Efficient Decodable and Searchable Natural Language Adaptive Compression

Gonzalo Navarro (DCC – U. Chile)

Nieves R. Brisaboa (UDC – Spain)
José R. Paramá (UDC – Spain)
Antonio Fariña (UDC – Spain)
Outline

- Introduction
- Word-based compression
  - Semistatic Compressors
    - Word-based Huffman: Plain & Tagged Huffman
    - *End-Tagged Dense Code (ETDC)*
  - Dynamic Compressors
    - *Dynamic End-Tagged Dense Code (DETDC)*
- Dynamic Lightweight ETDC (DLETDC)
  - Basic Ideas
  - Searching DLETDC
  - Empirical Results
- Conclusions
Introduction

- Why compression? My huge disk is cheap.
- 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 can be integrated into Text Retrieval Systems, improving their performance in all aspects

- Word-based semistatic models proved successful
  - MG system (Witten, Moffat, Bell)
  - Byte-oriented Huffman (Moura, N., Ziviani, Baeza-Yates)
  - End-Tagged Dense Codes (Brisaboa, Fariña, N.)
Introduction

Semistatic compression is preferred in Text Databases

- Reduces their size: **25%-30%** compression ratio
- Permits **direct access** and **local decompression**
- Permits **direct search** on the compressed text
  - Searches are up to 8 times faster.
- Decompression is only needed for presenting results

- **But what about sending results to a remote receiver?**
  - Uncompressed form (wastes network bandwidth)
  - Keep in compressed form (plus the large model of the corpus)
  - Semistatic recompression (uneffective for small files)
  - **Dynamic recompression** (effective, permits a conversation)
    - Gzip (40%), DETDC (32%), arithmetic encoding (25%), …
Introduction

Dynamic compression: Receiver

Decompression (to present results)
Keyword search (to classify documents, alert users, …)

*Dictionary-based (Ziv-Lempel):*
  - Fast decompression
  - Decent direct searching
  - Not so good compression ratios

*Bad for low bandwidth networks*

*Statistical (arithmetic, PPM):*
  - Good compression ratios
  - Costly at sender and receiver side
  - Direct searching impossible

*Bad for weak receivers*
Introduction

- Our contribution: DLETC
  - New dynamic compressor for text databases
    - Simple to understand and implement
  - Statistical
    - Good compression ratio
    - Good for low-bandwidth networks
  - Easier to handle by the receiver
    - Breaks the usual sender/receiver symmetry
    - Very fast decompression
    - Good for weak receivers
  - Searchable without decompressing
    - First statistical method permitting direct search
Outline

- Introduction
- Word-based compression
  - Semistatic Compressors
    - Word-based Huffman: Plain & Tagged Huffman
    - End-Tagged Dense Code (ETDC)
  - Dynamic Compressors
    - Dynamic End-Tagged Dense Code (DETDC)
- Dynamic Lightweight ETDC (DLETDC)
  - Basic Ideas
  - Searching DLETDC
  - Empirical Results
- Conclusions
Semistatic compression

- Statistical semistatic compression

- 2 passes
  - Association between source symbol <-> codeword does not change across the text
  - Direct search is possible
  - Most representative method: Huffman.
Word-based Huffman

- Moffat proposed the use of words instead of characters as the coding alphabet.
- Distribution of words more biased than that of characters.
- Compression ratio about 25% (English texts).
- This idea joins the requirements of compression algorithms and of Information Retrieval systems.
Plain Huffman & Tagged Huffman

- Moura, Navarro, Ziviani and Baeza:
  - 2 new techniques: Plain Huffman and Tagged Huffman

- Common features
  - Huffman-based
  - Word-based
  - Byte- rather than bit-oriented (compression ratio ±30%)

- Plain Huffman = Huffman over bytes (256-ary tree)
- Tagged Huffman flags the beginning of each codeword

First bit is:
\[
\begin{align*}
&\text{"1"} \rightarrow \text{for 1st bit of 1st byte} \\
&\text{"0"} \rightarrow \text{for 1st bit remaining bytes}
\end{align*}
\]

1xxxxxxxx 0xxxxxxxx 0xxxxxxxx
Plain Huffman & Tagged Huffman

Differences:

- Plain Huffman. (tree of arity $2^b=256$)
- Tagged Huffman. (tree of arity $2^{b-1}=128$)

- Direct search (improved searches)
  - Boyer-Moore
- Random access (random decompression)
Outline

- Introduction
- Word-based compression
  - Semi-static Compressors
    - Word-based Huffman: Plain & Tagged Huffman
    - End-Tagged Dense Code (ETDC)
  - Dynamic Compressors
    - Dynamic End-Tagged Dense Code (DETDC)
- Dynamic Lightweight ETDC (DLETDC)
  - Basic Ideas
  - Searching DLETDC
  - Empirical Results
- Conclusions
End-Tagged Dense Code

- Small change: Flag signal the end of a codeword
  - First bit is:
    - "1" --> for 1st bit of last byte
    - "0" --> for 1st bit remaining bytes

| Two-byte codeword | 0xxxxxxx | 1xxxxxxx |
| Three-byte codeword | 0xxxxxxx | 0xxxxxxx | 1xxxxxxx |

Huffman tree is not needed: Dense coding.
- Improving Tagged Huffman compression ratio by 2.5 perc. points

- Flag bit → Same Tagged Huffman searching capabilities
## End-Tagged Dense Code

### Encoding scheme

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000000 10000001 11111111</td>
<td>First 128 words are encoded using <strong>one byte</strong> (2^7 codewords)</td>
</tr>
<tr>
<td>00000000 10000000 01111111</td>
<td>Words from 128+1 to 128+128^2 are encoded using <strong>two bytes</strong> (128^2 = 2^{14} codewords)</td>
</tr>
<tr>
<td>00000000 00000000 10000000 01111111</td>
<td>Words from 128+ 128^2+1 to 128 +128^2 +128^3 use <strong>three bytes</strong> (128^3 = 2^{21} codewords)</td>
</tr>
</tbody>
</table>

- Codewords depend on the rank, not on the frequency
Outline

- Introduction
- Word-based compression
  - Semi-static Compressors
    - Word-based Huffman: Plain & Tagged Huffman
    - *End-Tagged Dense Code (ETDC)*
  - Dynamic Compressors
    - *Dynamic End-Tagged Dense Code (DETDC)*
- Dynamic Lightweight ETDC (DLETDC)
  - Basic Ideas
  - Searching DLETDC
  - Empirical Results
- Conclusions
Dynamic codes

- Statistical dynamic compression
  - Dynamic modeling
    - Frequencies are computed dynamically as text arrives.
  - Sender and receiver…
    - Start with an empty model
    - Adapt their model during the process.
    - Both processes are symmetric
Dynamic End-Tagged Dense Code

- Symmetric sender and receiver.
- It uses the on-the-fly encoding and decoding algorithms
  - Requirement: maintaining the vocabulary sorted by freq.
- Example

<table>
<thead>
<tr>
<th>word</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>8</td>
</tr>
<tr>
<td>is</td>
<td>4</td>
</tr>
<tr>
<td>code</td>
<td>2</td>
</tr>
</tbody>
</table>

- The codeword given to a word $s_i$ may change each time $s_i$ is processed.
Dynamic End-Tagged Dense Code: transmission

Example: ... a rose **rose** is a very nice one

```
<table>
<thead>
<tr>
<th>word</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>rose</td>
<td>1</td>
</tr>
</tbody>
</table>
```

sender

```
<table>
<thead>
<tr>
<th>word</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>126</td>
<td></td>
</tr>
<tr>
<td>127</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td></td>
</tr>
<tr>
<td>129</td>
<td></td>
</tr>
</tbody>
</table>
```

receiver

```
<table>
<thead>
<tr>
<th>word</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>126</td>
<td>1</td>
</tr>
<tr>
<td>127</td>
<td>1</td>
</tr>
<tr>
<td>128</td>
<td>1</td>
</tr>
<tr>
<td>129</td>
<td></td>
</tr>
</tbody>
</table>
```
Dynamic End-Tagged Dense Code: transmission

Example: … a rose rose **is** a very nice one
Outline

- Introduction
- Word-based compression
  - Semi-static Compressors
    - Word-based Huffman: Plain & Tagged Huffman
    - End-Tagged Dense Code (ETDC)
  - Dynamic Compressors
    - Dynamic End-Tagged Dense Code (DETDC)
- Dynamic Lightweight ETDC (DLETDC)
  - Basic Ideas
  - Searching DLETDC
  - Empirical Results
- Conclusions
**DLETDC – Basic Ideas**

- **The idea:**
  - Break the correspondence between position and codeword
  - Codewords are maintained explicitly by the sender
  - SENDER: After processing a symbol $s_i$ ...
    - Exchange $s_i \leftrightarrow \text{top}(f_j)$ to keep the vocabulary sorted,
    - Keep original codewords unless codeword length should vary
    - Otherwise swap, and notify the receiver.
  - 2 fixed slots are reserved:
    - new-symbol, ascii word
    - Swap, $C_i$, CodeIn $j$
  - RECEIVER: does not maintain frequencies
    - Decodes codewords
    - Adds new symbols ($C_{\text{new-symbol}}$)
    - Only swaps when it decodes a $C_{\text{swap}}$
**Example:** … a rose rose is a very nice one

<table>
<thead>
<tr>
<th>vocabulary</th>
</tr>
</thead>
<tbody>
<tr>
<td>word</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>rose</td>
</tr>
<tr>
<td>is</td>
</tr>
<tr>
<td>a</td>
</tr>
<tr>
<td>very</td>
</tr>
</tbody>
</table>

No codeword swap needed since: “a” $\leftrightarrow$ C_{127} and “is” $\leftrightarrow$ C_{126}
Example: … a rose rose is a **very** nice one

A codeword swap is needed
DLETDC: transmission

Example: ... a rose rose is a very nice one

vocabulary

<table>
<thead>
<tr>
<th>word</th>
<th>freq</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>rose</td>
<td>2</td>
<td>C_{125}</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
<td>C_{127}</td>
</tr>
<tr>
<td>very</td>
<td>2</td>
<td>C_{126}</td>
</tr>
<tr>
<td>is</td>
<td>1</td>
<td>C_{128}</td>
</tr>
<tr>
<td>nice</td>
<td>1</td>
<td>C_{129}</td>
</tr>
</tbody>
</table>

A new word case
DLETDC – Evolution of swaps.

- Swaps worsen compression ratio and processing time
- How many swaps occur in practice?

ZIFF corpus
4,6x10^7 words
237,622 diff words

Between lengths 2 and 3
Outline

- Introduction
- Word-based compression
  - Semi-static Compressors
    - Word-based Huffman: Plain & Tagged Huffman
    - End-Tagged Dense Code (ETDC)
  - Dynamic Compressors
    - Dynamic End-Tagged Dense Code (DETDC)
- Dynamic Lightweight ETDC (DLETDC)
  - The code
  - Searching DLETDC
  - Empirical Results
- Conclusions
DLETDC – Searches – Keyword filtering

- Multipattern Horspool
  - A trie is traversed to recognize text backwards

- Search process:
  - Initial phase
    - It suffices to search for:
      - \( C_{\text{new}} \)
      - \( C_{\text{pat}} \)

and make some neighborhood checks
Outline

- Introduction
- Word-based compression
  - Semi-static Compressors
    - Word-based Huffman: Plain & Tagged Huffman
    - *End-Tagged Dense Code (ETDC)*
  - Dynamic Compressors
    - *Dynamic End-Tagged Dense Code (DETDC)*
- Dynamic Lightweight ETDC (DLETDC)
  - The code
  - Searching DLETDC
  - Empirical Results
- Conclusions
Empirical Results

- We used some text collections from TREC-2 & TREC-4, to perform the experiments

<table>
<thead>
<tr>
<th>CORPUS</th>
<th>size (bytes)</th>
<th># words</th>
<th>Diff. words</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALGARY</td>
<td>2,131,045</td>
<td>528,611</td>
<td>30,995</td>
</tr>
<tr>
<td>FT91</td>
<td>14,749,355</td>
<td>3,135,383</td>
<td>75,681</td>
</tr>
<tr>
<td>CR</td>
<td>51,085,545</td>
<td>10,230,907</td>
<td>117,713</td>
</tr>
<tr>
<td>FT92</td>
<td>175,449,235</td>
<td>36,803,204</td>
<td>284,892</td>
</tr>
<tr>
<td>ZIFF</td>
<td>185,220,215</td>
<td>40,866,492</td>
<td>237,622</td>
</tr>
<tr>
<td>FT93</td>
<td>197,586,294</td>
<td>42,063,804</td>
<td>291,427</td>
</tr>
<tr>
<td>FT94</td>
<td>203,783,923</td>
<td>43,335,126</td>
<td>295,018</td>
</tr>
<tr>
<td>AP</td>
<td>250,714,271</td>
<td>53,349,620</td>
<td>269,141</td>
</tr>
</tbody>
</table>

- Dual Intel Pentium-III 800 Mhz with 768Mb RAM.
  - Debian GNU/Linux (kernel 2.2.19)
  - gcc 3.3.3 20040429 and –O9 optimizations
  - Time represents CPU user-time
Empirical Results. Compression Ratio

Comparison with other techniques.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Compression Ratio (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>gzip</td>
<td>35.09</td>
</tr>
<tr>
<td>DLETDC</td>
<td>33.69</td>
</tr>
<tr>
<td>DETDC</td>
<td>33.66</td>
</tr>
<tr>
<td>arith</td>
<td>27.98</td>
</tr>
<tr>
<td>bzip2</td>
<td>25.98</td>
</tr>
</tbody>
</table>
Empirical Results. Compression time

<table>
<thead>
<tr>
<th>Technique</th>
<th>Compression time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>gzip</td>
<td>431.5</td>
</tr>
<tr>
<td>DLETDC</td>
<td>154.6</td>
</tr>
<tr>
<td>DETDC</td>
<td>150.1</td>
</tr>
<tr>
<td>arith</td>
<td>510.0</td>
</tr>
<tr>
<td>bzip2</td>
<td>1342.0</td>
</tr>
</tbody>
</table>
Empirical Results. Decompression time

- **gzip**: 59.8 seconds
- **DLETDC**: 49.5 seconds
- **DETDC**: 75.8 seconds
- **arith**: 394.1 seconds
- **bzip2**: 432.4 seconds
Search speed

- Avg time over 10,000 random searches for words of length >= 6
Outline

- Introduction
- Word-based compression
  - Semi-static Compressors
    - Word-based Huffman: Plain & Tagged Huffman
    - End-Tagged Dense Code (ETDC)
  - Dynamic Compressors
    - Dynamic End-Tagged Dense Code (DETDC)
- Dynamic Lightweight ETDC (DLETDC)
  - Basic Ideas
  - Searching DLETDC
  - Empirical Results
- Conclusions
Conclusions

- We have presented DLETDC
  - A new dynamic “dense” compressor.
  - Designed for low bandwidth and weak receivers.
- Competitive compression ratio (around 32-34%, < gzip)
- Fast at compression (much faster than other adaptive)
- Very fast at decompression (even faster than gunzip!)
- Very fast at searching (>3 times faster than plain searching)

- DLETDC breaks the common symmetry between sender and receiver in adaptive techniques.
  - Lightweight and fast decompressor/receiver.
  - A dynamic searchable technique.