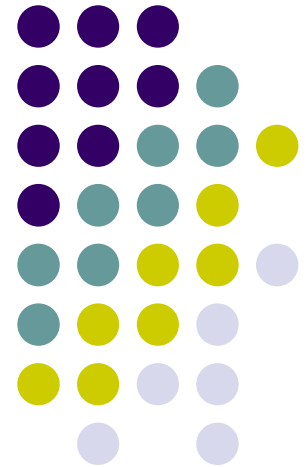
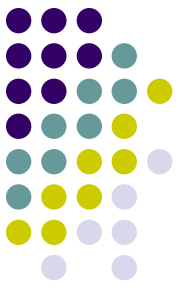


Guaranteed Synchronization of Huffman Codes

Marek Biskup
University of Warsaw, Poland

Data Compression Conference, 2008-03-27

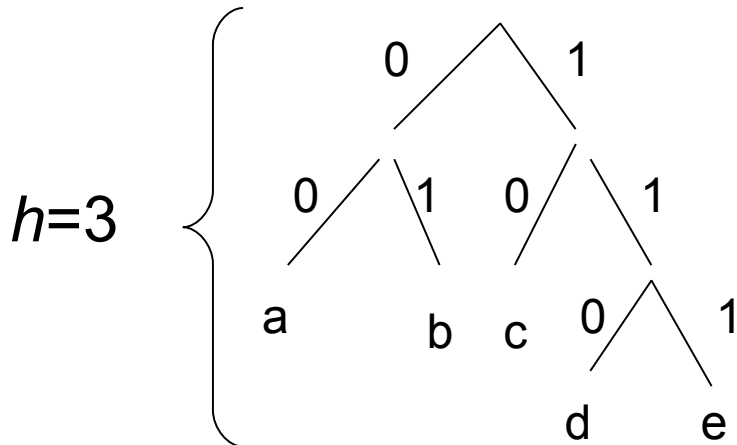




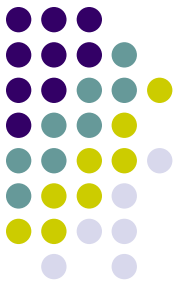
Huffman Codes

$a \rightarrow 00$
 $b \rightarrow 01$
 $c \rightarrow 10$
 $d \rightarrow 110$
 $e \rightarrow 111$

$N=5$

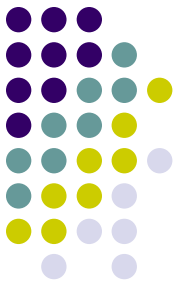


- Each letter has a corresponding binary string (its **codeword**)
- The codewords form a **complete binary tree**
- The depth of a letter depends on its probability
- The code is uniquely decodable



Why Huffman Codes?

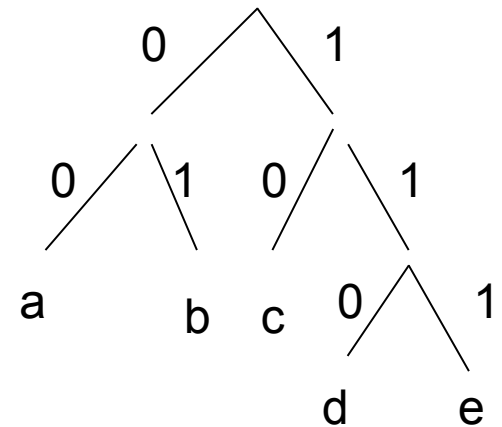
- Arithmetic coding or LZ generally perform better
 - But they need to be decoded sequentially
 - They have no easy correspondence of between source letters and bit strings
- Huffman Codes use-cases:
 - Large static Information Retrieval System
 - Short fragments decoded on demand
 - Searching in a compressed text
 - Decompressing in parallel
 - Final stage of some other compression (e.g. LZW)

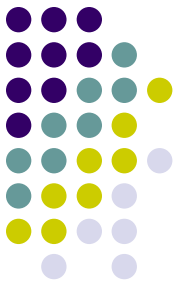


Bit corruption

- Correct:
- 0 1011110000 111011011110
- b b e a a c b c e c

- Bit error:
- 1 1011110000 111011011110





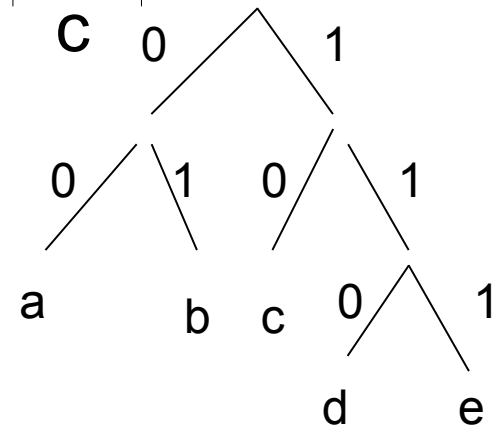
Bit corruption

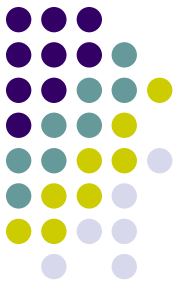
- Correct:

- 0 1 | 0 1 | 1 1 1 | 0 0 | 0 0 | 1 1 1 | 0 1 | 1 0 | 1 1 1 | 1 0
- b b e a a c b c e c

- Bit error:

- 1 1 | 0 1 | 1 1 1 | 1 0 | 0 0 | 1 1 1 | 0 1 | 1 0 | 1 1 1 | 1 0
- **d** **e** **c** **a** **b** **d** **d** e c





Bit corruption

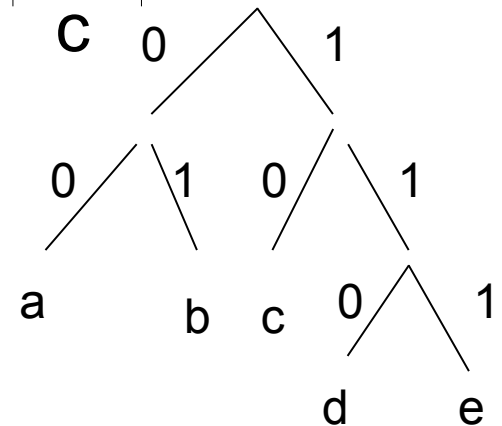
- Correct:

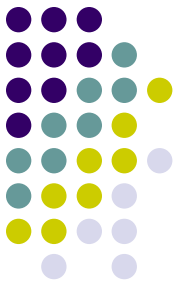
- 0 1 | 0 1 | 1 1 1 | 0 0 | 0 0 | 1 1 1 | 0 1 | 1 0 | 1 1 1 | 1 0
- b | b | e | a | a | c | b | c | e | c

- Bit error:

- 1 1 | 0 1 | 1 1 1 | 1 0 | 0 0 | 0 0 | 1 1 | 1 0 | 1 1 | 0 | 1 1 1 | 1 0
- d | e | c | a | b | d | d | e | c

Synchronization!





Bit corruption

- Correct:

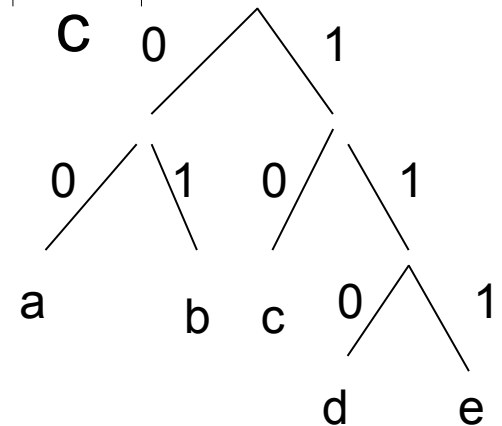
- 0 1 | 0 1 | 1 1 1 | 0 0 | 0 0 | 1 1 1 | 0 1 | 1 0 | 1 1 1 | 1 0
- b b e a a c b c e c

- Bit error:

- 1 1 | 0 1 | 1 1 1 | 1 0 | 0 0 | 0 0 | 1 1 1 | 0 1 | 1 0 | 1 1 1 | 1 0
- **d** **e** **c** **a** **b** **d** **d** e c

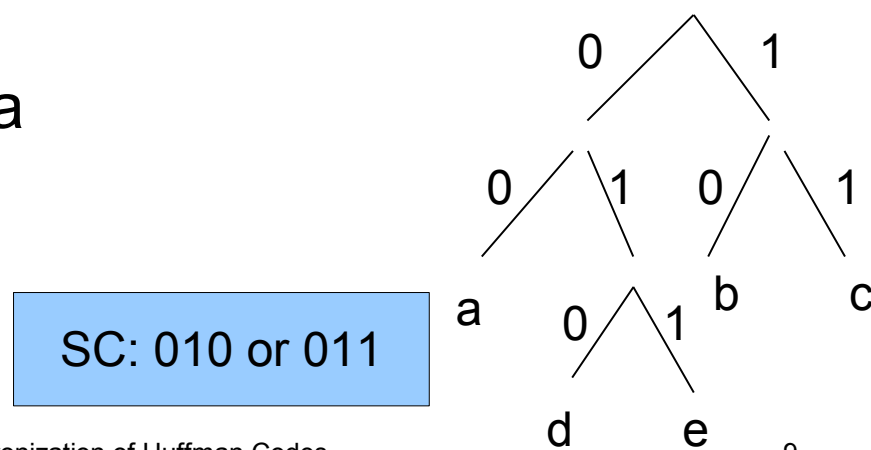
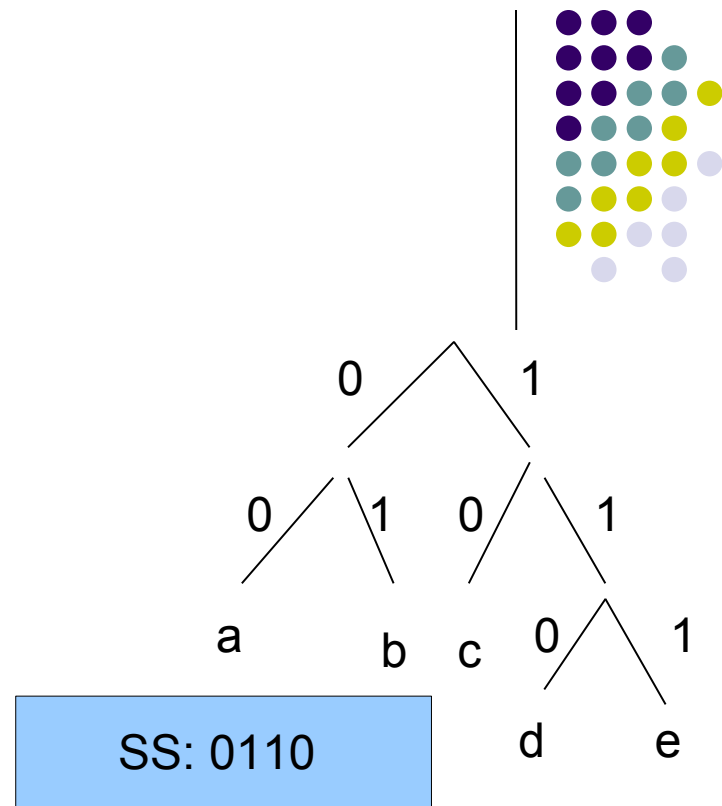
Synchronization delay

Synchronization!

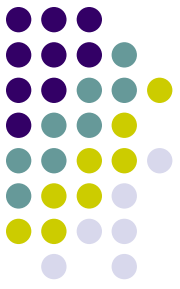


Definitions

- Huffman code is **statistically synchronizable** if the decoder always eventually synchronizes
- A **synchronizing string** (SS) is such a string that when received by the decoder always puts it into synchronization
- A **synchronizing codeword** is a SS that is a codeword

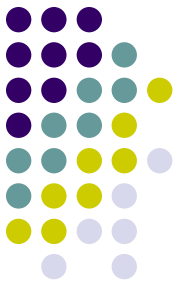


Known facts



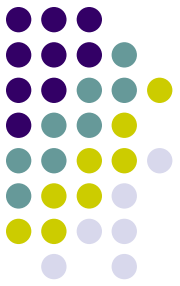
- A code is statistically synchronizable iff it has a synchronizing string (Capocelli et al., 1988)
- **Almost all** Huffman codes have a synchronizing string (Freiling et al., 2003)
- If $\text{GCD}(\text{lengths of codewords}) = 1$ then there **exists** a HC with a synchronizing string (Shützenberger, 1967)
- **Each code** can be made statistically synchronizable by adding a little **redundancy** (Capocelli et al. 1992)

Known facts (2)

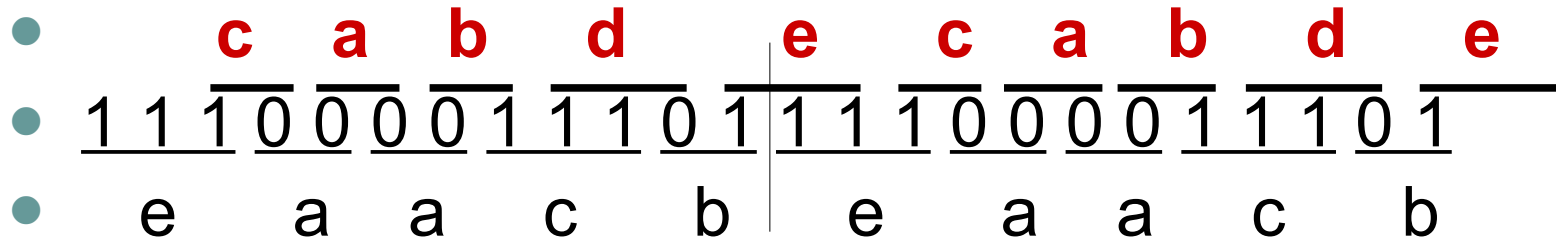


- **For some cases** there are algorithms that construct a code with a **synchronizing codeword** (e.g. Ferguson and Rabinowitz 1984)
- The average synchronization delay is **usually low** (e.g. 3.8 symbols for a 26-elem. code (Maxted and Robinson 1985))
- There are **heuristics** that give codes with low **average** synchronization delay (e.g. Zhou and Zhang, 2002)

Problems



- Synchronization is only **statistical** – no upper bound on the synchronization delay
- Example:

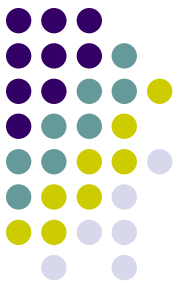


Guaranteed synchronization



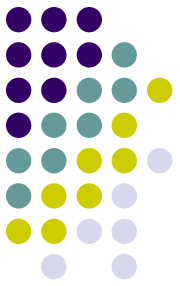
- Definition:
 - A **decoder** D_k is a decoder that starts at the k -th bit
- Goal:
 - limit the synchronization delay of any decoder to $< L$ bits
- Input: C – a code with a synchronizing **codeword** s
 - Assume that s **does not** correspond to any source letter
 - s will be a **synchronization marker** (SM)
- **Naïve approach:**
 - Insert SM between codewords in **regular intervals**

Guaranteed synchronization (2)



- **New approach:**
 - Check the synchronization delay of **each decoder** (i.e. starting from every bit) at each bit
 - Insert SM iff the delay of **some decoder** would exceed L
- The synchronization delay of decoders is computed **during encoding**
- On decoding just **skip** SM

Computing the sync. delay



- Two algorithms:
 - **Slow**: $O(h)$ per encoded codeword (worst-case)
 - Preprocessing – time: $O(N^2)$; memory: $O(N^2)$
 - Good for small trees, unusable for large trees
 - **Simple implementation**
 - **Fast**: $O(1)$ per encoded bit (worst-case)
 - Preprocessing – time: $O(N)$; memory: $O(N)$
 - **Good for any trees**

Zero-redundancy sync.



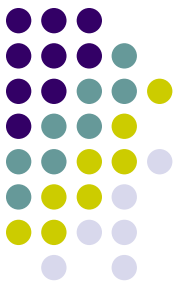
- Previously:
 - SM was a codeword that did not correspond to any letter
 - Suboptimal code: redundancy even if no SM inserted
- New setting: SM is **any synchronizing string**
 - No assumption that SM:
 - is a codeword
 - does not appear in the message

Zero-redundancy sync. (2)



- Idea:
 - The decoder **mimics the encoder** and locates the places where the SM was inserted
- Properties:
 - The redundancy depends only in the number of SM insertions
 - No redundancy if any decoder always synchronizes in L bits

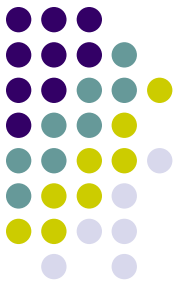
Results for guaranteed sync.



- Compression with <256 and <4096 element codes
- The delay of the order of $30|SM|$

File	size	compr.	h	$ SM $	max del. [b]	redun. [%]	A1	A2
dickens	9.7MB	5.6MB	23	10 / 5	226/ 106	0.0 /0.0	5.4	4.7
mozilla	48.8MB	38.1MB	11	13 / 7	16460/ 3112	.05 /0.02	6.8	4.8
samba	20.6MB	15.8MB	12	13 / 8	14419/ 1107	0.15 /0.05	7.2	5.0
xml	5.1MB	3.5MB	22	9 / 6	331/ 141	0.002/0.0	6.4	4.8
dickens.lzw	5.4MB	4.54MB	22	20 / 8	934/ 306	0.06 /0.003	12.8	6.0
mozilla.lzw	49.3MB	35.1MB	25	21 / 8	6351/ 1433	0.3 /0.06	8.4	4.8
samba.lzw	18.7MB	13.6MB	24	20 / 8	12485/12388	0.4 /0.06	9.3	5.0
xml.lzw	4.8MB	3.0MB	22	17 /10	2773/ 287	0.05 /0.0	9.9	4.6

Pros and cons



- Pros (applications):
 - Limited error propagation
 - Parallel decoding (cf. Klein and Wiseman, 2003)
 - Decoding of **any fragment** of an encoded message
- Cons:
 - The need to modify both the coder and the decoder
 - **Time overhead** for both coding and decoding
 - Difficult search in compressed data

Further research

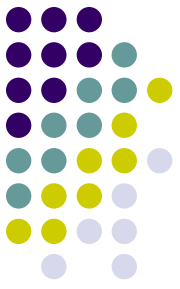


- Already done:
 - Guaranteed synchronization with **known start bit number** (ESA 2008?)
 - Explicit synchronization markers (ITW 2008)
 - Finding all the synchronizing codewords (ESA 2008?)
 - Bounds on the length of a synchr. string (MFCS 2008?)

Further research



- Already done:
 - Guaranteed synchronization with **known start bit number** (ESA 2008?)
 - Explicit synchronization markers (ITW 2008)
 - Finding all the synchronizing codewords (ESA 2008?)
 - Bounds on the length of a synchr. string (MFCS 2008?)
- To be done:
 - Applications
 - N-ary codes and other coding methods (**arithmetic?**)
 - Reduction of the time overhead
 - Search in data compressed with this method
 - Is finding the shortest synchronizing string polynomial time?



Summary

- Modification of Huffman coding
- The decoder **always resynchronizes** in L bits
 - L is a parameter
- The optimality of the code is kept
- The redundancy depends only on the number of SM insertions
- Markers are inserted only when needed