

Salvador de Bahía. Aug 2005
28th Annual International ACM SIGIR

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 (ineffective 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

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

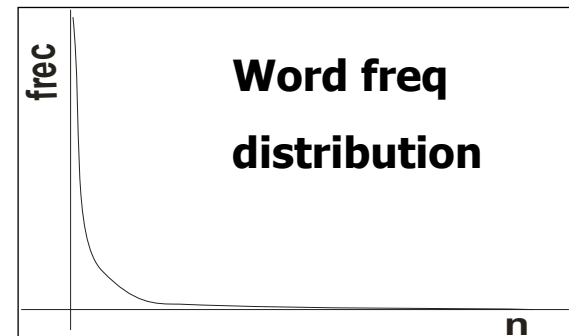
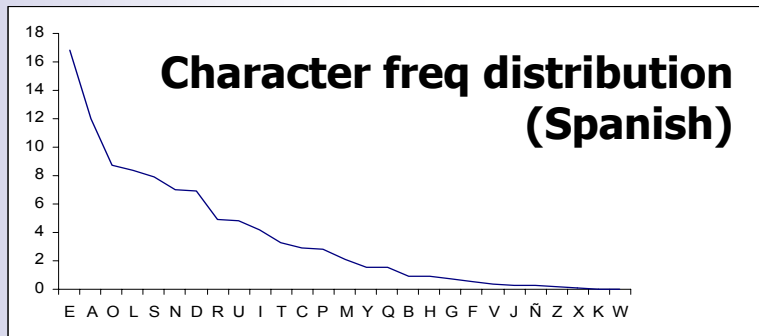
- Statistical semistatic compression

2 passes

- Association between source symbol \leftrightarrow 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 $\pm 30\%$)
- Plain Huffman = Huffman over bytes (256-ary tree)
- Tagged Huffman **flags** the beginning of each codeword

First bit is: {
• “1” → for 1st bit of 1st byte
• “0” → for 1st bit remaining bytes

1xxxxxxx 0xxxxxxx 0xxxxxxx

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)

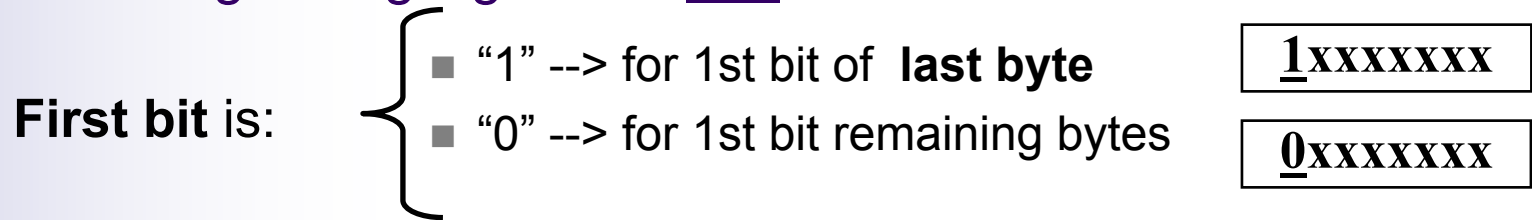
Outline

- 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



Two-byte codeword

0XXXXXXXX

1XXXXXXXX

Three-byte codeword

0XXXXXXXX

0XXXXXXXX

1XXXXXXXX

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

<pre> 10000000 10000001 11111111 </pre>	<p>First 128 words are encoded using one byte (2^7 codewords)</p>
<pre> 00000000:10000000 01111111:11111111 </pre>	<p>Words from 128+1 to 128+128² are encoded using two bytes ($128^2 = 2^{14}$ codewords)</p>
<pre> 00000000:00000000:10000000 01111111:01111111:11111111 </pre>	<p>Words from 128+ 128²+1 to 128 +128² +128³ use three bytes ($128^3 = 2^{21}$ codewords)</p>
<pre> </pre>	

- 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

Sender's vocabulary

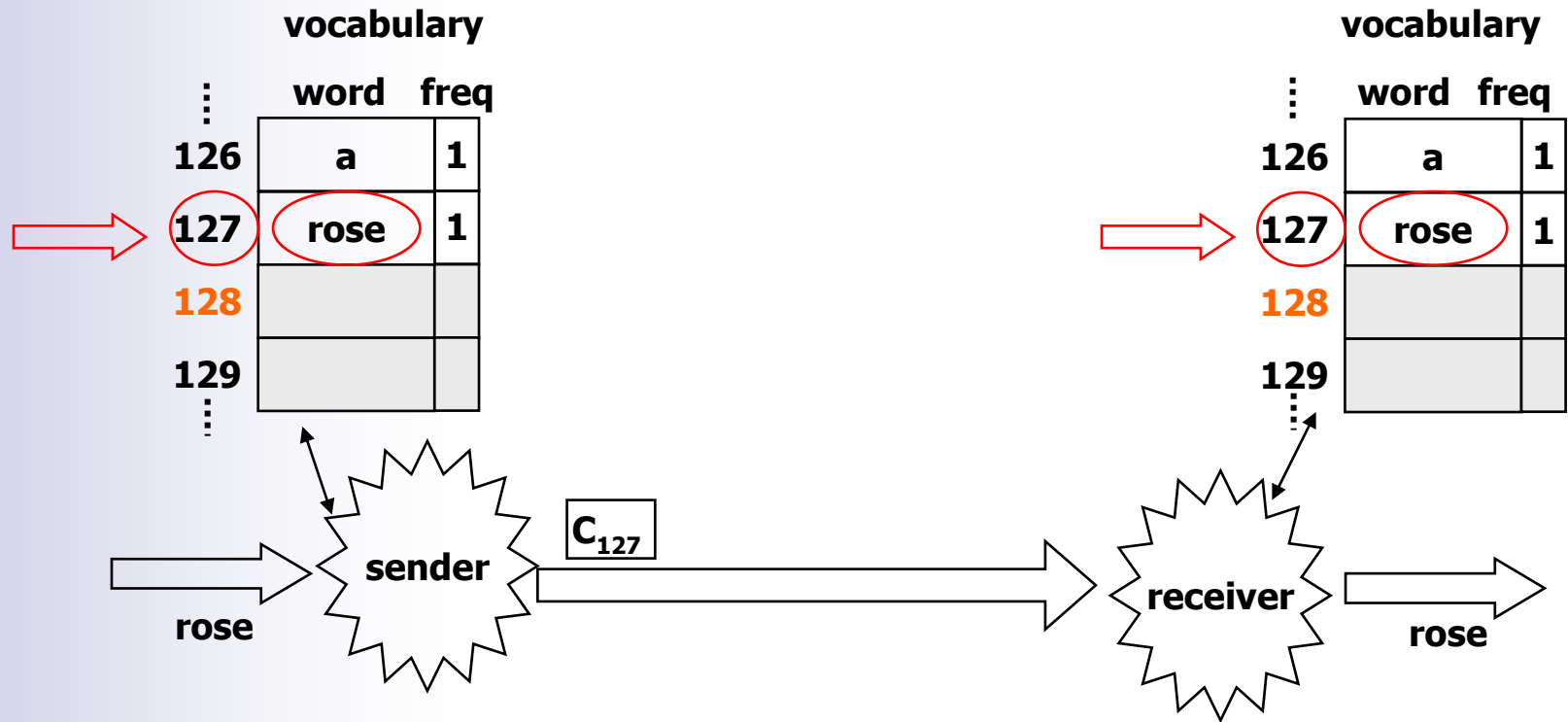
	word	freq
1	the	8
2	is	4
3	code	2

DETDC ...

- The codeword given to a word s_i may change each time s_i is processed.

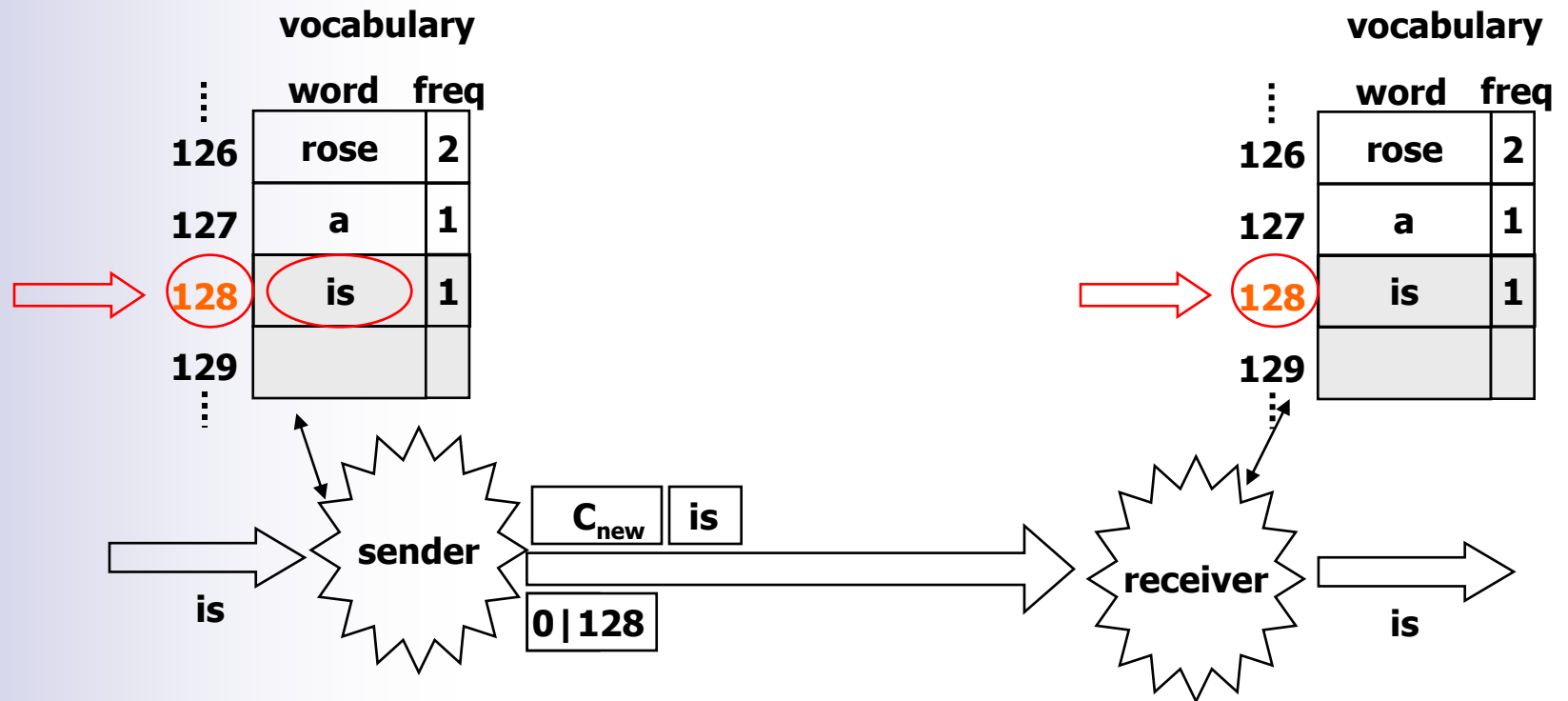
Dynamic End-Tagged Dense Code: transmission

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



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 ...

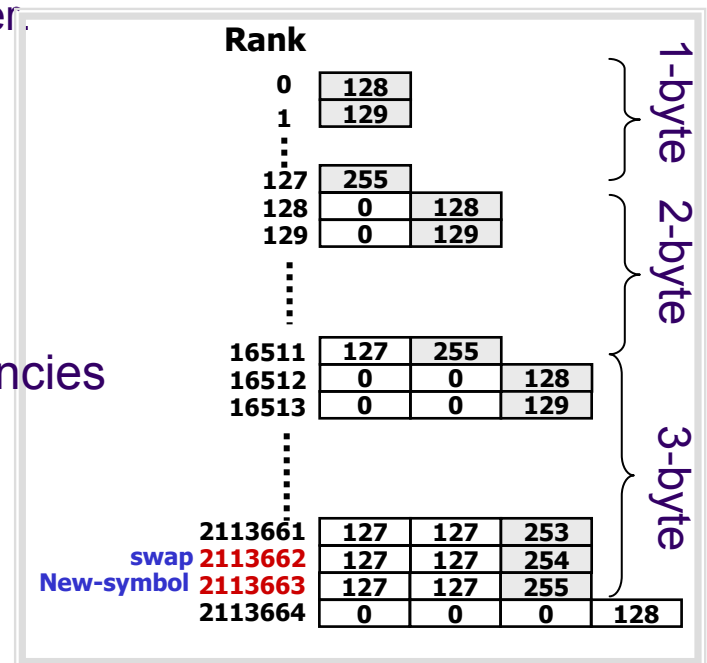
- ▶ Exchange $s_i \leftrightarrow \text{top}(f_i)$ to keep the vocabulary sorted,
- ▶ Keep original codewords unless codeword length should vary
- ▶ Otherwise **swap**, and notify the receiver.

- 2 fixed slots are reserved:

- ▶ **new-symbol, ascii word**
- ▶ **Swap, C_i , CodeIn_j .**

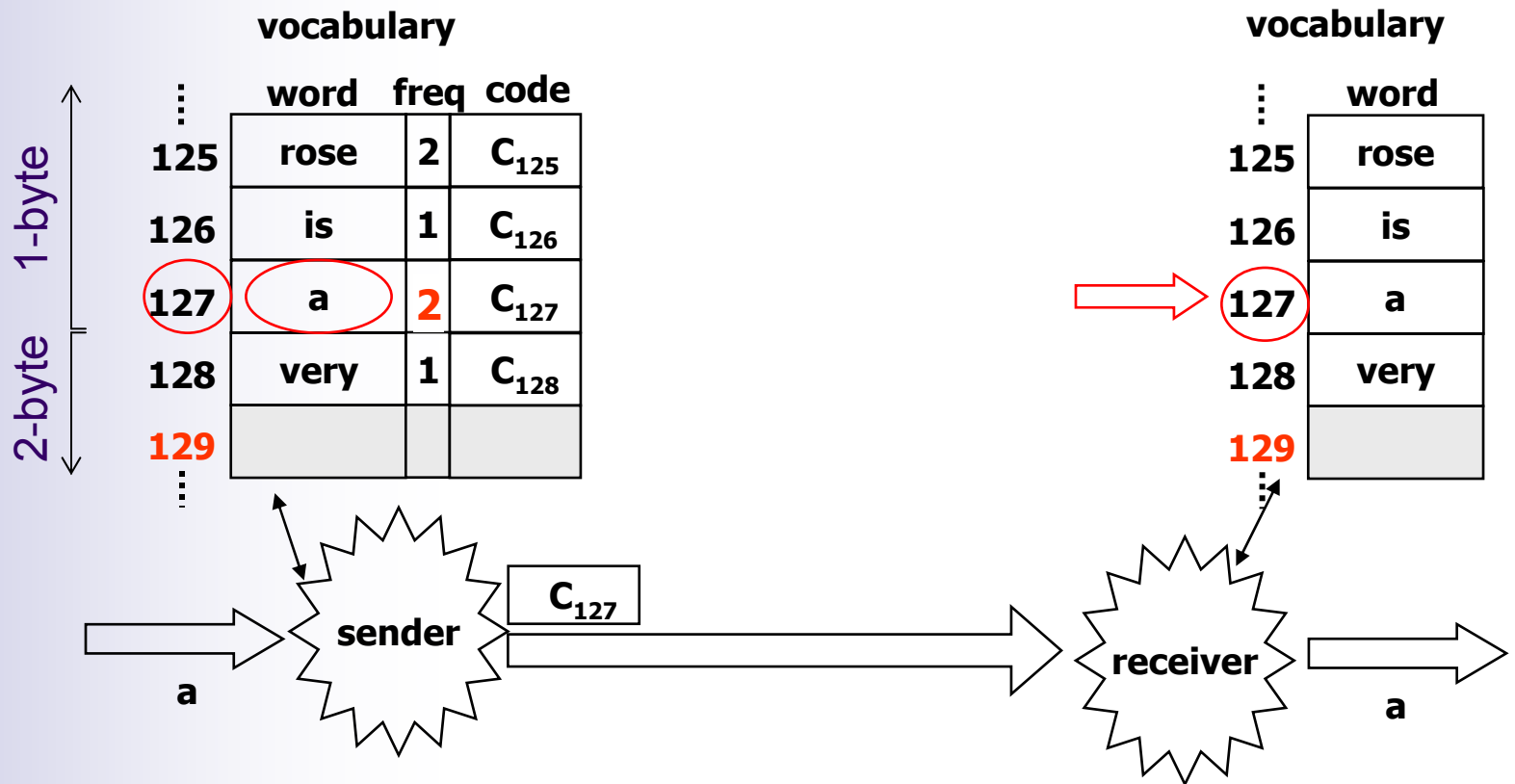
- RECEIVER: does not maintain frequencies

- ▶ Decodes codewords
- ▶ Adds new symbols ($C_{\text{new-symbol}}$)
- ▶ Only swaps when it decodes a C_{swap}



DLETDC: transmission

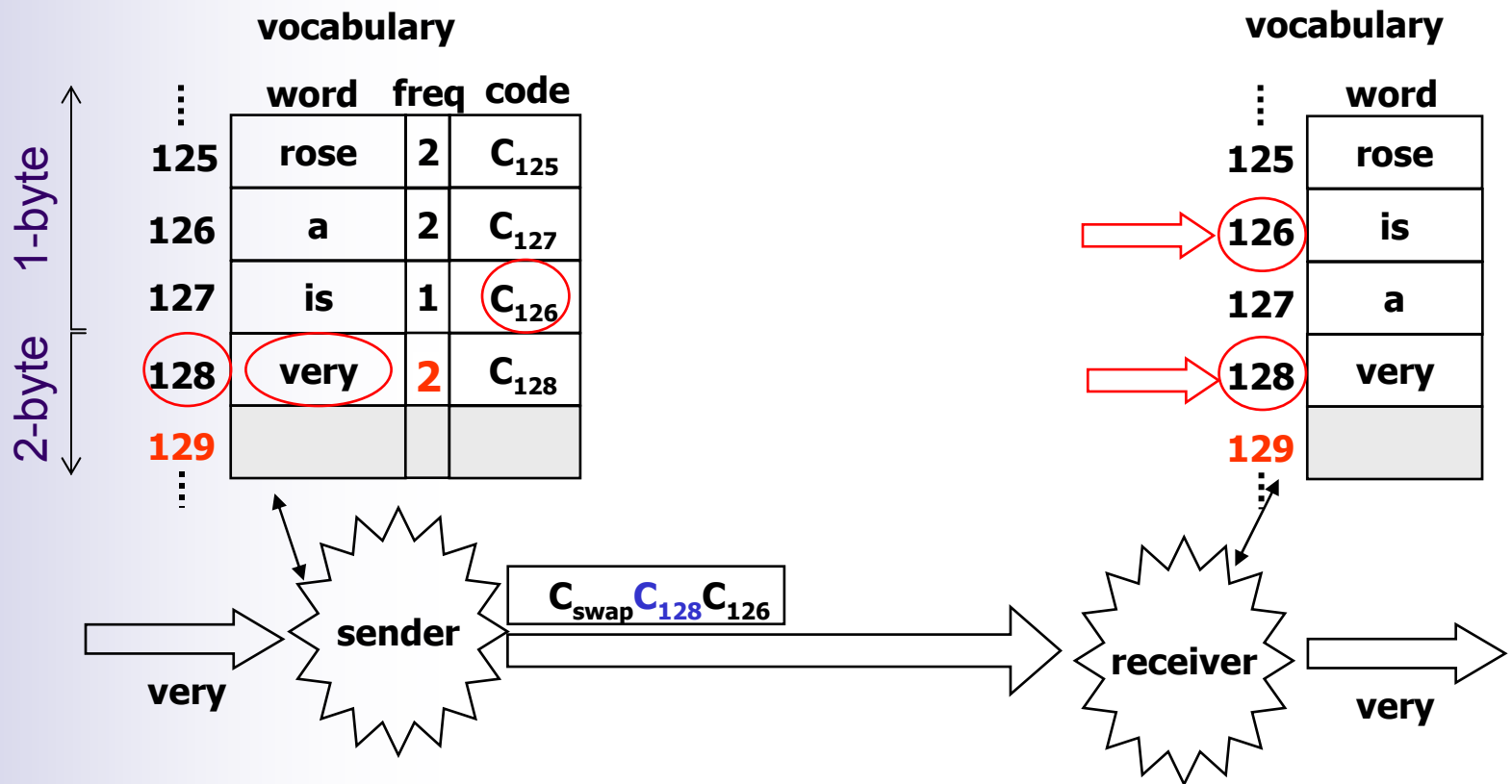
Example: ... a rose rose is a very nice one



No codeword swap needed since: "a" \leftrightarrow C_{127} and "is" \leftrightarrow C_{126}

DLETDC: transmission

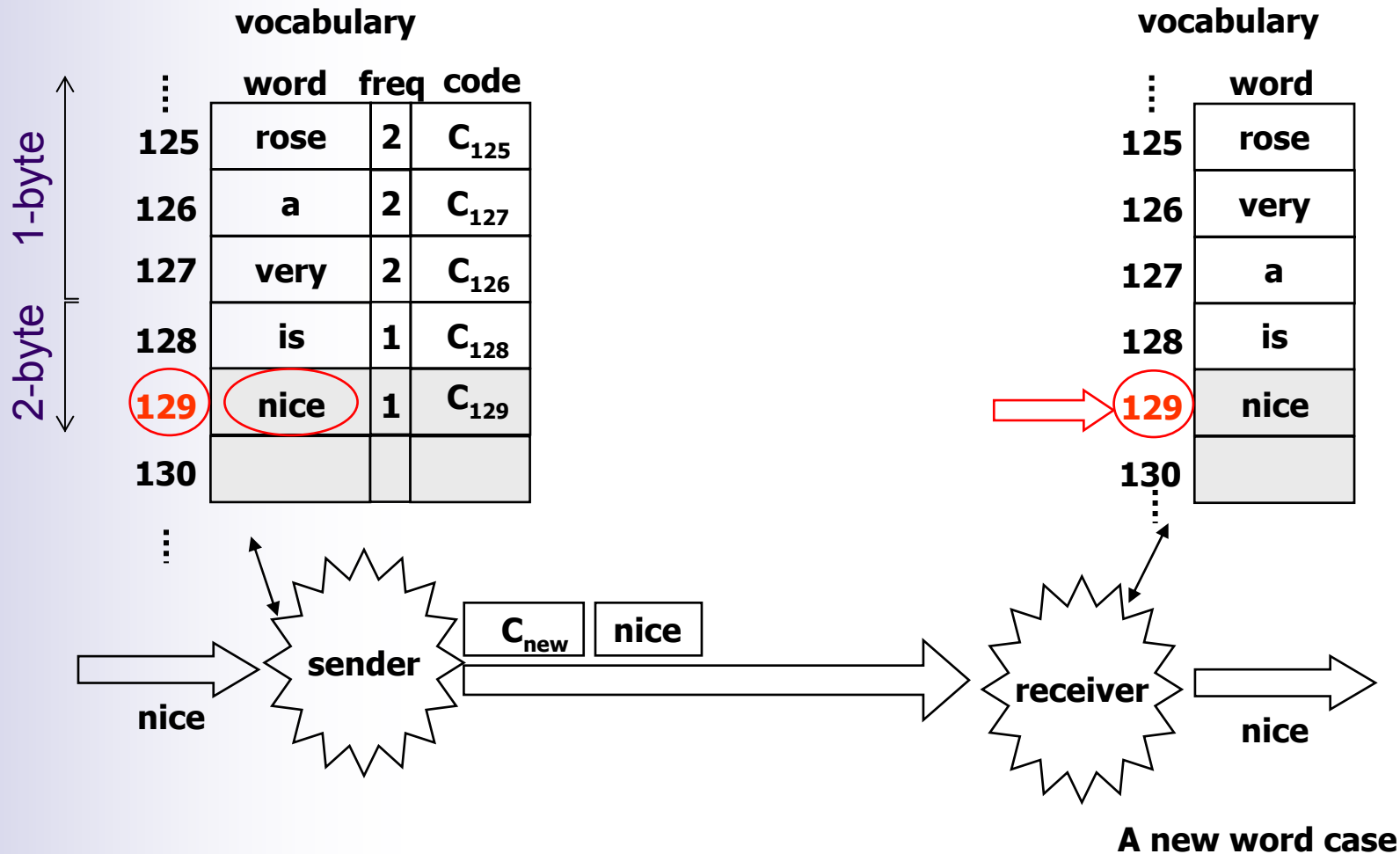
Example: ... a rose rose is a very nice one



A codeword swap is needed

DLETDC: transmission

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

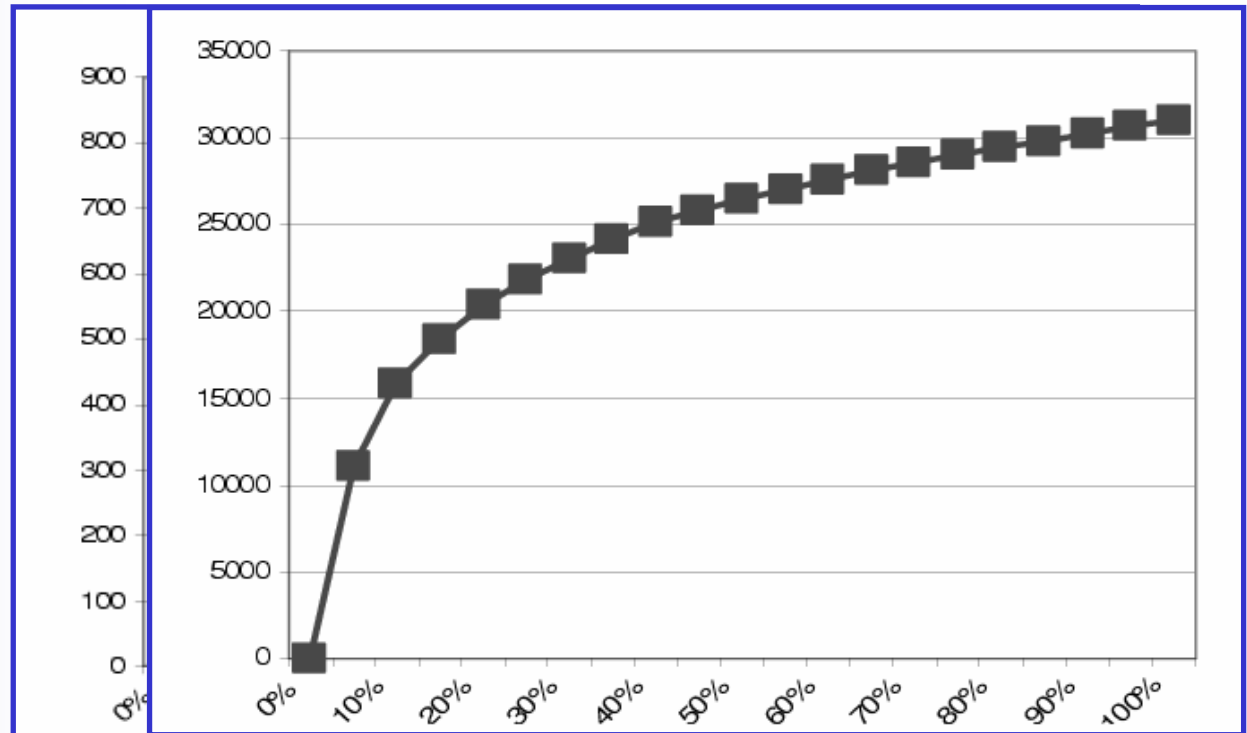


DLETDC – Evolution of swaps.

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

Between
lengths 2 and 3

ZIFF corpus
 $4,6 \times 10^7$ words
237,622 diff words



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_new
- C_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

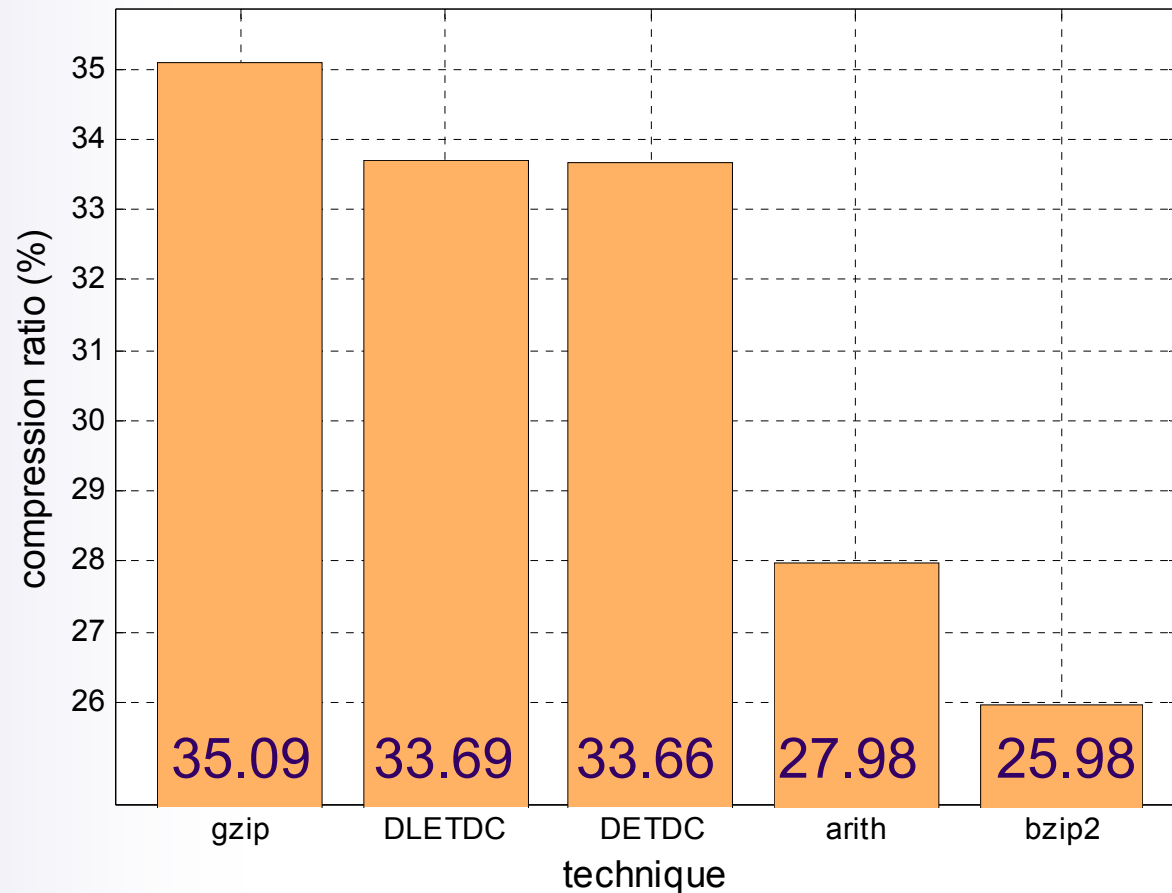
- We used some text collections from **TREC-2** & **TREC-4**, to perform the experiments

<u>CORPUS</u>	<u>size (bytes)</u>	<u># words</u>	<u>Diff. words</u>
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	53,310,620	260,141

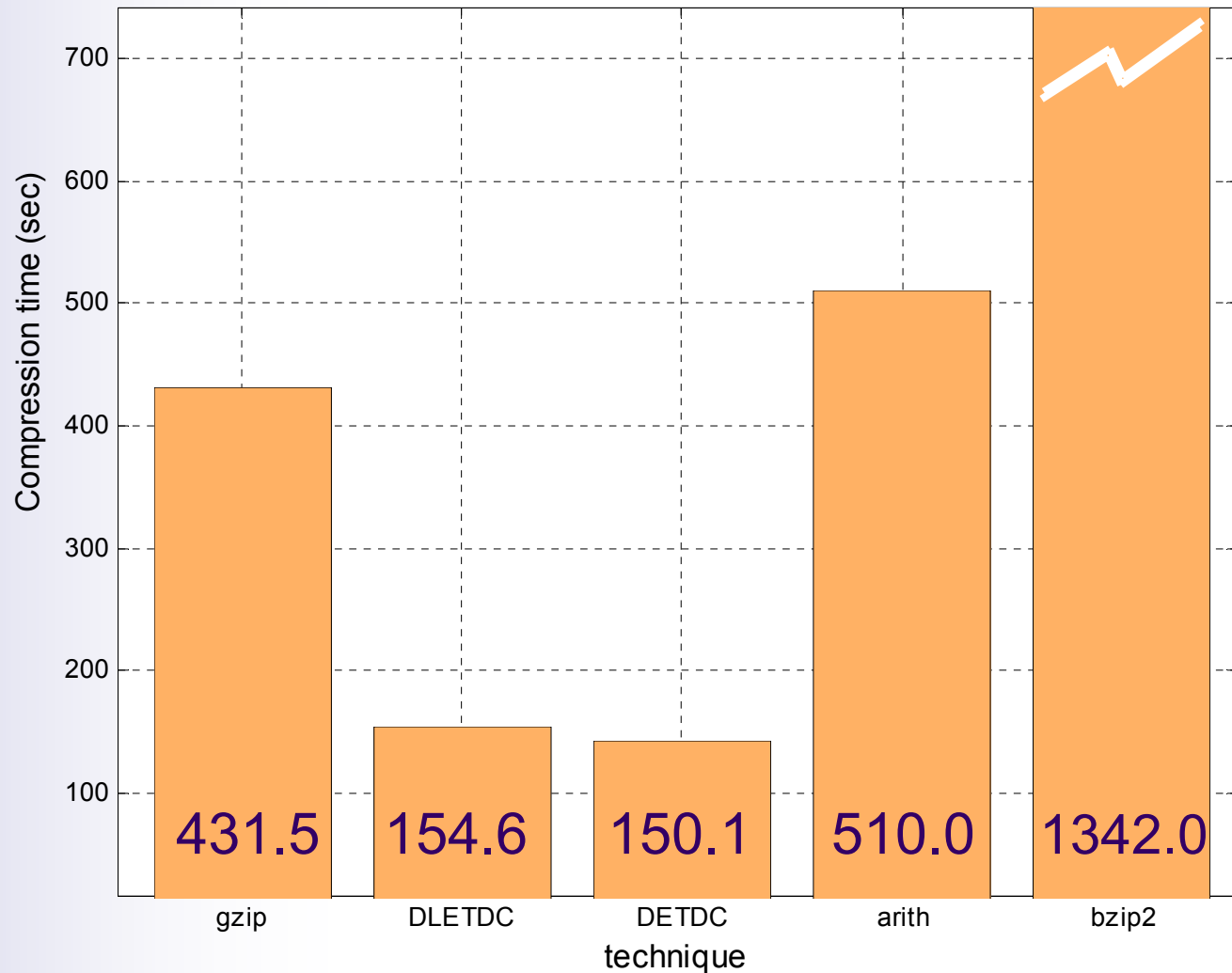
- Dual Intel Pentium-III 800 Mhz with 768Mb RAM.
 - ▶ Debian GNU/Linux (kernel 2.2.19)
 - ▶ gcc 3.3.3 20040429 and –O9 optimizations
 - ▶ Time represents CPU user-time

Empirical Results. Compression Ratio

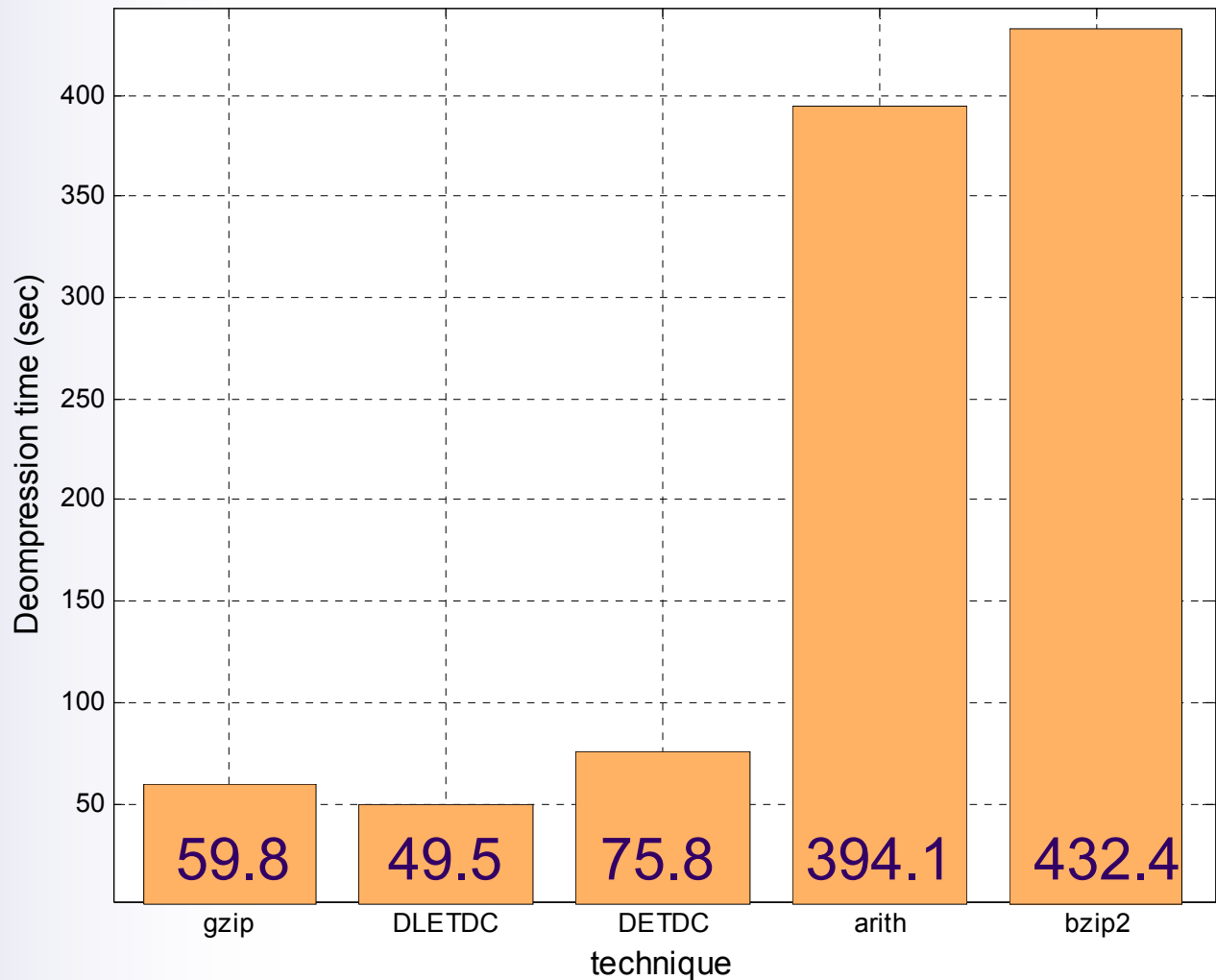
Comparison with other techniques.



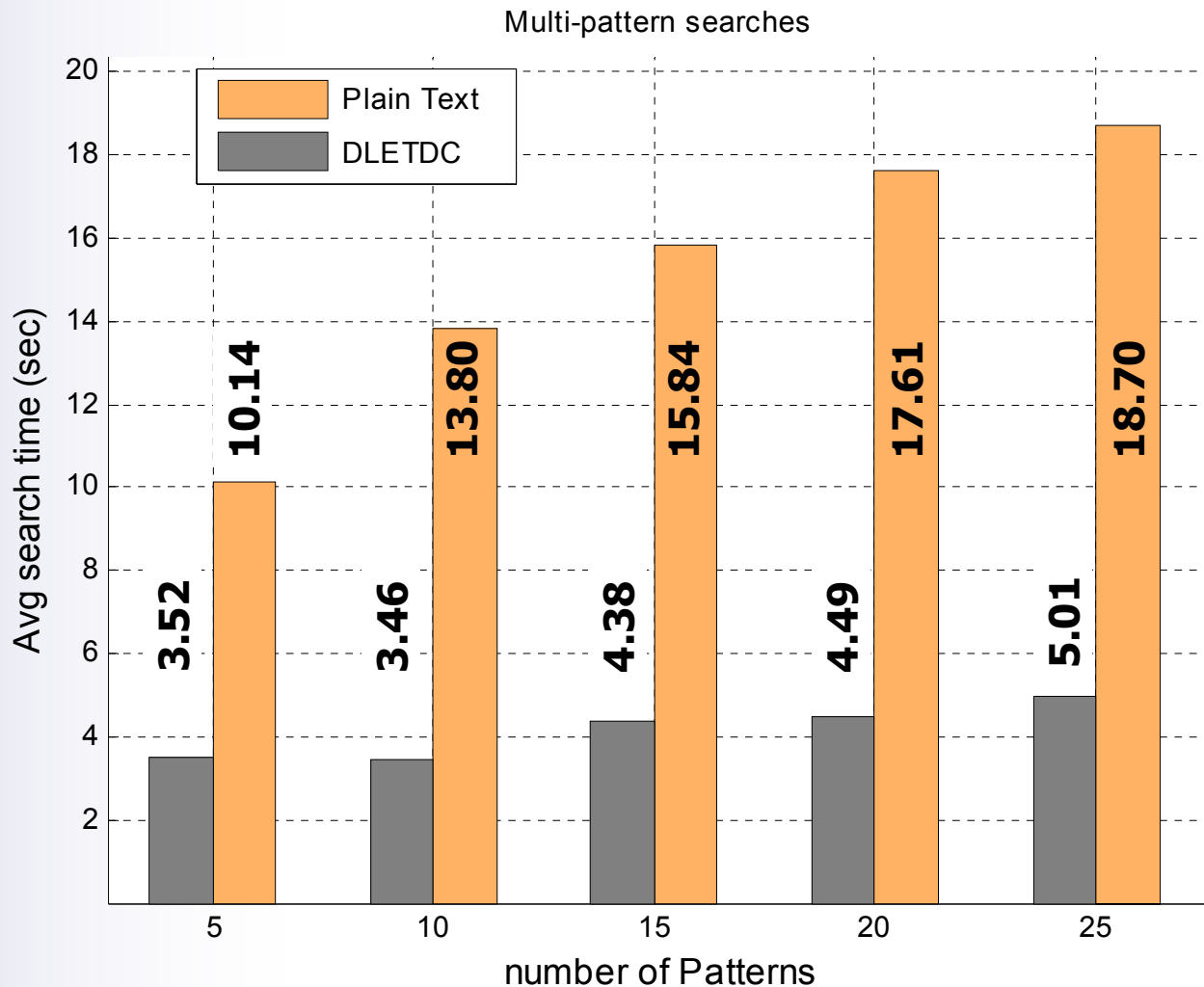
Empirical Results. Compression time



Empirical Results. Decompression time



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.