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
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
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, …)

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

**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 \(\rightarrow\) shorter codewords
    - Ex: Huffman, dense codes,…
Introduction

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
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

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 compression

- In the previous example (using Pair-based ETDC)

- Tradeoff “size of compressed text” Vs “size of vocabulary”
Using Phrases: Problems to face

• Choosing good phrases is crucial
  – Using some phrases $\Rightarrow$ forbids choosing “subphrases” or “overlapping” ones
    \[
    \text{ABBCABAACAB} \Rightarrow \{A; \textcolor{red}{B}; \text{BBC}; AB; AAC; AB\}
    \]
  – The number of occs of the words in a phrase “decreases” $\Rightarrow$ a longer codeword could be given to them
    \[
    \text{ABBCABAACAB} \ldots \Rightarrow \{A; BBC; AB; AAC; AB\}
    \]

• 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)

**General structure: 2 main phases**

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

\[
\text{skipbytes} = f_\alpha \times |C_\alpha| + f_\beta \times |C_\beta| \\
\text{addbytes} = f_{\alpha\beta} \times |C_{\alpha\beta}| + (f_\alpha - f_{\alpha\beta}) \times |C'_\alpha| + (f_\beta - f_{\alpha\beta}) \times |C'_\beta| + K
\]
Our Approach (v2vdc)

**LCP-based phrase generation and compression**

- 2nd phase:
  - Compression
    - Replaces any word/phrase by its codeword.
      - $X^{th}$ word $\Rightarrow$ Codeword $C_x$
      - $Y^{th}$ phrase ABC $\Rightarrow$
        - 1st occurrence is encoded as $C_A, C_B, C_C$ (and offset is kept)
        - Rest of occurrences $\Rightarrow$ Codeword $C_y$
  - 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)**

*Data structures for compression*

I SAW A SQUIRREL [...] THAT SQUIRREL WAS EATING [...] THAT SQUIRREL WAS ALSO [...]
Our Approach (v2vdc)

Data structures for compression

I SAW A SQUIRREL [...] THAT SQUIRREL WAS EATING [...] THAT SQUIRREL WAS ALSO [...]
Our Approach (v2vdc)

Data structures for compression

I SAW A SQUIRREL [...] THAT SQUIRREL WAS EATING [...] THAT SQUIRREL WAS ALSO [...]
**Our Approach (v2vdc)**

*Data structures for compression*

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

---

**Compressed text:**

![Compressed text diagram](image)
Our Approach (v2vdc)

Data structures for compression

I saw a squirrel [...] THAT SQUIRREL WAS eating [...] THAT SQUIRREL WAS also [...]
Our Approach (v2vdc)

Data structures for compression

I SAW A SQUIRREL [...] THAT SQUIRREL WAS EATING [...] THAT SQUIRREL WAS ALSO [...]
Our Approach (v2vdc)

Data structures for compression

I SAW A SQUIRREL [...] THAT SQUIRREL WAS EATING [...] THAT SQUIRREL WAS ALSO [...]
Our Approach (v2vdc)

Data structures for decompression

I SAW A SQUIRREL [...] THAT SQUIRREL WAS EATING [...] THAT SQUIRREL WAS ALSO [...]
Our Approach (v2vdc)

Data structures for decompression

I SAW A SQUIRREL [...] THAT SQUIRREL WAS EATING [...] THAT SQUIRREL WAS ALSO [...]
Our Approach (v2vdc)

Phrase – searching

I SAW A SQUIRREL [...] THAT SQUIRREL WAS EATING [...] THAT SQUIRREL WAS ALSO [...]
● Introduction
  ● Variable-to-variable compression
● Our Approach (v2vdc)
  ● General structure
    ● LCP-based phrase generation & compression
  ● Searching
● Experimental results
● Conclusions
Experimental results

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, ppmdii
    - Re-Pair based on Wan && Moffat ‘s Repair: Non searchable approach
  - Search time.
    - Comparison against: Horspool (plain text) and ETDC (horspool)
Experimental results

**Compression Ratio: dependency on threshold**

(corpus ZIFF): compression ratio depending on Min freq

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

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

Compression time comparison

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

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

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

• A new \textit{v2vdc} compressor based on word-based ETDC.
  – \textbf{Good compression ratio} (around 23-25%)
  – \textbf{Slow at compression} and memory demanding (LCP-based)
  – Very \textbf{fast at decompression}:
    • Similar algorithm to that of ETDC
    • But… less codewords need to be decompressed !!! (1 phrase = several words)
  – Allowing \textbf{fast searches}.
    • The best searchable choice with such compression ratio.

• Near \textbf{future work}
  – Other heuristics.
  – Multipattern search algoritm.
  – …
A new searchable variable-to-variable compressor

Nieves R. Brisaboa
Antonio Fariña
Juan R. López
Eduardo Rodríguez
Gonzalo Navarro