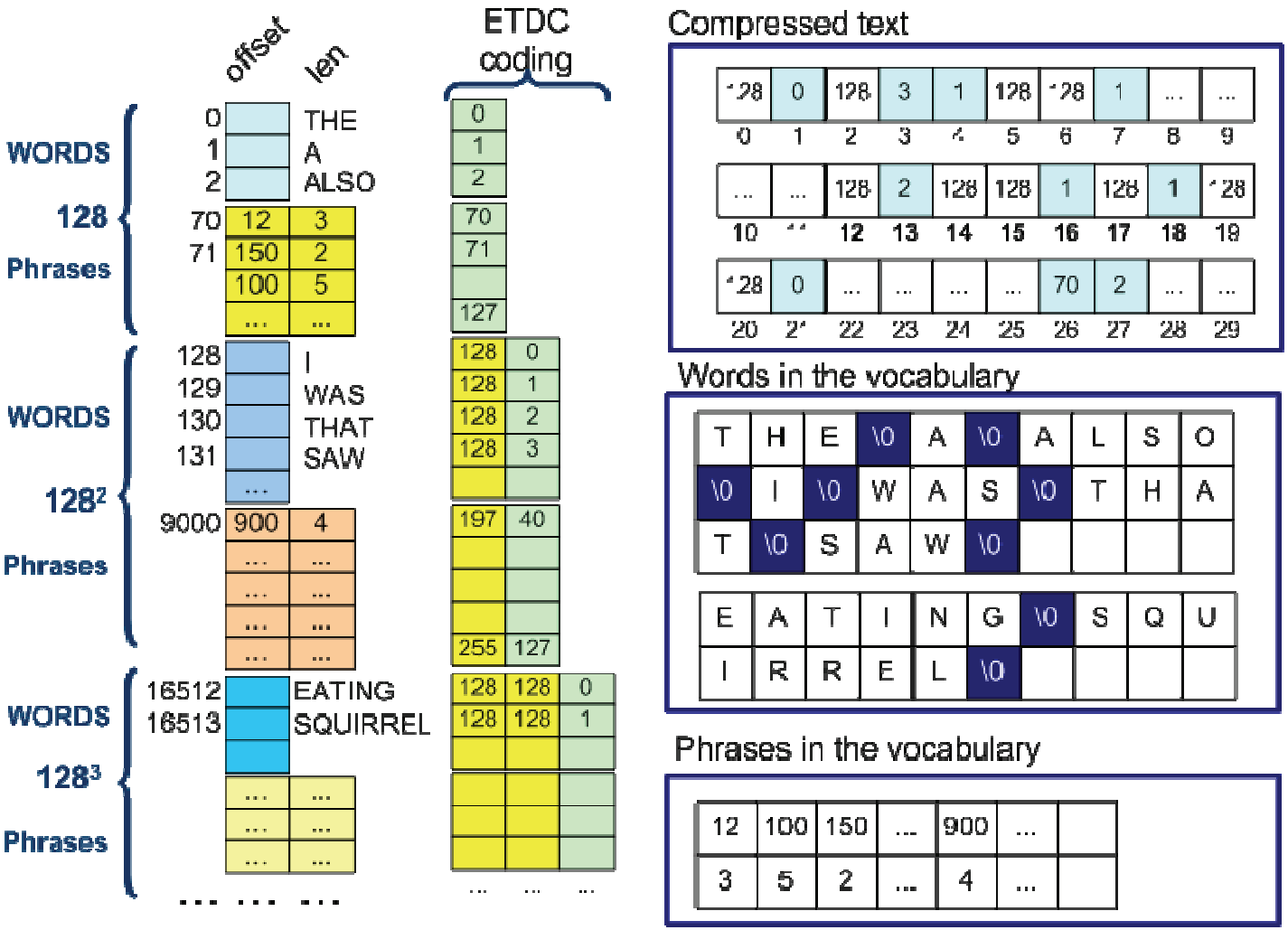


Our Approach (v2vdc)

Fariña et al, DCC 2010

Data structures for compression

I SAW A SQUIRREL [...] THAT SQUIRREL WAS EATING [...] THAT SQUIRREL WAS ALSO [...]

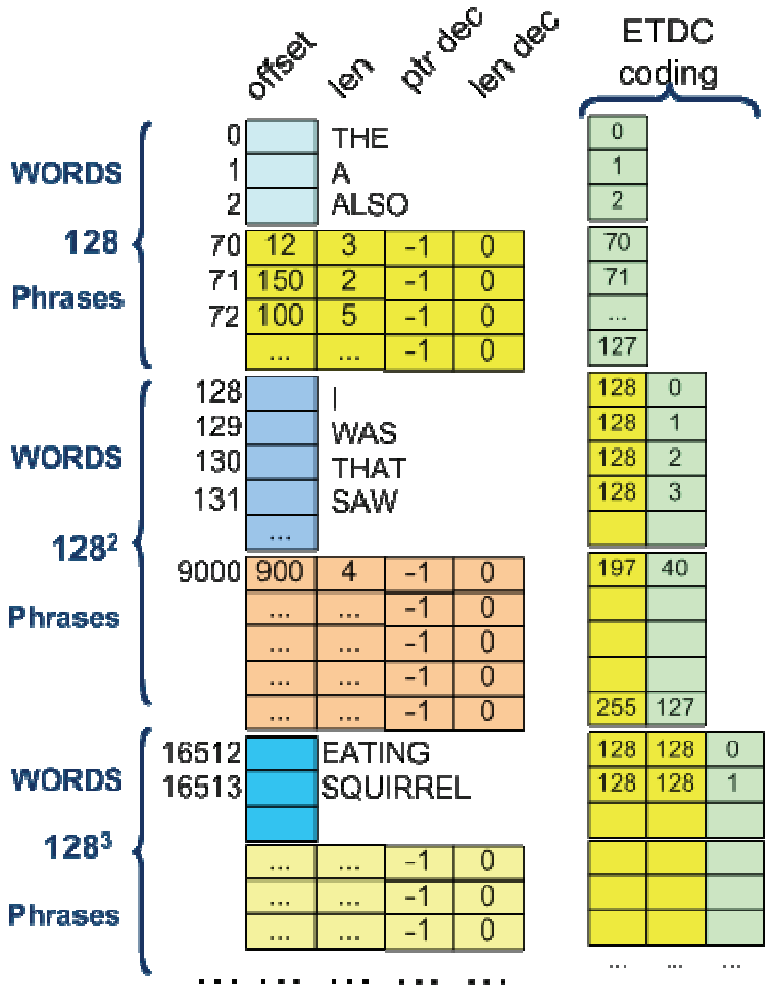


Our Approach (v2vdc)

Fariña et al, DCC 2010

Data structures for decompression

I SAW A SQUIRREL [...] THAT SQUIRREL WAS EATING [...] THAT SQUIRREL WAS ALSO [...]



Beginning of phrases

offset	12	100	150	...	...	900	...	...	...	...
rank	70	72	71	...	...	9000	...	...	...	...
	0	1	2	3	4	5	6	7	8	9

Compressed text

128	0	128	3	1	128	128	1	...	...
0	1	2	3	4	5	6	7	8	9
...	...	128	2	128	128	1	128	1	128
10	11	12	13	14	15	16	17	18	19
128	0	...	...	...	...	70	2	...	...
20	21	22	23	24	25	26	27	28	29

Uncompressed text

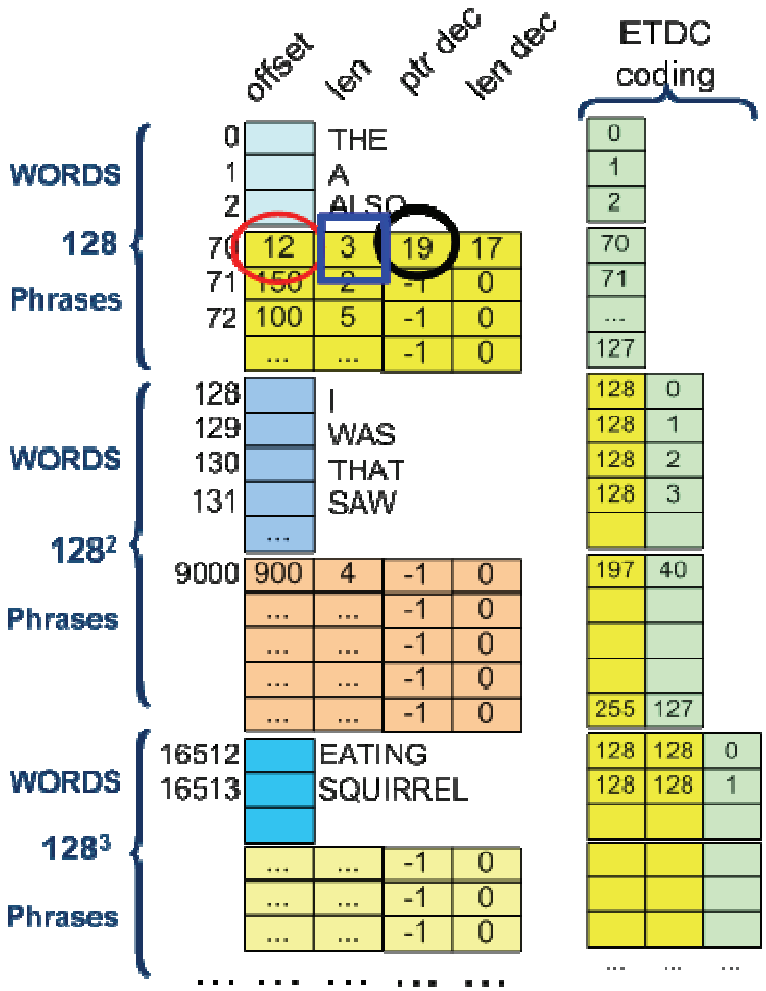
I		S	A	W		A		R	E	I	N	D	E	E
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
R	...	...	...											
15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
45	46	47	48	49	50	51	52	53	54	55	56	57	58	59

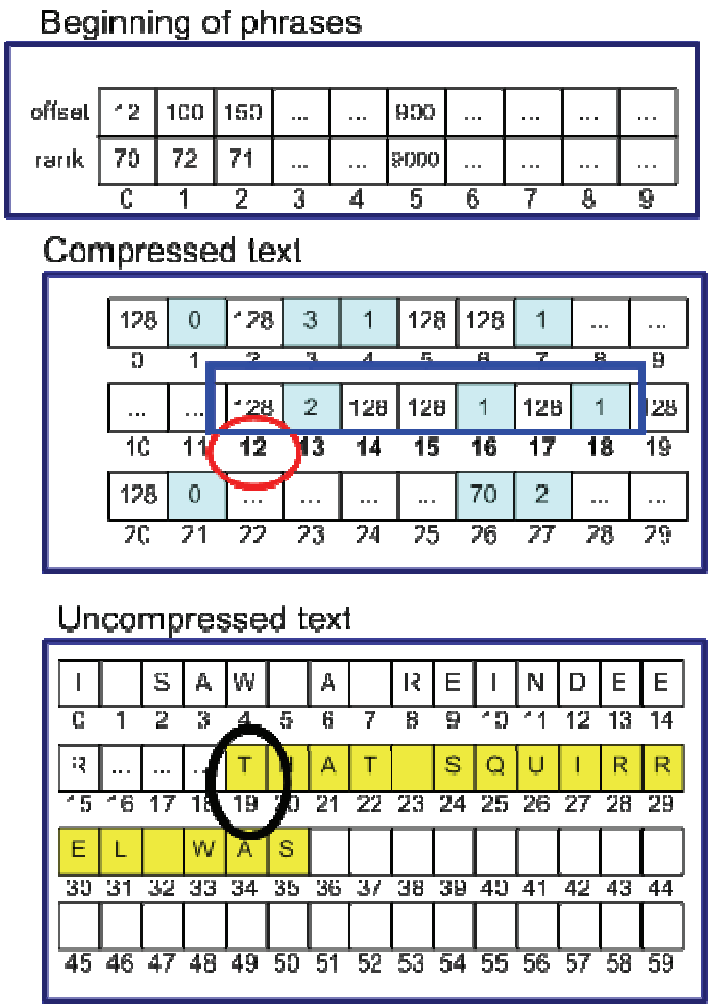
Our Approach (v2vdc)

Fariña et al, DCC 2010

Data structures for decompression

I SAW A SQUIRREL [...] THAT SQUIRREL WAS EATING [...] THAT SQUIRREL WAS ALSO [...]



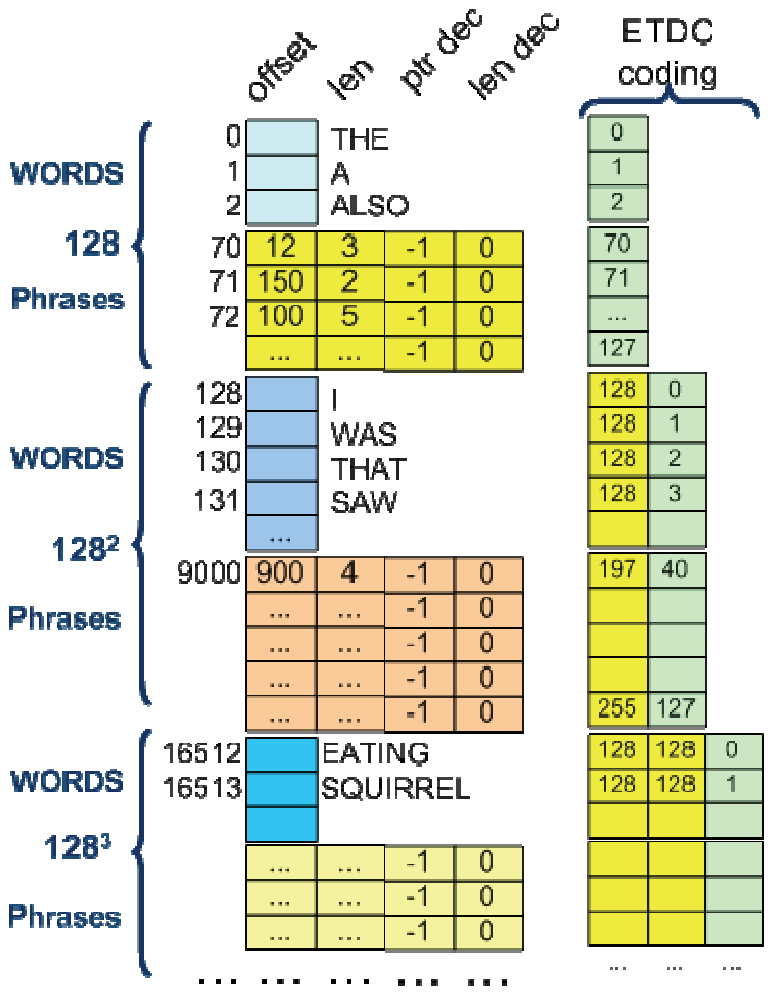


Our Approach (v2vdc)

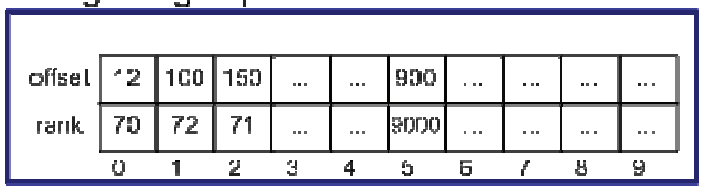
Fariña et al, DCC 2010

Phrase – searching

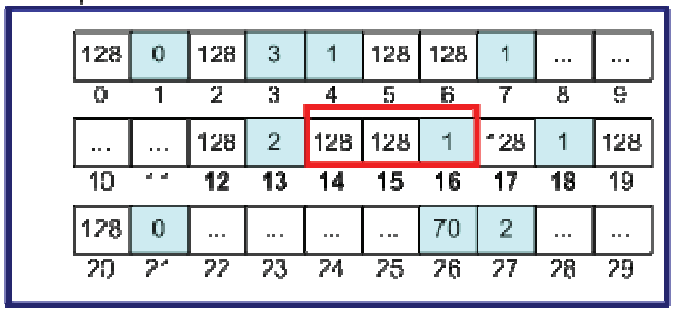
I SAW A SQUIRREL [...] THAT SQUIRREL WAS EATING [...] THAT SQUIRREL WAS ALSO [...]



Beginning of phrases



Compressed text



- Set-Horspool for the least freq. word: (C) ←(ABCD)
- After a match "xCy"
  - Verify it "to left / to right"
  - If compatible phrase with ABCD, including "C"→ add it to the search trie

Outline

- Introduction
  - Variable-to-variable compression
- Our Approach (v2vdc)
  - General structure
    - LCP-based phrase generation & compression
  - Searching
- Experimental results
- Conclusions

Experimental results

Fariña et al, DCC 2010

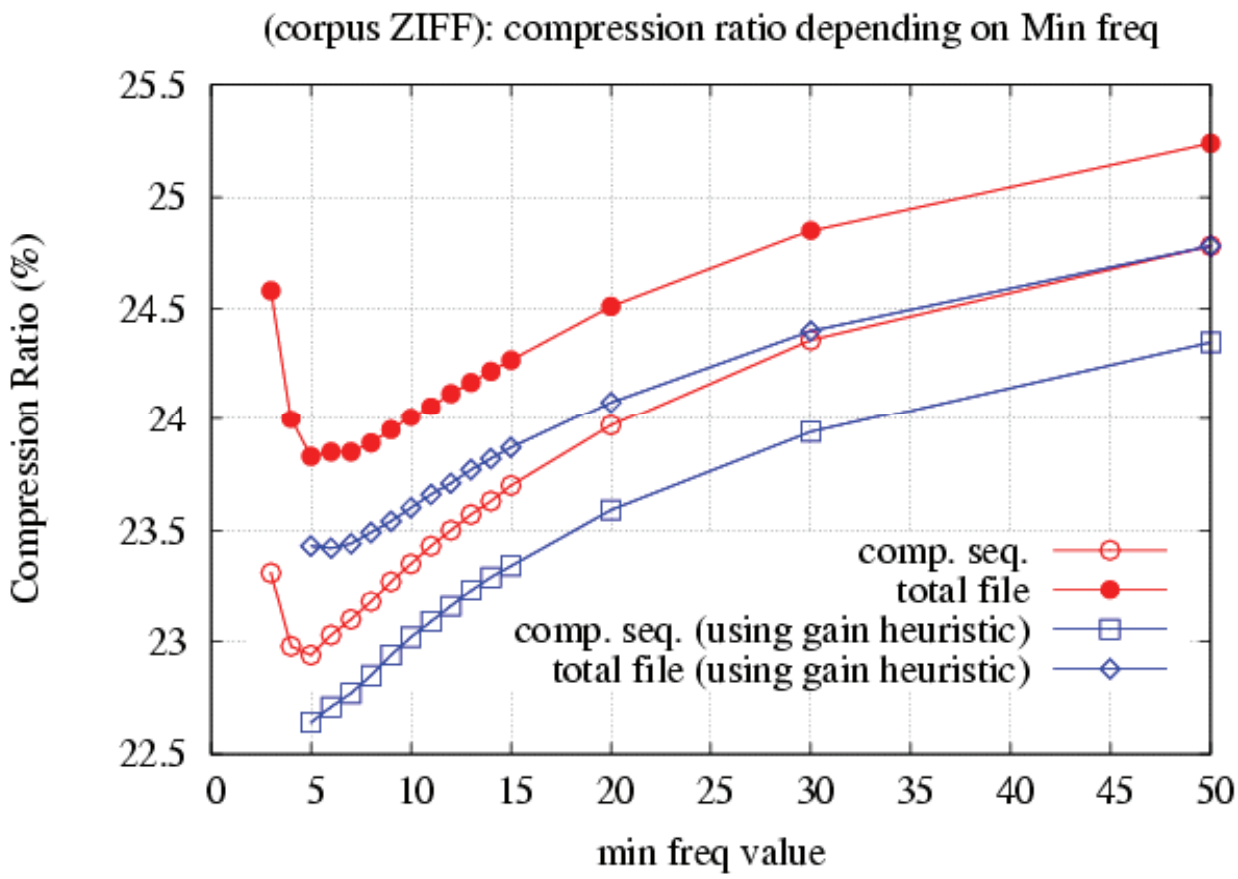
Framework used

- Ziff Data corpus (TREC-2:176.82Mb)
- Intel Core2duo[E6420]@ 2.13GHz, 4GB-DDR2@800MHz
  Ubuntu 8.04, gcc (-O9 –m32)
- Showing:
  - **Compression ratio**, **compression time**, and **decompression time**
    - Comparison against: ETDC, Re-Pair, Bzip2, and P7zip, ppmdi
    - Re-Pair based on Wan && Moffat ‘s Repair: Non searchable approach
  - **Search time**.
    - Comparison against: Horspool (plain text) and ETDC (horspool)

Experimental results

Fariña et al, DCC 2010

Compression Ratio: dependency on threshold

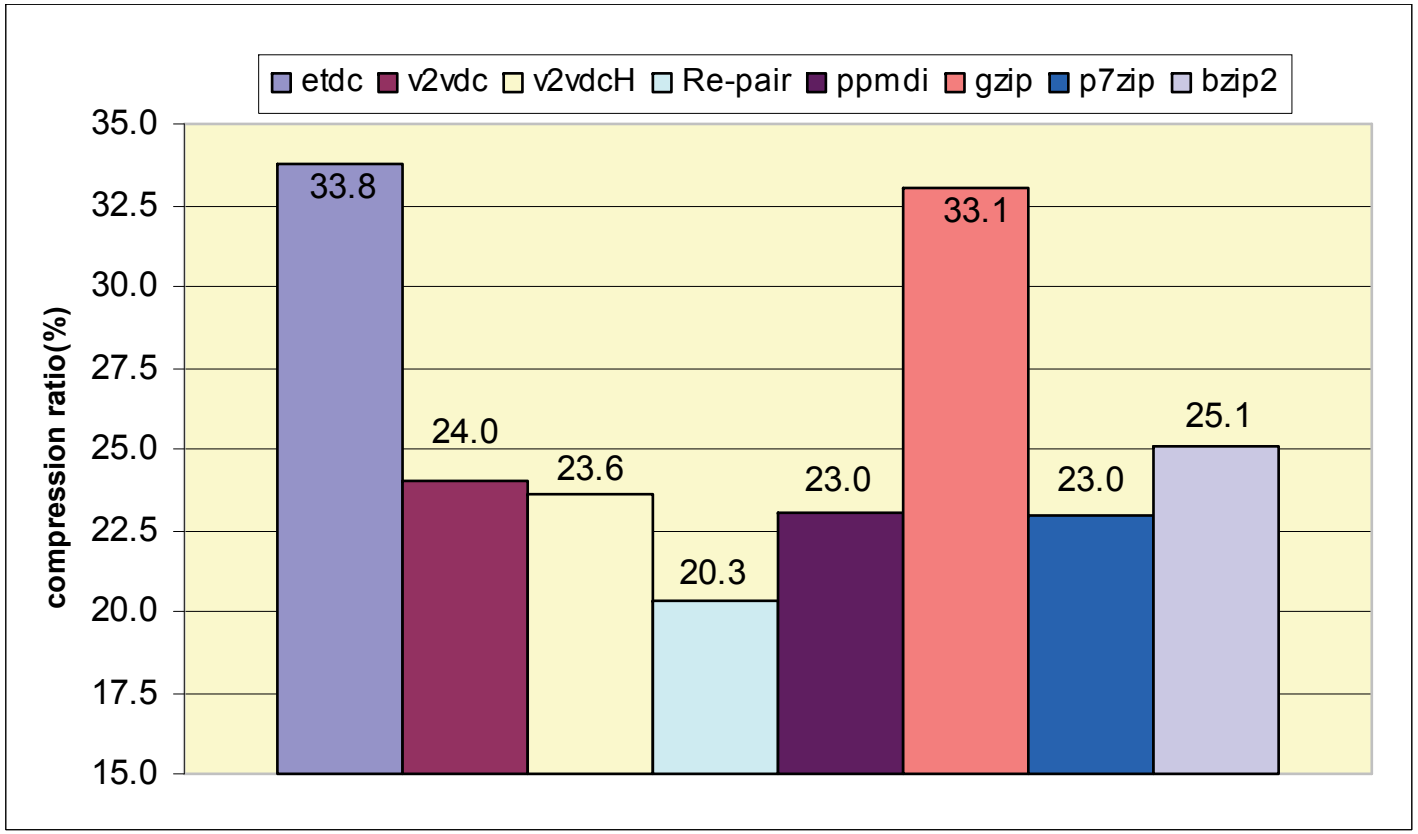


- **Using different threshold values for the freq of candidate phrases.**
- **With heuristic and without heuristic**

Experimental results

Fariña et al, DCC 2010

Compression Ratio: comparison

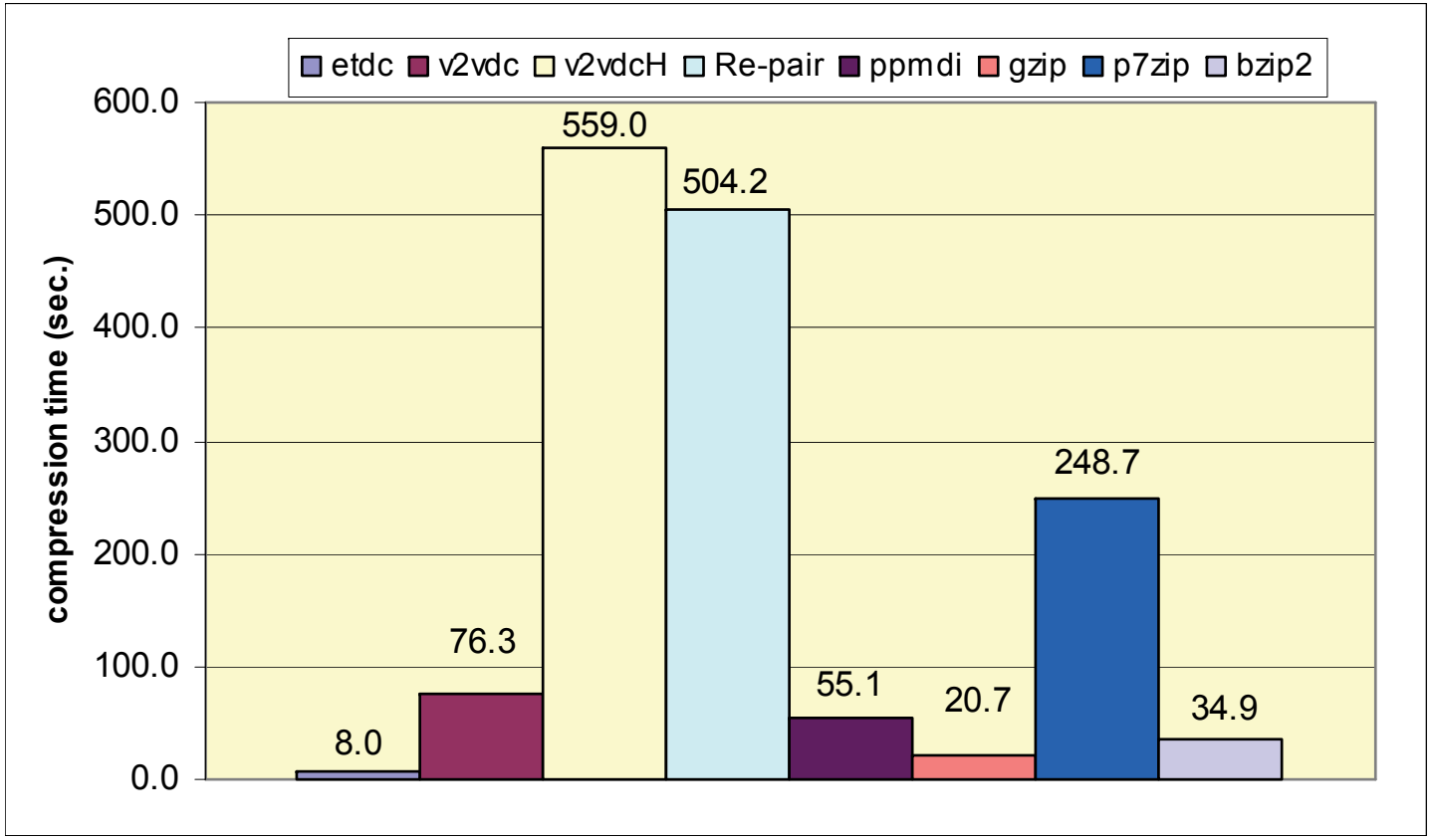


- Improving **ETDC** by around 8 pp (compression ratio)
- Improving **Bzip2**… close to **p7zip** and **ppmdi**… and being **searchable**
- As expected, worse than **Re-Pair** (with Huffman).

Experimental results

Fariña et al, DCC 2010

Compression time comparison

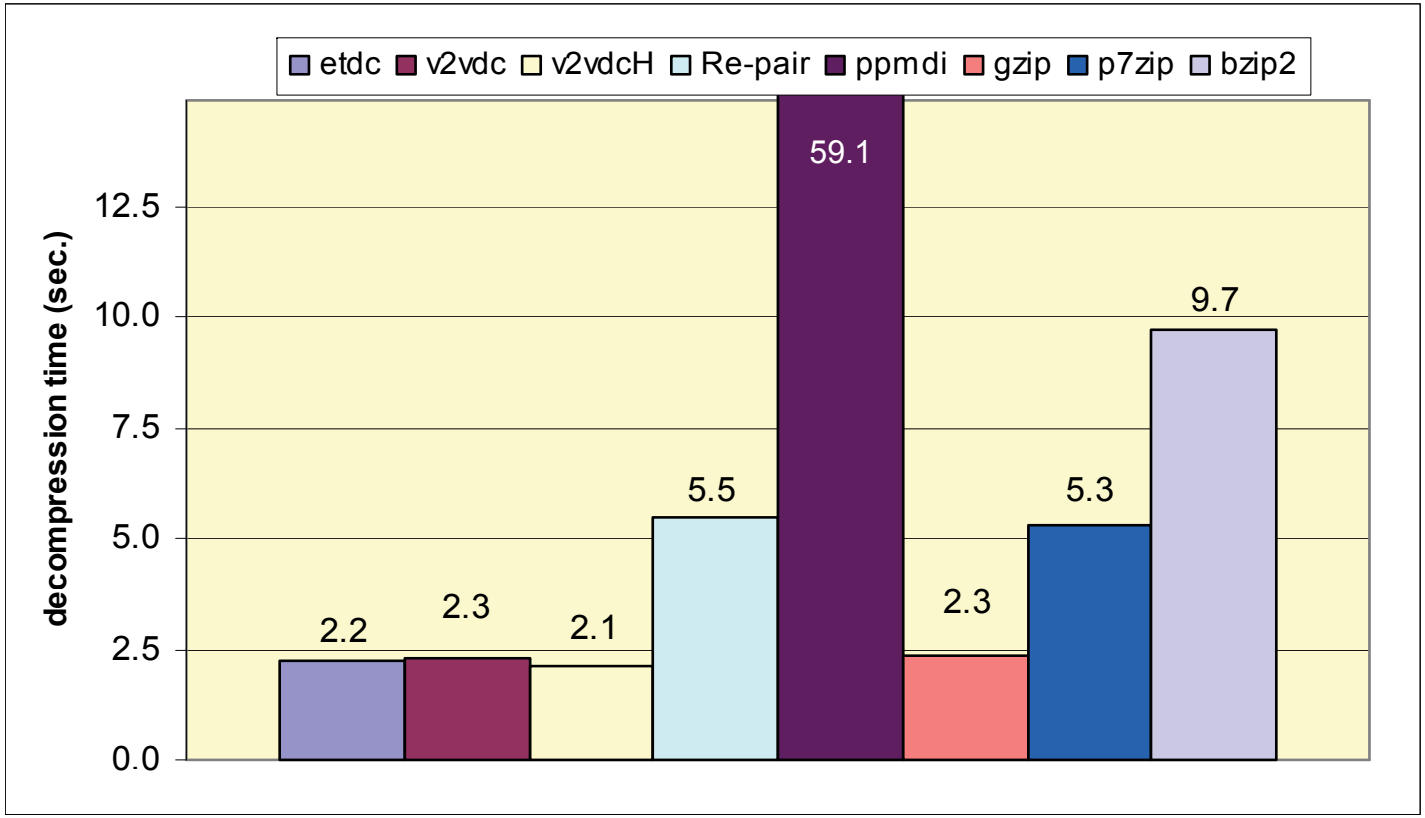


- Very **slow** (building LCP), and very **memory demanding**
- Twice **slower** than **p7zip** (which uses 2 cpu-cores)

Experimental results

Fariña et al, DCC 2010

Decompression time comparison

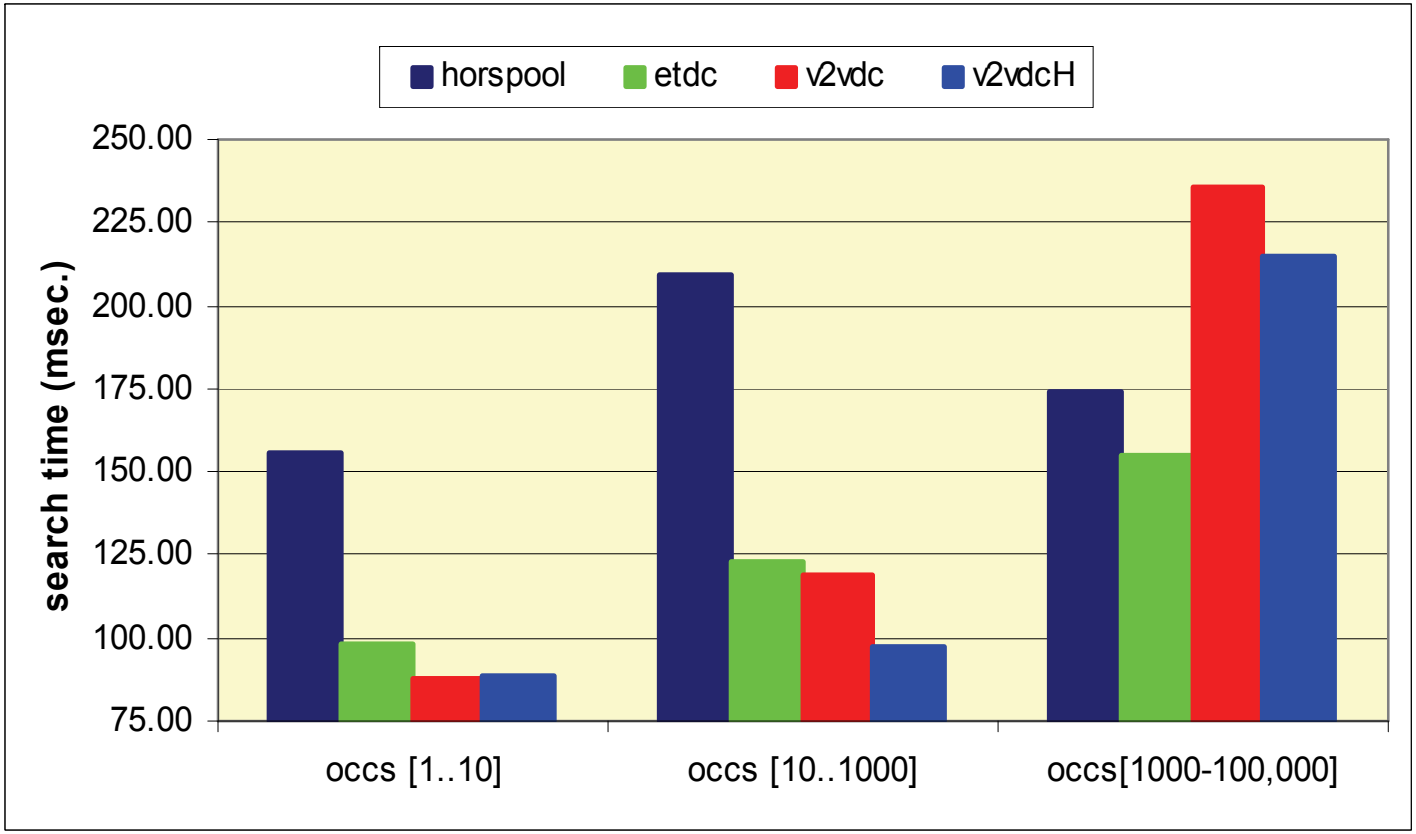


- **Very fast**. Close to **ETDC** and **gzip**
- Around twice faster than **p7zip** and **Re-Pair**
- Much faster than **bzip2** and **ppmdi**

Experimental results

Fariña et al, DCC 2010

Searching (single word patterns)



– Competitive results: comparable to ETDC
  - Overcoming **ETDC** in **low** and **mid-frequency** patterns!!
  - Worsening as the number of occurrences of the pattern increases.

Outline

- Introduction
  - Variable-to-variable compression
- Our Approach (v2vdc)
  - General structure
    - LCP-based phrase generation & compression
  - Searching
- Experimental results
- Conclusions

Conclusions

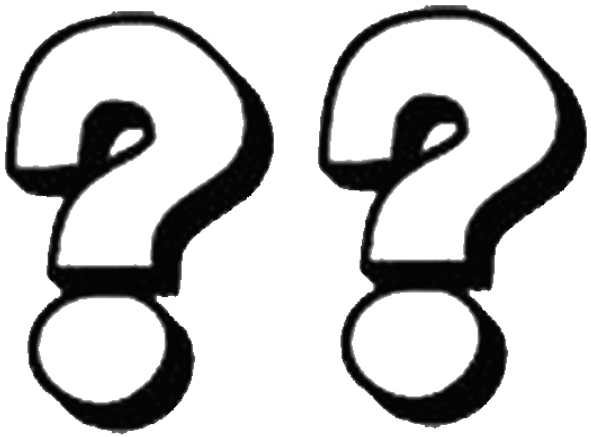
Fariña et al, DCC 2010

- A new **v2vdc** compressor based on word-based ETDC.
  - **Good compression ratio** (around 23-25%)
  - **Slow at compression** and memory demanding (LCP-based)
  - Very **fast** at **decompression**:
    - Similar algorithm to that of ETDC
    - But… less codewords need to be decompressed !!! (1 phrase = several words)
  - Allowing **fast searches**.
    - The best searchable choice with such compression ratio.
- Near **future work**
  - Other heuristics.
  - Multipattern search algoritm.
  - …

Data Compression Conference 2010

Snowbird, UT. March 25, 2010

A new searchable variable-to-variable compressor



Nieves R. Brisaboa

Antonio Fariña

Juan R. López

Eduardo Rodríguez

Gonzalo Navarro