Technion

Electronics
Computers
Communications

Information in Storage Devices

049063 – EE Department, Technion

LECTURE 6: DATA COMPRESSION IN STORAGE: FIXED TO VARIABLE

Introduction

Source

Seq.	Data	Compressed data
0	00000	000
1	00001	001
2	00010	010
3	00100	011
4	01000	100
5	10000	101
6	00011	110
7	00110	111

Need **dictionary**: map compressed data to source data items

Random Source

Random Source

Seq.	Probability $\underline{p}$
0	0.4
1	0.2
2	0.1
3	0.1
4	0.05
5	0.05
6	0.05
7	0.05

Entropy of source:

$$h\left(\underline{p}\right) = \sum_{i=0}^{7} -p_i \log_2 p_i$$

Depends only on distribution $\underline{p}$, not on source sequences.

Expected compressed length

Random Source

Seq.	Probability p
0	0.4
1	0.2
2	0.1
3	0.1
4	0.05
5	0.05
6	0.05
7	0.05

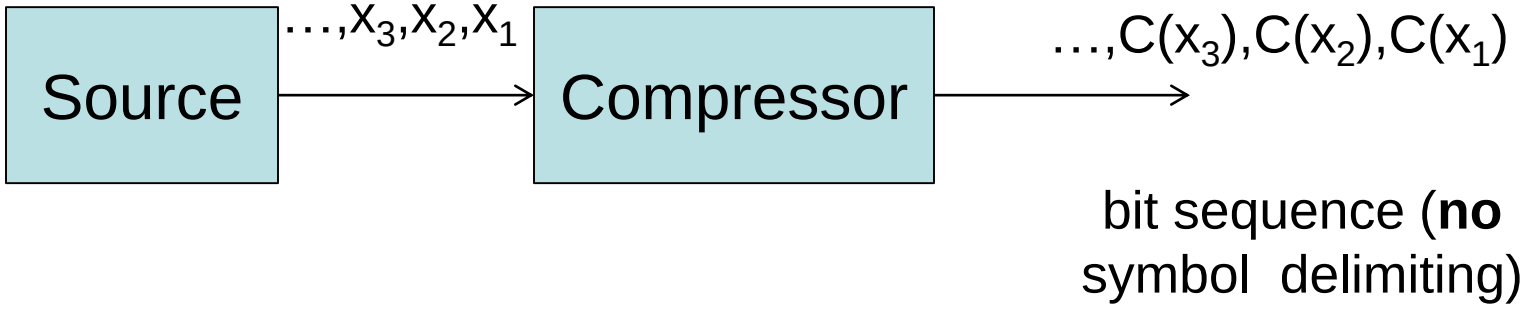
Compressed data C	Len l
0	1
10	2
1100	4
1101	4
11100	5
11101	5
11110	5
11111	5

Expected length: $L(C) = \sum_{i=0}^{7} p_i l_i$

Ex. $L(C) = 2.6$

$L(un\_compressed) = 3$

Stream Compression



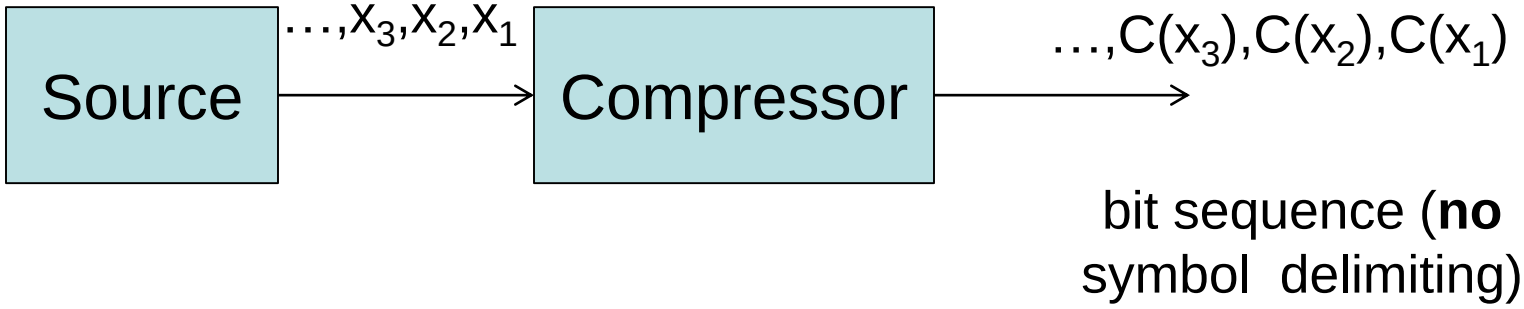
Definition – compression ratio:

For a source with an alphabet of N symbols, the *compression ratio* is defined as

$$1 \geq \frac{|C(xn), \cdots, C(x_2), C(x_1)| + |\text{dictionary}|}{n \log_2 N} \approx \frac{L(C)}{\log_2 N}$$

for n sufficiently large

Uniquely Decodable



Definition – uniquely decodable code:

C is *uniquely decodable* if every bit sequence can be obtained by **at most** one sequence of source symbols.

Theorem:

Every uniquely decodable code satisfies $L(C) \geq h\left(p\right)$.

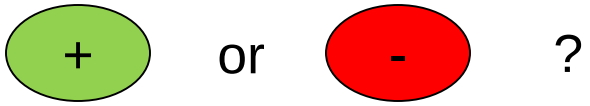
Main proof tool: Kraft's inequality (next).

Kraft's Inequality

Lemma (Kraft's Inequality):

The coded lengths of every uniquely decodable code satisfy

$$\sum_{i=0}^{N-1} 2^{-l_i} \leq 1$$



Negative result: lengths cannot be too small.

Prefix Codes

Definition – prefix code:

C is a *prefix code* if for every i, $C(x_i)$ is not a prefix (start) of any $C(x_j)$, $i \neq j$.

Ex. 10 is prefix of 1011

$\Rightarrow$

Prefix codes: can decode from left to right symbol after symbol (instantaneous codes)

prefix code uniquely decodable code

$\not\Leftarrow$

Counter example: $C(x_1)=0$, $C(x_2)=01$

0000101001010 000010101010

$x_1x_1x_1x_2x_2x_1x_2x_2x_1$

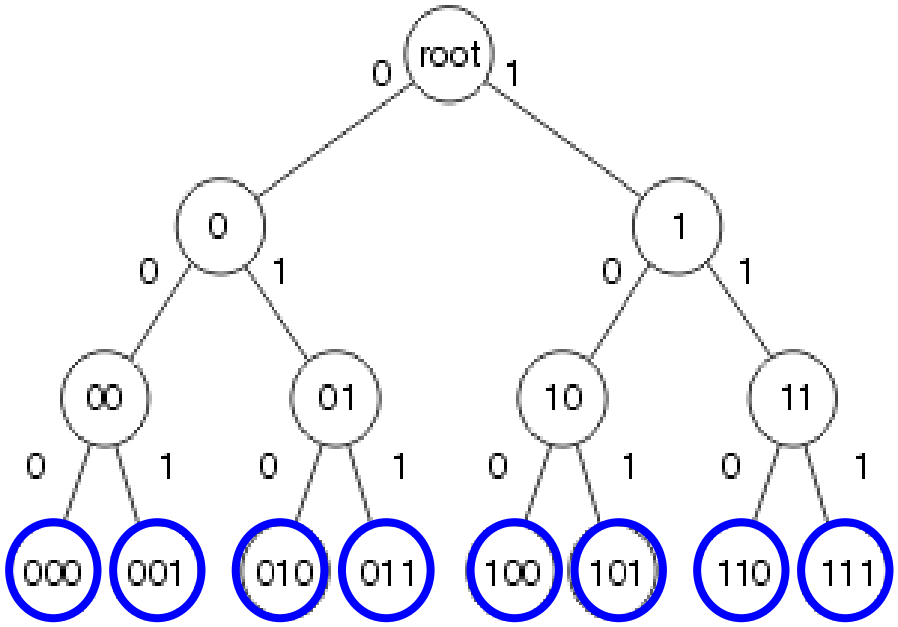
The miracle of data compression

Lemma (existence of prefix codes):

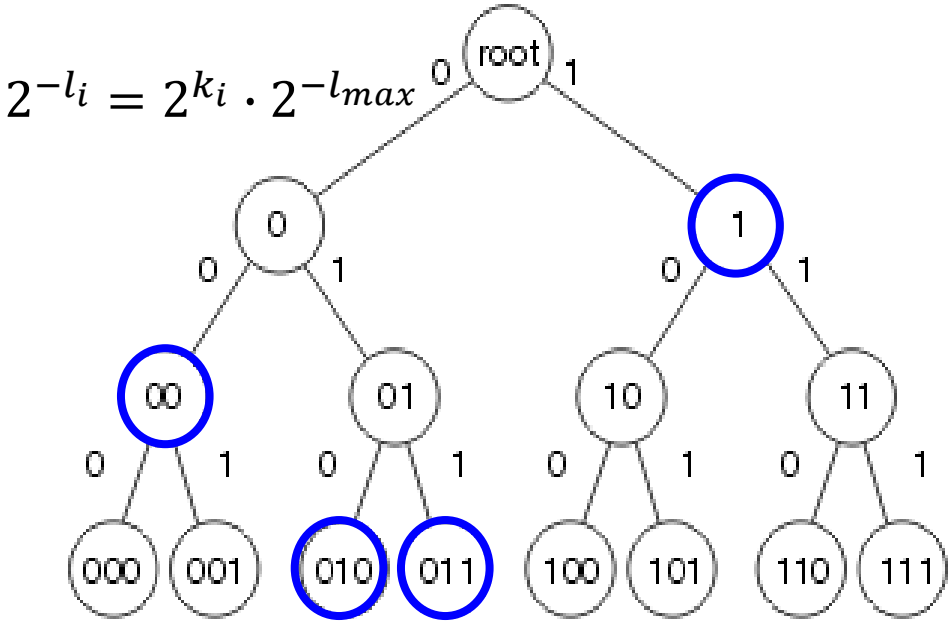
A prefix code exists if

$$\sum_{i=0}^{N-1} 2^{-l_i} \leq 1$$

Proof idea: tree depth=$\max_i(l_i)$



$$2^{-l_i} = 2^{k_i} \cdot 2^{-l_{max}}$$



Prefix codes are sufficient

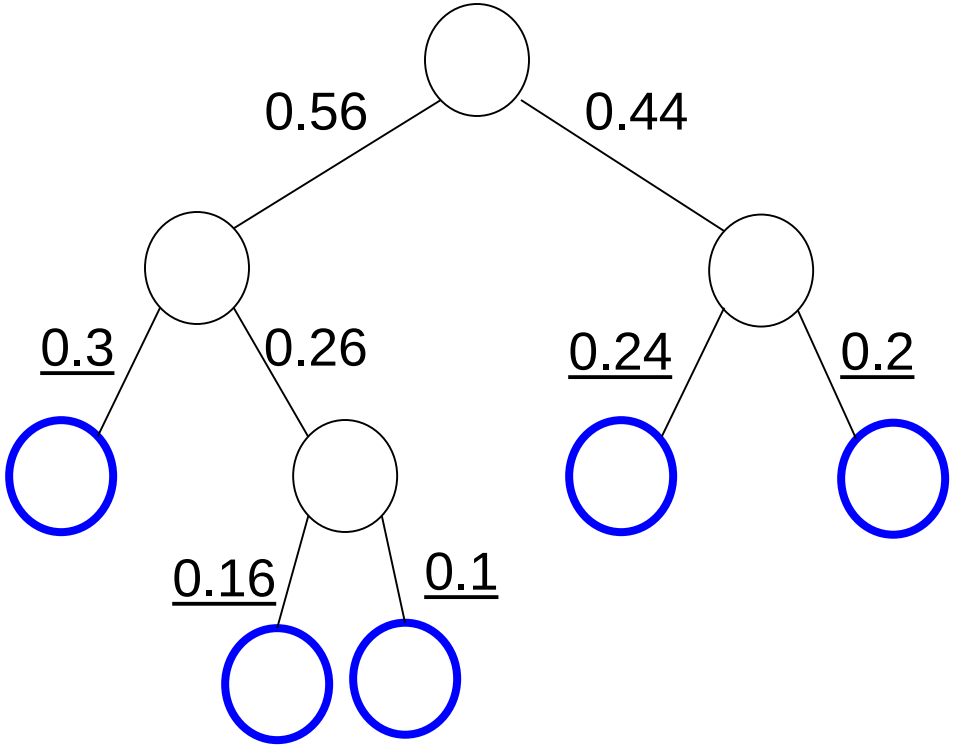
Lemma (existence) + Kraft ⇒

prefix codes are sufficient for optimal compression

Huffman Coding ['52]

0.3	0.3	0.44	0.56
0.24	0.26	0.3	0.44
0.2	0.24	0.26	
0.16	0.2		
0.1			

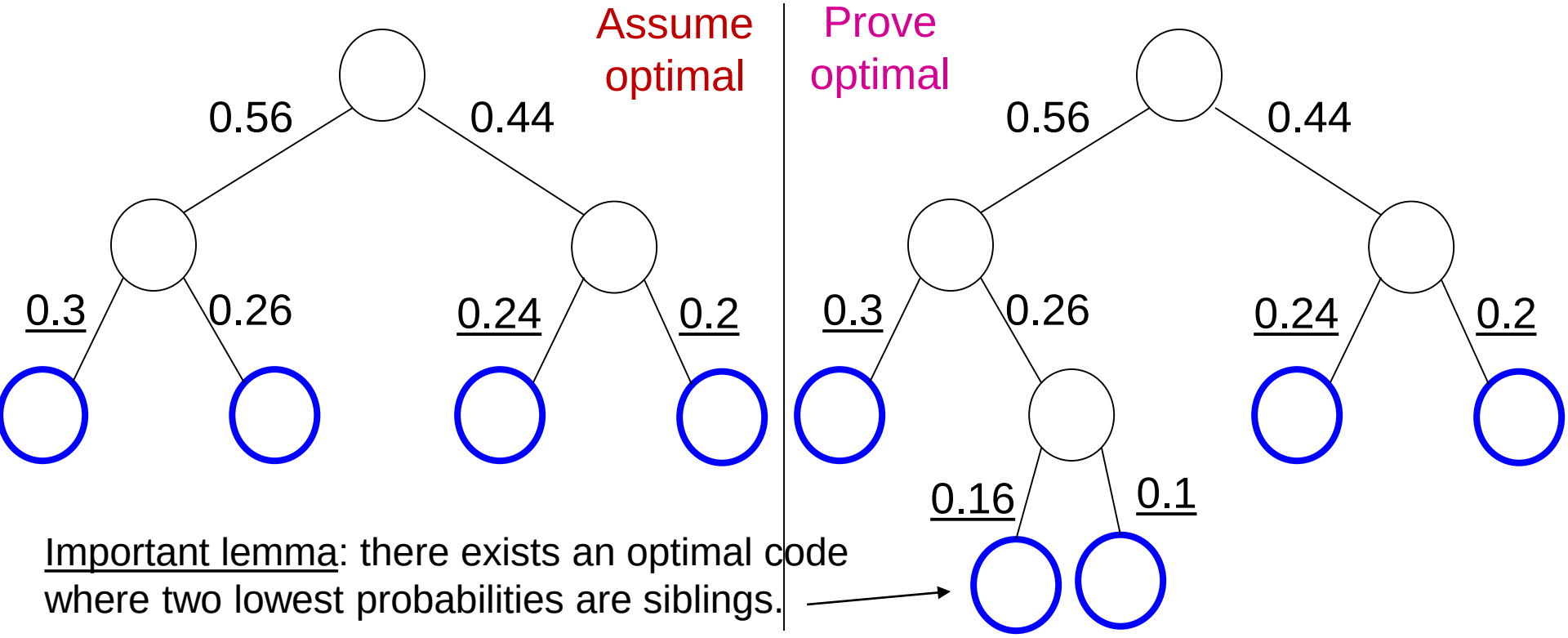
	code
0.3	00
0.24	10
0.2	11
0.16	010
0.1	011



Huffman codes are optimal

Theorem:
Given source distribution p, Huffman coding minimizes $L(C)$.

Proof idea: induction on alphabet size

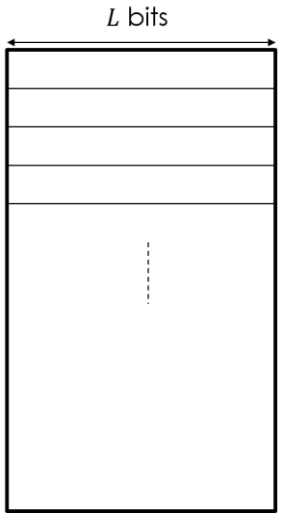


Important lemma: there exists an optimal code where two lowest probabilities are siblings.

Huffman properties

- Optimal prefix code (given alphabet)
- Up to 1 bit from entropy $h\left(\underline{p}\right)$
- Exactly $h\left(\underline{p}\right)$ for dyadic distributions
- In general: increase N to achieve optimality

Fixed-Width Compression



When Huffman codes are not the best answer

A well-known question: data compression

Question:

Given a source with n output elements, with p_i, $i \in \{1, \ldots, n\}$ the probability of element i.

How should the source elements be represented?

Example source

Elem #	Probability
1	0.4
2	0.4
3	0.08
4	0.02
5	0.02
6	0.02
7	0.02
8	0.02
9	0.01
10	0.01

A well-known answer: Huffman coding

Algorithm+Theorem [Huffman'52]:
An algorithm finding an optimal uniquely-decodable code (variable length).

Algorithm finds l_i, $i \in \{1, \dots, n\}$ that minimize

$$L(C) = \sum_{i=1}^{n} p_i l_i$$

among all uniquely decodable codes.

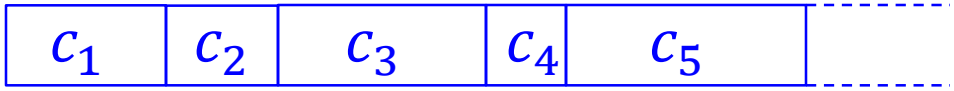
Example Huffman code

Elem #	Prob.	Compressed codeword	Len l
1	0.4	0	1
2	0.4	10	2
3	0.08	1100	4
4	0.02	11010	5
5	0.02	11011	5
6	0.02	11100	5
7	0.02	11101	5
8	0.02	11110	5
9	0.01	111110	6
10	0.01	111111	6

$L(C) = 2.14\ [bits]$

When is Huffman the correct answer?

- A limitless stream
  - minimum expected length of coded stream

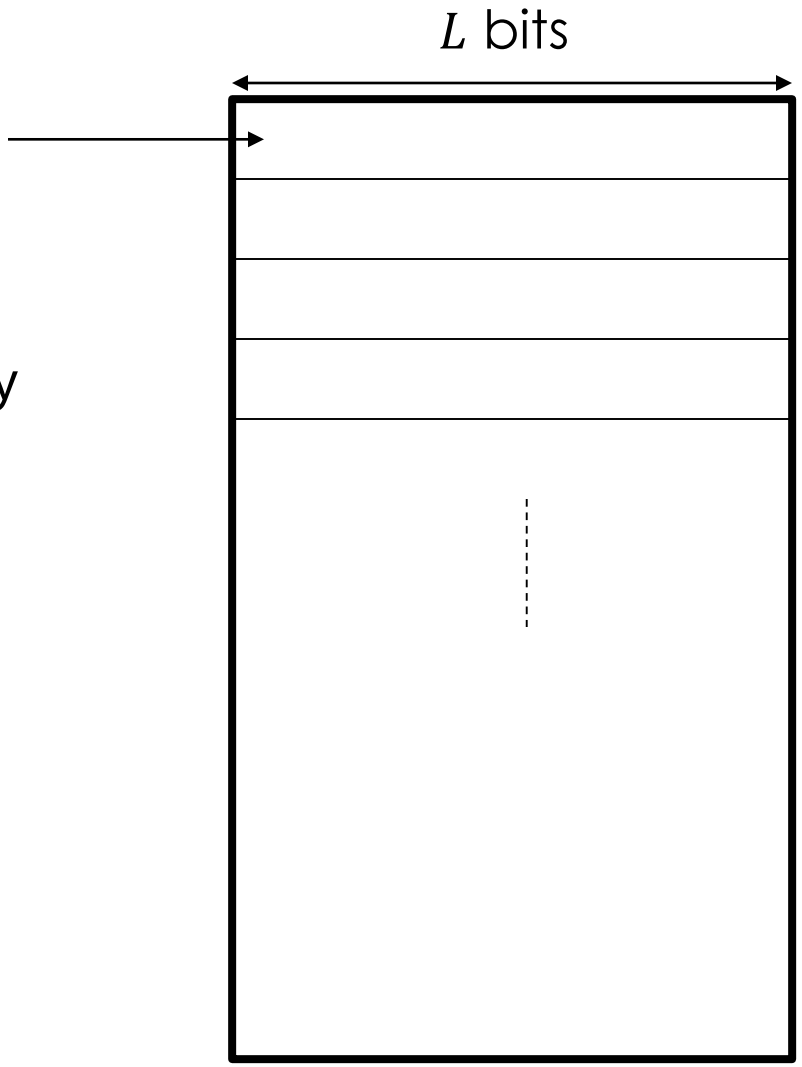


  - prefix property → instantaneous decoding

Memories with Fixed Width

Setup:

A pair of elements in each word



Motivation:

Large tables in fast expensive memory

- routers/switches
- machine learning
  - (name , attribute)
  - (x-coord , y-coord)
  - etc.

O. Rottenstreich and Y. Cassuto, Optimal Compression for Two-Field Entries in Fixed-Width Memories, IEEE Transactions on Information Theory, to appear.

Fixed-Width Compression

Problem:

Given a source with probabilities p_i, $i \in \{1, \ldots, n\}$.

Find a code that maximizes the probability of the encoder fitting 2 elements in a word of L bits.

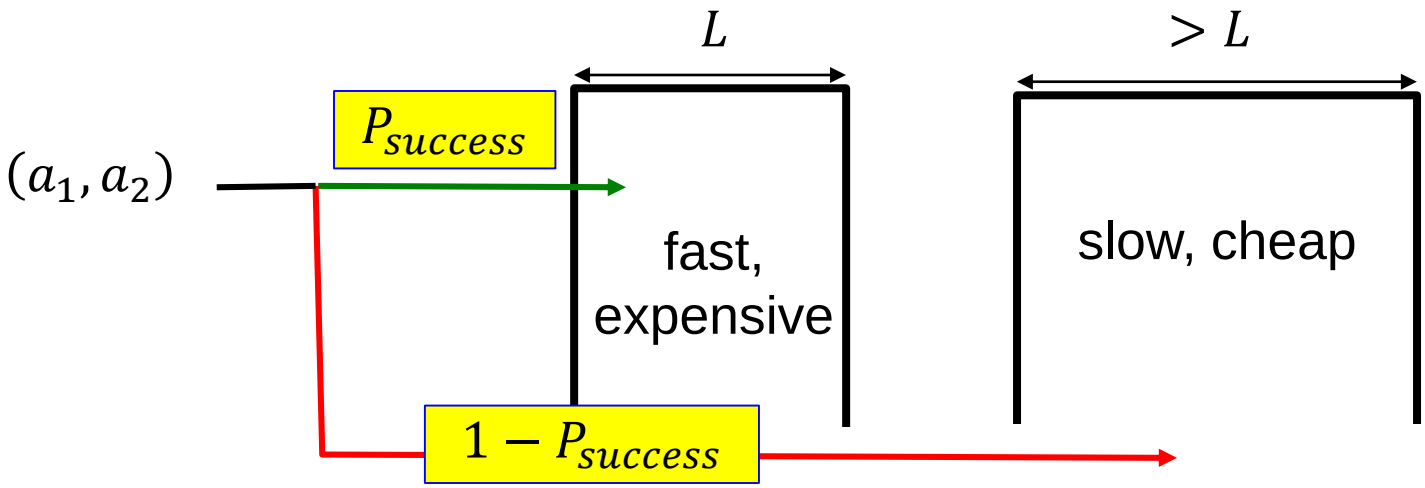
Fixed-Width Compression

Problem:

Given a source with probabilities p_i, $i \in \{1, \ldots, n\}$.
Find a code that maximizes the probability $P_{success}$ of the encoder fitting 2 elements in a word of L bits.

Idea:

- Encoding failure → write in slower memory
- Maximize access to fast memory



Operational model

Source element distribution: $(S,P) = ((s_1, \ldots, s_n), (p_1, \ldots, p_n))$
ordered $p_1 \geq p_2 \geq \cdots \geq p_n$

Pair encoder:
$\Sigma: (S,S) \rightarrow \{0,1\}^L \cup \perp$
$\perp$: encoding failure
$P_{success}(P,\Sigma) \triangleq \Pr(\Sigma(a_1,a_2) \neq \perp)$

Pair decoder:
$\Pi: \{0,1\}^L \rightarrow (S,S)$
such that $\Pi(b) = (a_1,a_2)$ if $\Sigma(a_1,a_2) = b \neq \perp$

Lossless compression

We assume in the lecture that both fields have the same (S,P),
but the scheme works for $[(S_1,P_1),(S_2,P_2)]$ as well.

A trivial but undesired solution

Product distribution:

$(S^2, P^2) = \left((s_1, s_1), (s_1, s_2), \ldots, (s_n, s_n)\right), (p_1^2, p_1 p_2, \ldots, p_n^2)\right)$

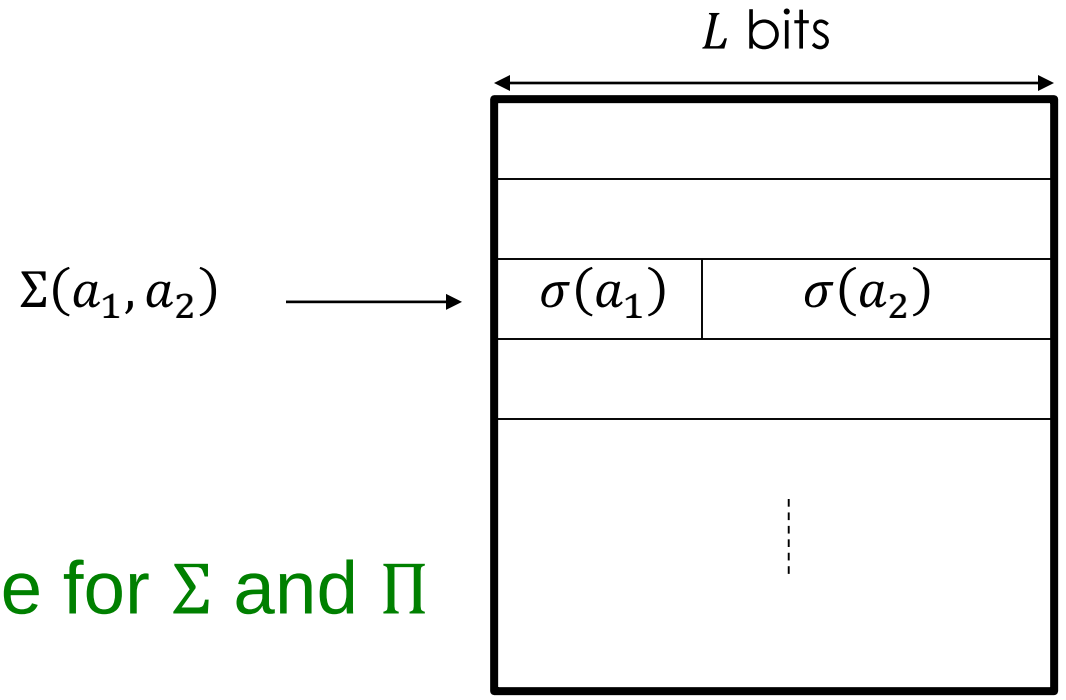
Take top 2^L pairs and assign each a codeword of length L

Not applicable: $\Theta(n^2)$ dictionary size!

Formal problem definition

Optimization problem:

Given L and $(S, P) = ((s_1, \dots, s_n), (p_1, \dots, p_n))$,
find a prefix code σ for n elements with maximal $P_{success}$.



- $O(n)$ dictionary size for Σ and Π

Huffman is not optimal

Source

L=6

Elem #	Prob.
1	0.4
2	0.4
3	0.08
4	0.02
5	0.02
6	0.02
7	0.02
8	0.02
9	0.01
10	0.01

Huffman

Compressed data C	Len l
0	1
10	2
1100	4
11010	5
11011	5
11100	5
11101	5
11110	5
111110	6
111111	6

$Psuccess = 0.848$

Compressed data C	Len l
00	2
01	2
1000	4
1001	4
1010	4
1011	4
1100	4
1101	4
1110	4
1111	4

$Psuccess = 0.96$

~factor 3.8 faster

Without compression: need L=8

Solution tools: Prefix Codes

- Any prefix code satisfies Kraft's inequality
  - $\sum_{i=1}^{n} 2^{-l_i} \leq 1$
- Define weight of length-l codeword as
  - $\frac{2^{-l}}{2^{-L}} = 2^{L-l}$

=> weights are integers, and

- $\sum_{i=1}^{n} \text{weight}(\sigma(s_i)) \leq 2^{L}$

Kraft inequality and prefix codes

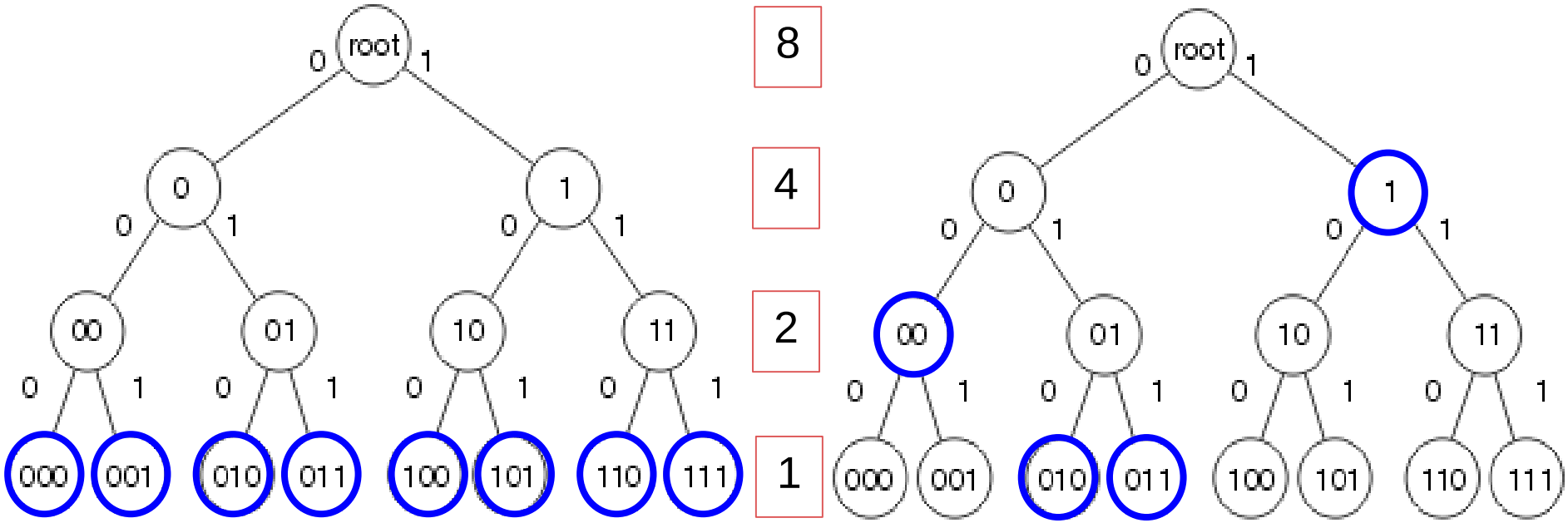
Lemma (existence of prefix codes):

A prefix code exists iff

$$\sum_{i=1}^{n} 2^{-l_i} \leq 1$$

$$2^{-l_i} = wt \cdot 2^{-L}$$

Proof idea: tree depth=L



Optimization problem

Optimization problem:

Given L and $(S, P) = ((s_1, \dots, s_n), (p_1, \dots, p_n))$,
find lengths $(l_1, \dots, l_n)$ with total weight 2^L such that

is maximal. $$\sum_{i=1}^{n} \sum_{j=1}^{n} p_i \cdot p_j \cdot I\left[l_i + l_j \leq L\right]$$

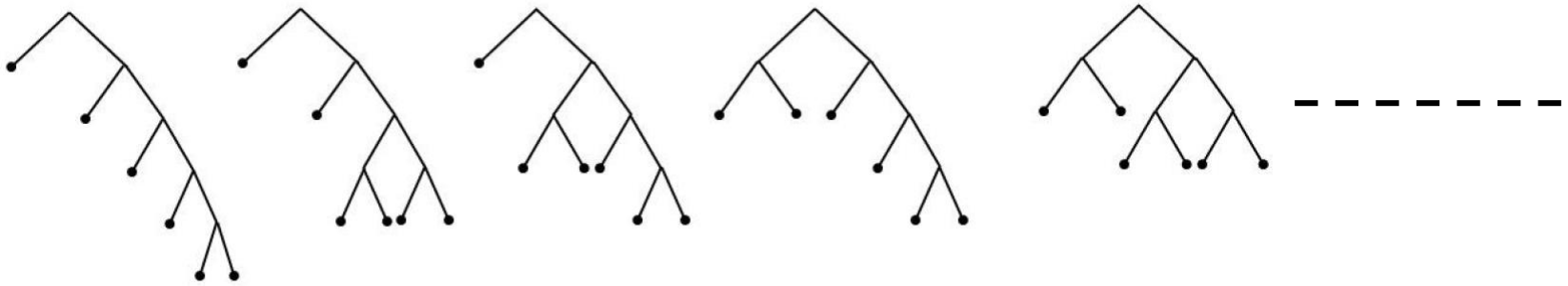
success

depends on code

Observations on the problem

- Lengths of $s_1, \ldots, s_n$ are non-decreasing
- May choose not to assign codewords to the items $s_{n'+1}, \ldots, s_n$.

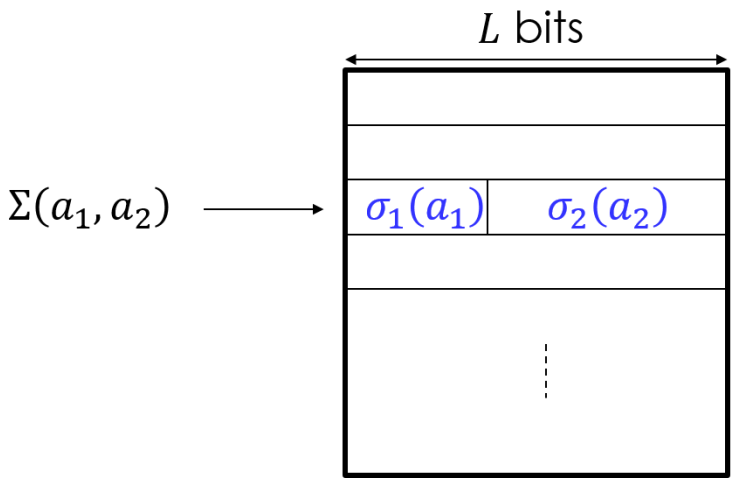
Brute-force search



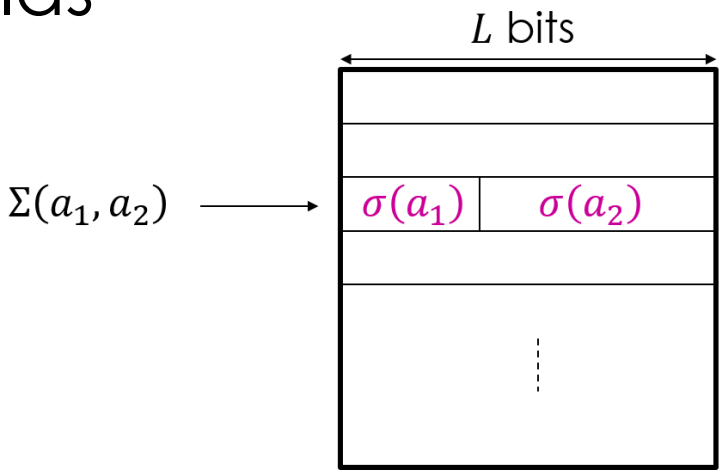
- Exhaustive enumeration of non-decreasing prefix trees
  - Exponential (in n) complexity ($\geq 2^{\frac{n}{3}-1}$ trees)

2 problem variants

1. A code for each field

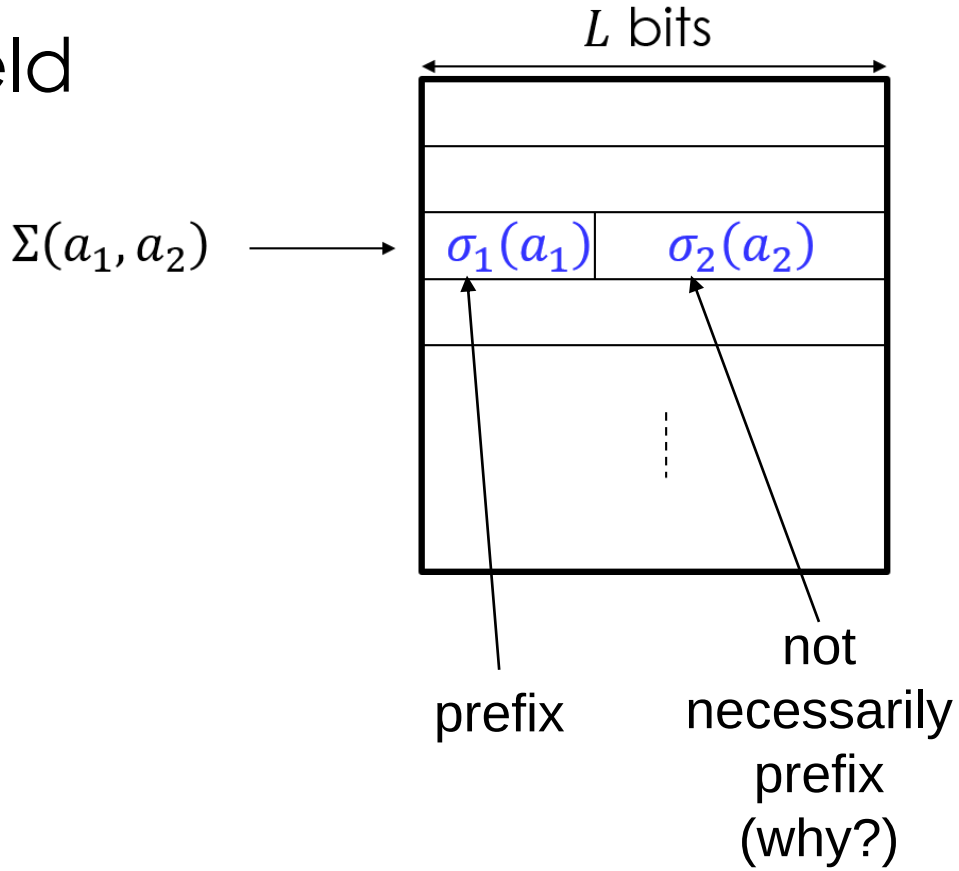


2. Same code for both fields



Variant 1

1. A code for each field
   - easier



Main ingredient: **conditionally** optimal prefix code

- Input: $(S, P), \sigma_2$
- Task: find σ_1 that maximizes $P_{success}$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} p_i \cdot p_j \cdot I\left[l_i + l_j \leq L\right]$$

$$\sum_{i=1}^{n} p_i \sum_{j=1}^{n} p_j \cdot I\left[l_i \leq L - l_j\right]$$

known from σ_2

Divide and conquer

$$k \longrightarrow \sum_{i=1}^{n} p_i \sum_{j=1}^{n} p_j \cdot I\left[l_i \le L - l_j\right]$$

- Find code for
  - $(s_1, \ldots, s_k)$ (top k elements)
  - using weight budget N

  } $OPT(k, N)$
- Optimal $P_{success} \triangleq OPT(n, 2^L)$
- We can find $OPT(k, N)$ from previous $OPT(k-1, N')$:

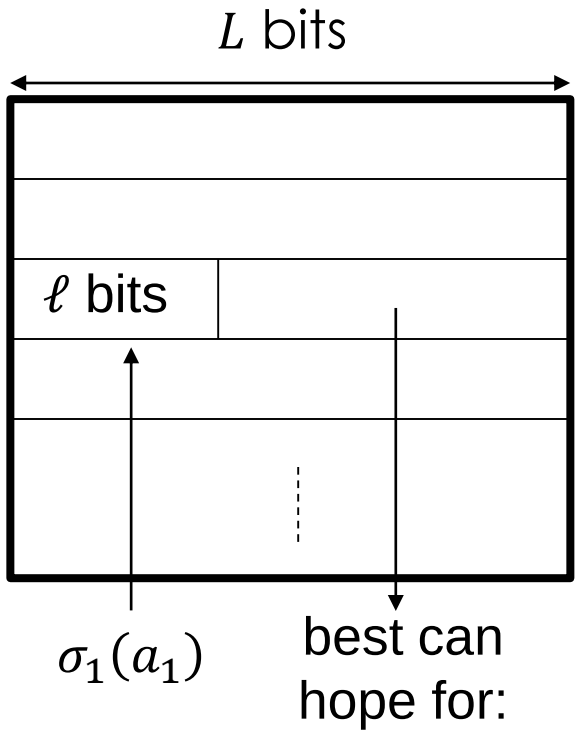
$$OPT(k, N) = \max_{\ell \in \{1,\ldots,L\}} \left[OPT(k-1, N - N_\ell) + p_k \sum_{j=1}^{n} p_j \cdot I\left[\ell \le L - l_j\right] \right]$$

But what is σ_2?

- Optimal given σ_1…

- Not necessarily prefix

$$\sigma_2(a) = \begin{cases} BIN_{L-\ell}(\text{seq}(a)) & \text{seq}(a) \le 2^{L-\ell} \\ \perp & \text{otherwise} \end{cases}$$



Variant 1 summary

$$OPT(k, N) = \max_{\ell \in \{1,\ldots,L\}} \left[OPT(k-1, N - N_\ell) + p_k \sum_{j=1}^{n} p_j \cdot I\left[j \leq 2^{L-\ell}\right] \right]$$

Polynomial upper bound on running time (loose):
$O(Ln^4)$

Variant 1 example

$L = 4$

$P_1 = (0.4, 0.3, 0.16, 0.08, 0.06)$

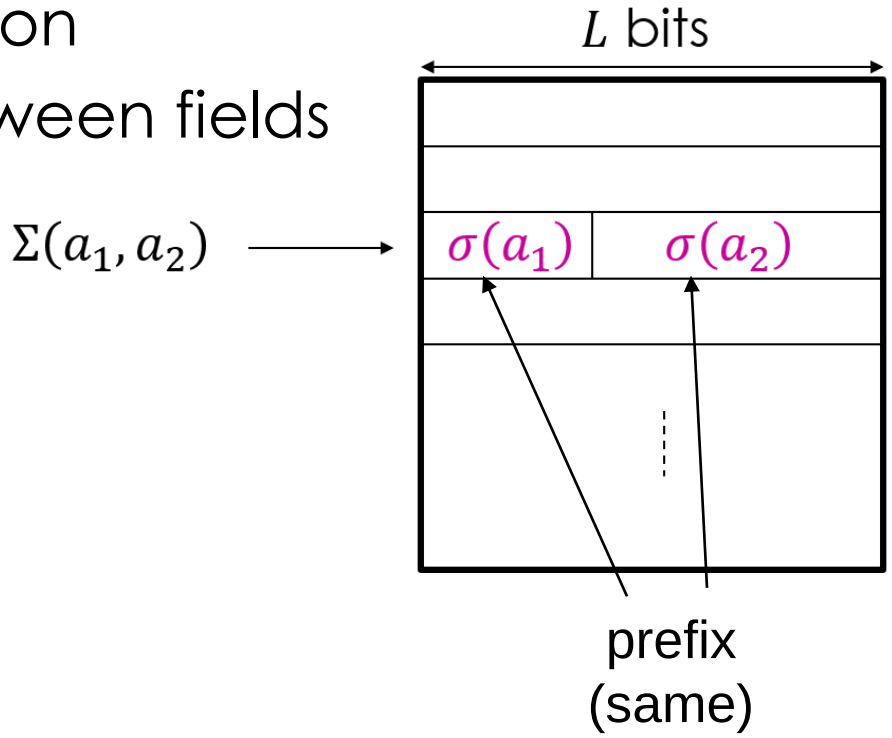
$P_2 = (0.5, 0.3, 0.2)$

$OPT(k, N)$

	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
$N=0$	0	0	0	0	0
	(-)	(-,-)	(-,-,-)	(-,-,-,-)	(-,-,-,-,-)
$N=1$	0.2	0.2	0.2	0.2	0.2
	(4)	(4,-)	(4,-,-)	(4,-,-,-)	(4,-,-,-,-)
$N=2$	0.32	0.35	0.35	0.35	0.35
	(3)	(4,4)	(4,4,-)	(4,4,-,-)	(4,4,-,-,-)
$N=3$		0.47	0.47	0.47	0.47
		(3,4)	(3,4,-)	(3,4,-,-)	(3,4,-,-,-)
$N=4$	0.4	0.56	0.56	0.56	0.56
	(2)	(3,3)	(3,3,-)	(3,3,-,-)	(3,3,-,-,-)
$N=5$			0.64	0.64	0.64
			(3,3,4)	(3,3,4,-)	(3,3,4,-,-)
$N=6$		0.64	0.688	0.688	0.688
		(2,3)	(3,3,3)	(3,3,3,-)	(3,3,3,-,-)
$N=7$			0.72	0.728	0.728
			(2,3,4)	(3,3,3,4)	(3,3,3,4,-)
$N=8$		0.7	0.768	0.768	0.768
		(2,2)	(2,3,3)	(2,3,3,-)	(2,3,3,-,-)
$N=9$			0.78	0.808	0.808
			(2,2,4)	(2,3,3,4)	(2,3,3,4,-)
$N=10$			0.828	0.832	0.838
			(2,2,3)	(2,3,3,3)	(2,3,3,4,4)
$N=11$				0.868	0.868
				(2,2,3,4)	(2,2,3,4,-)
$N=12$			0.86	0.892	0.898
			(2,2,2)	(2,2,3,3)	(2,2,3,4,4)
$N=13$				0.9	0.922
				(2,2,2,4)	(2,2,3,3,4)
$N=14$				0.924	0.94
				(2,2,2,3)	(2,2,3,3,3)
$N=15$					0.954
					(2,2,2,3,4)
$N=16$				0.94	0.972
				(2,2,2,2)	(2,2,2,3,3)

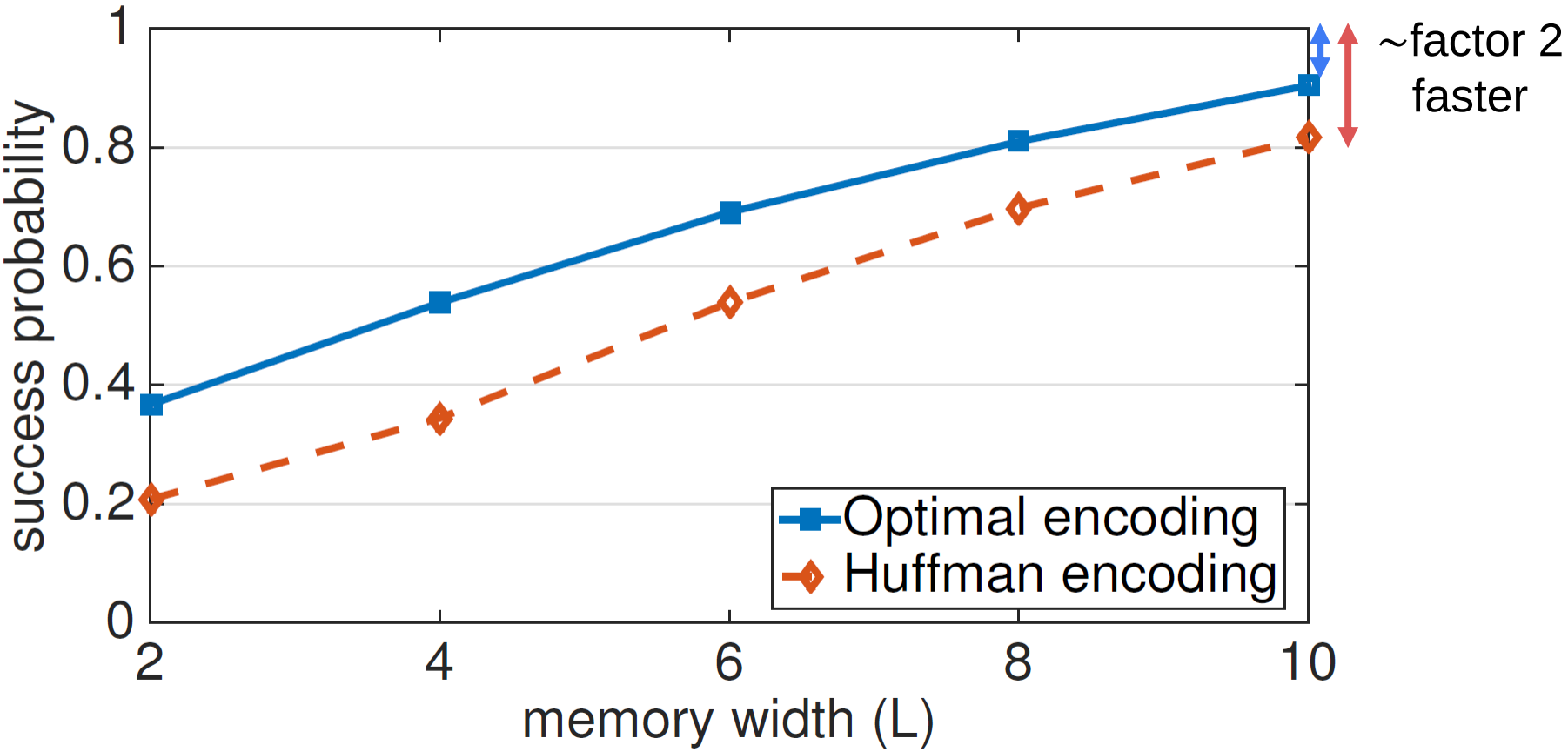
Variant 2

1. Same code for both fields
   - Simpler implementation
   - Can move items between fields



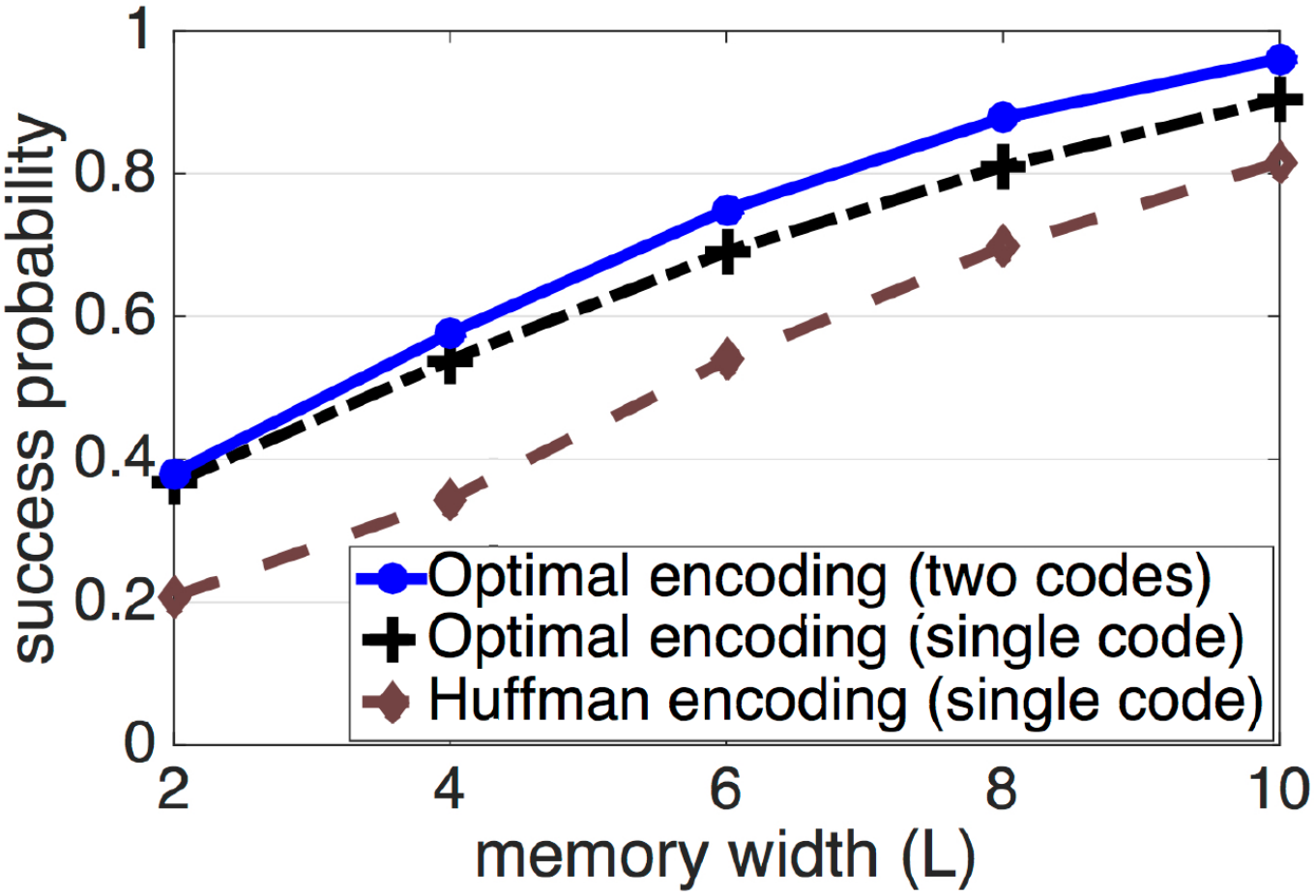
Solution: omitted

Numerical results: compare to Huffman



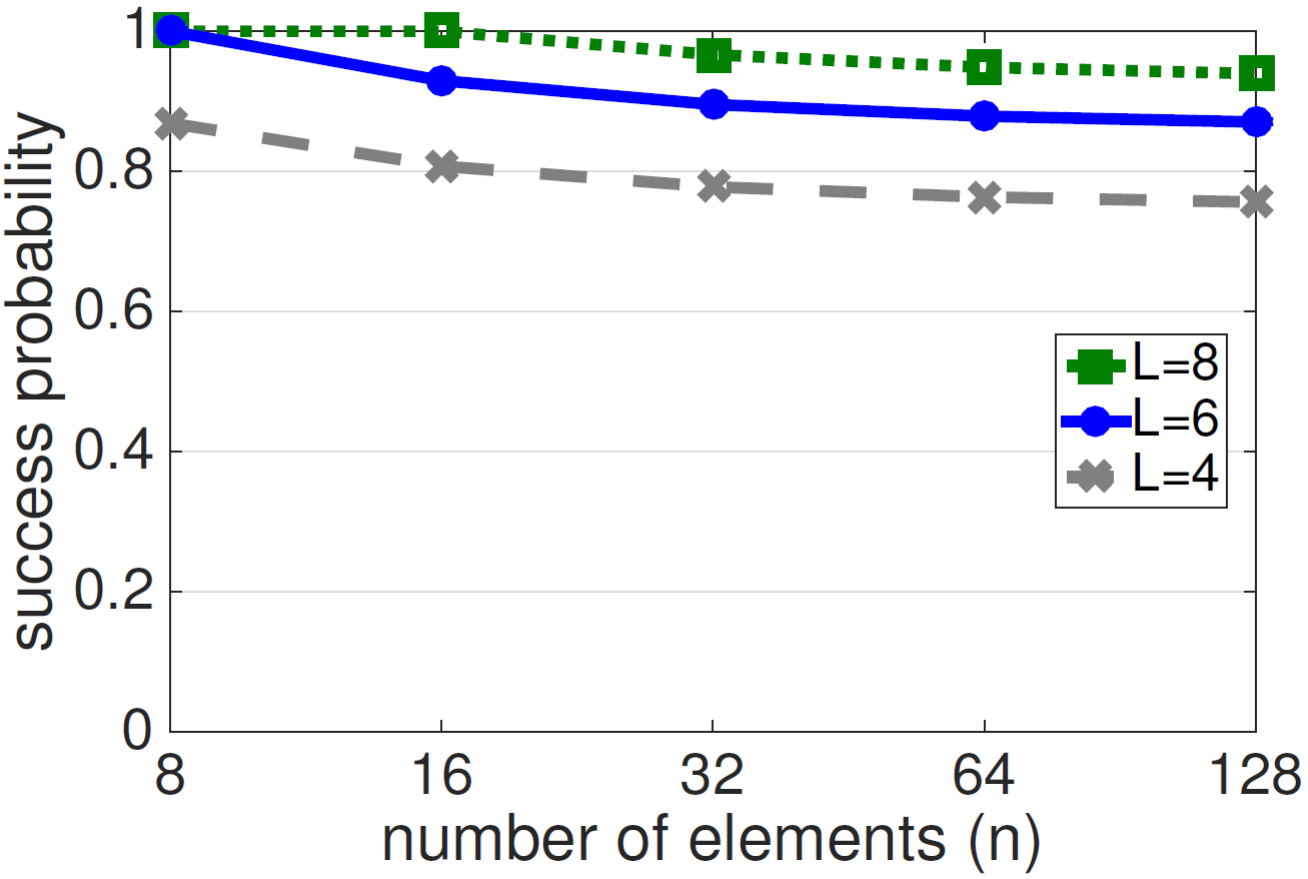
$n = 128$, Zipf $\mu = 1.6$

Two codes are even better



$n = 128$, Zipf $\mu = 1.6$

Numerical results: variable n and L



Zipf $\mu = 2$ (single code)