CS630 Graduate Algorithms

November 5, 2024

by Dora Erdos and Jeffrey Considine

- Compression part 2

slides by Dora Erdos and Jeffrey Considine, based on Kevin Wayne, Adam Smith

Compression with Independent Identically Distributed Symbols

Encode a sequence of n symbols drawn independently from the same probability distribution as random variable X…

- For large n, average bits per symbol with *any* encoding scheme is at least the entropy

$$H(X) = -\sum_{x \in X} p(x) \log_2 p(x)$$

(Shannon's Source Coding Theorem)

- If all $p(x)$ are powers of $\frac{1}{2}$, i.e. some $\frac{1}{2^k}$,

then Huffman codes achieve this limit.

Compression Topics Today

New challenges -

- Variable symbol distributions
- Optimal compression with probabilities that are not powers of $\frac{1}{2}$
- Breaking the relationship between whole bits and individual symbols

File Compression

Generic but dominant usage - make data files smaller.

- Text files are very compressible - often ~10% of original size.
- Can get ~50% just from most characters not being used.
- Rest is from dynamically recognizing repetitions within a file.
  - Not just varying probabilities but adding new symbols.

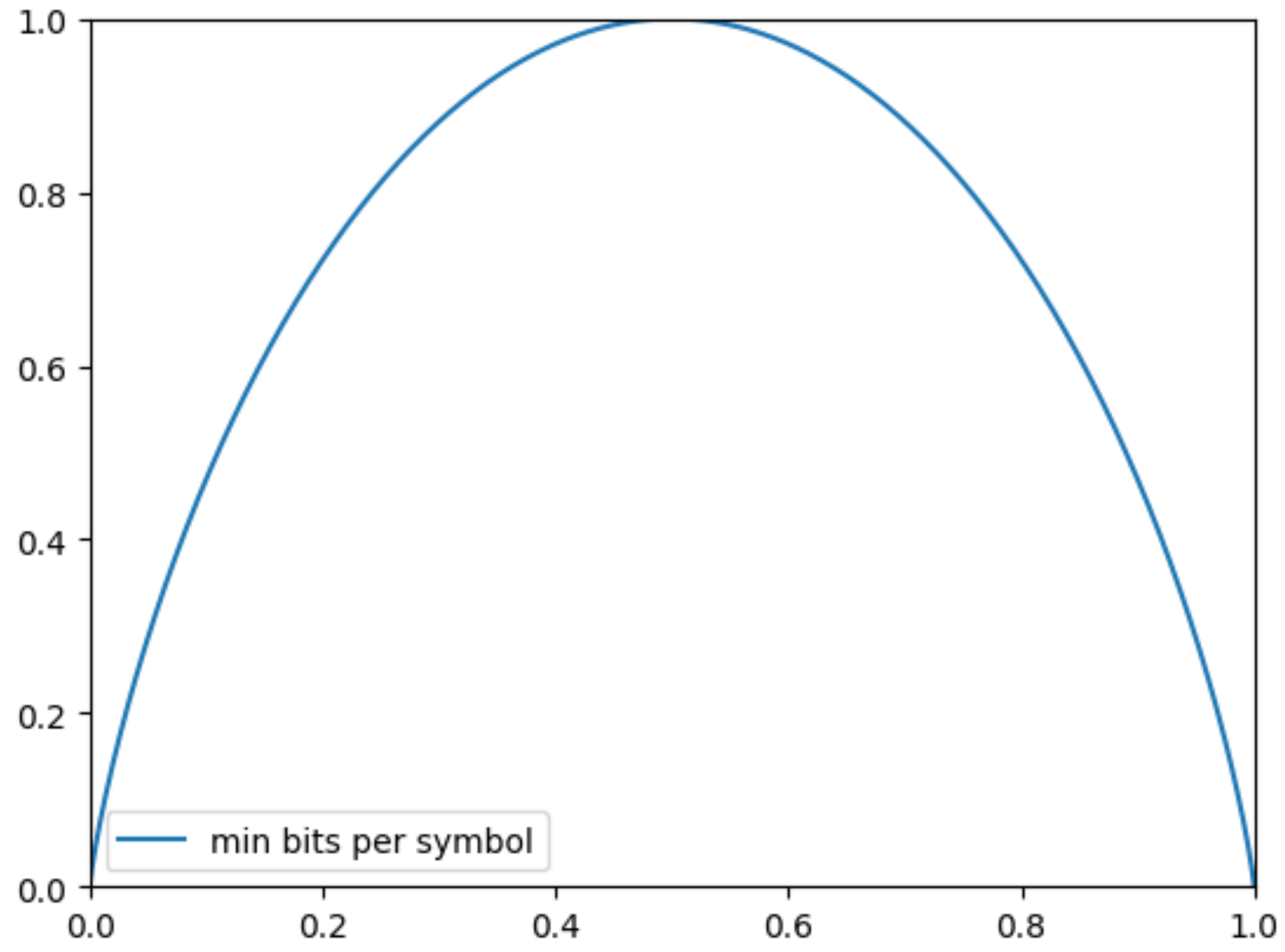
Compressed Bloom Filters

What if we want something like a Bloom filter but care about size a lot more than query time?

Compressed Bloom Filters

What if we want something like a Bloom filter but care about size a lot more than query time?

- Idea:
  - Make a huge Bloom filter so we have really low false positive rates.
  - Only use one hash function, so Bloom filter is mostly zeros (instead of 50/50).
  - Entropy bound says we can compress this a lot?



"Compressed Bloom Filters" by Mitzenmacher (2001)

Compressed Game Solving (personal plug)

How do computers solve a game to play perfectly?

- Some games can be implemented with set operations.
- Deterministic finite automata (DFAs) are one way to represent those sets.

Compressed Game Solving (personal plug)

How do computers solve a game to play perfectly?

- Some games can be implemented with set operations.
- Deterministic finite automata (DFAs) are one way to represent those sets.

But most of the set operations on DFAs take quadratic time.

- Worst case complexity comes from worst case output size.
- Worst case output size from quadratic pairs of input states.
- Game solving actually has linear number of pairs in practice.
  - So roughly $O(n)$ pairs vs $O(n^2)$ possibilities.
  - Another big sparse bit vector?

Compressed Game Solving (personal plug)

How do computers solve a game to play perfectly?

- Some games can be implemented with set operations.
- Deterministic finite automata (DFAs) are one way to represent those sets.

But most of the set operations on DFAs take quadratic time.

- Worst case complexity comes from worst case output size.
- Worst case output size from quadratic pairs of input states.
- Game solving actually has linear number of pairs in practice.
  - So roughly $O(n)$ pairs vs $O(n^2)$ possibilities.
  - Another big sparse bit vector?

Major implementation decision was how to balance memory usage vs computation.

Previous Symbol Distribution as Next Symbol Distribution

Proposal:

- Seed Huffman encoder with a default distribution (probably uniform).
- Update distribution after each symbol encoded.
  - New distribution based on all previous symbols.

This distribution will automatically adjust to languages, vocabulary in input text…

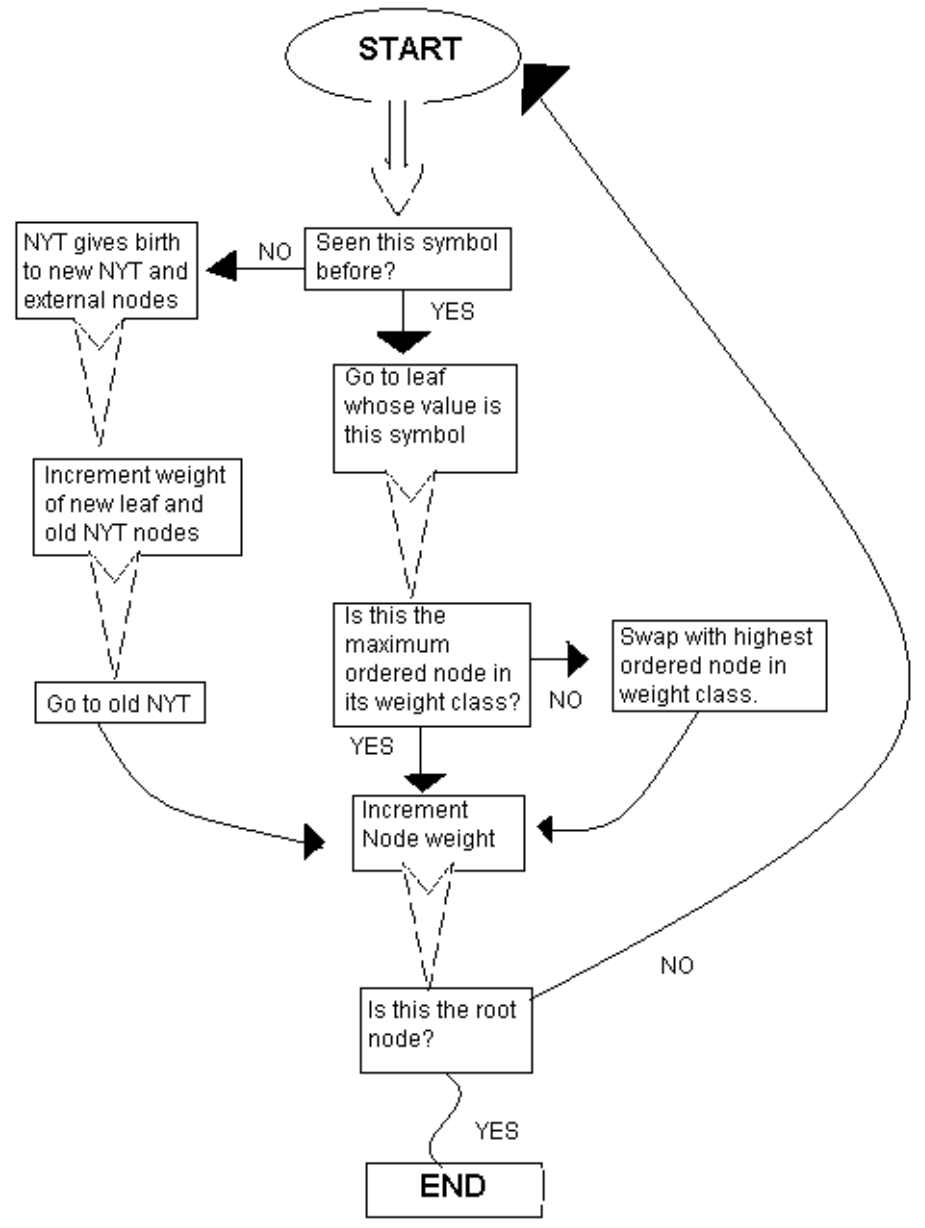
“Design and Analysis of Dynamic Huffman Codes” by Vitter (1987)

- Not the first with this scheme.

Dynamic Huffman Codes

Three takeaways for dynamic Huffman codes…

1. They are a little bit complicated.
2. The updates are often slow.
3. The performance is not always great.


START
Seen this symbol before?
NO
NYT gives birth to new NYT and external nodes
Increment weight of new leaf and old NYT nodes
Go to old NYT
YES
Go to leaf whose value is this symbol
Is this the maximum ordered node in its weight class?
NO
Swap with highest ordered node in weight class.
YES
Increment Node weight
Is this the root node?
NO
YES
END

"Introduction to Data Compression" by Sayood (2001)

Dynamic Huffman Codes

Short version:

- After each symbol, update probabilities based on application.
- Reconstruct Huffman code tree based on shifting probabilities.
- Most commonly, maintain tree with the empirical probabilities so far.
    - Either whole prefix or last k symbols.

Dynamic Huffman Codes

Short version:

- After each symbol, update probabilities based on application.
- Reconstruct Huffman code tree based on shifting probabilities.
- Most commonly, maintain tree with the empirical probabilities so far.
    - Either whole prefix or last k symbols.

More details

- If n possible symbols, this is $O(n)$ work per update.
    - A lot of the optimal tree can change if the probability changes enough.
    - Quadratic compression costs?
- Better version:
    - Only rebuild if probabilities change enough to change the tree.
    - $O(1/p(x))$ work to check.
    - Still might update most of the tree.

Dynamic Huffman Codes

For a given stream of t symbols where the best static Huffman code uses S bits,

- If the Huffman encoding is updated after every symbol based on all previous symbols, then at most $2S + t$ bits are used.

"Design and Analysis of Dynamic Huffman Codes" by Vitter (1987)

Dynamic Huffman Codes

For a given stream of t symbols where the best static Huffman code uses S bits,

- If the Huffman encoding is updated after every symbol based on all previous symbols, then at most $2S + t$ bits are used.
  - Double space usage is not great?
  - Worst case based on Fibonacci sequence probabilities - not natural?

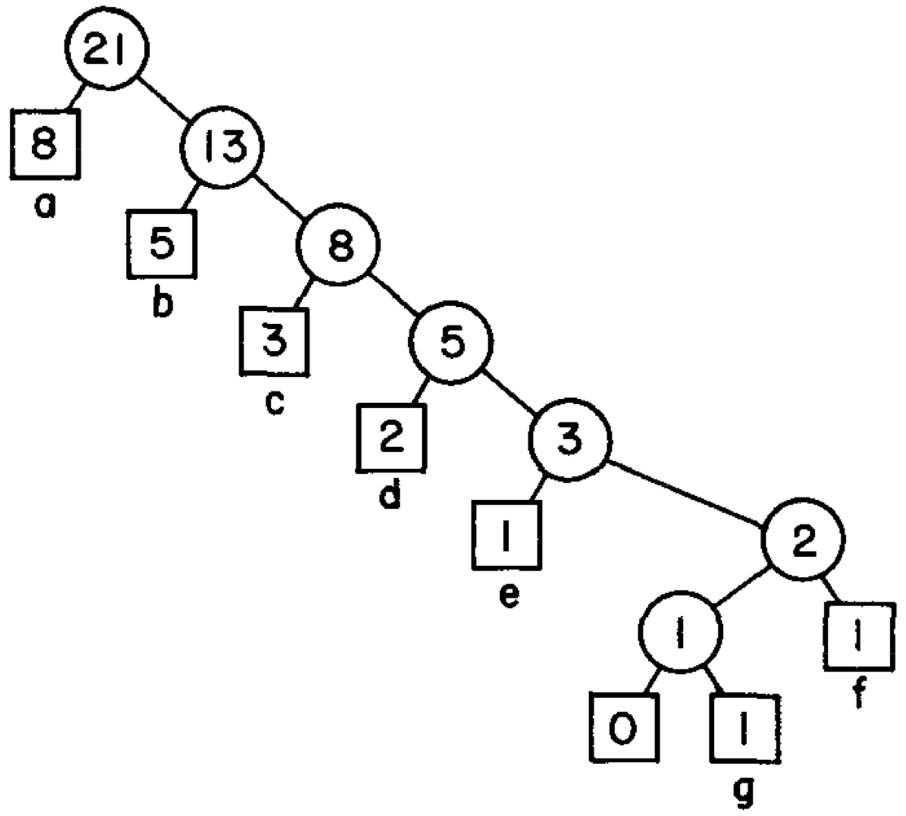


FIG. 5. Illustration of both the lower bound of Theorem 3.1 and the upper bounds of Lemma 3.2. The sequence of letters in the message so far is "abacabdabaceabacabdf" followed by "g" and can be constructed via a simple Fibonacci-like recurrence. For the lower bound, let $t = 21$. The tree can be constructed without any exchanges of types ↑, ↑↑, or ↓; it meets the first bound given in Theorem 3.1. For the upper bound, let $t = 22$. The tree depicts the Huffman tree immediately before the tth letter is processed. If the tth letter is "h", we will have $d_t = 7$ and $h_t = \lceil d_t/2 \rceil - 1 = 3$. If instead the tth letter is "g", we will have $d_t = 7$ and $h_t = \lceil d_t/2 \rceil = 4$. If the tth letter is "f", we will have $d_t = 6$ and $h_t = \lfloor d_t/2 \rfloor = 3$.

"Design and Analysis of Dynamic Huffman Codes" by Vitter (1987)

Dynamic Huffman Codes

A dynamic algorithm with a better worst case behavior exists.

- Limit tree restructuring steps per symbol to avoid wild swings.
- The optimal one-pass algorithm takes $S + t$ bits.
  - At most one extra bit per symbol vs optimal static dictionary.
  - Don’t need to know those probabilities beforehand.

But will move on to a better algorithm now.

“Design and Analysis of Dynamic Huffman Codes” by Vitter (1987)

Limits of Prefix Codes (including Huffman)

What kinds of limits have we seen on prefix codes so far?

- Whole number of bits per symbol
- Huffman optimality limited to powers of $\frac{1}{2}$

Which of these is a bigger limit?

Distributions with Extreme Skew

What if one bits have probability 0.000001?

- Literal one in a million

How many bits to encode 1 billion bits with this distribution with a Huffman code?

- 1 billion 😭

Forget about Prefix Codes?

What if one bits have probability 0.000001?

- Literal one in a million

How many bits to encode 1 billion bits with this distribution?

Forget about Prefix Codes?

What if one bits have probability 0.000001?

- Literal one in a million

How many bits to encode 1 billion bits with this distribution?

- There are approximately 10 bits set to one.

Forget about Prefix Codes?

What if one bits have probability 0.000001?

- Literal one in a million

How many bits to encode 1 billion bits with this distribution?

- There are approximately 10 bits set to one.
- Couldn’t we just write out ~10 30 bit numbers saying where they are?

Forget about Prefix Codes?

What if one bits have probability 0.000001?

- Literal one in a million

How many bits to encode 1 billion bits with this distribution?

- There are approximately 10 bits set to one.
- Couldn't we just write out ~10 30 bit numbers saying where they are?
- Yes, but a narrow solution. Mainly when one symbol is nearly always used.

Any more reusable ideas?

Multiple Symbols per Symbol

What if one bits have probability 0.000001?

- Literal one in a million

How many bits to encode 1 billion bits with this distribution?

- What if we add a fake symbol that represents 100K consecutive zeros?

Multiple Symbols per Symbol

What if one bits have probability 0.000001?

- Literal one in a million

How many bits to encode 1 billion bits with this distribution?

- What if we add a fake symbol that represents 100K consecutive zeros?
  - Use this symbol ~10 times until one bit is close.
  - Then ~100K zero bit symbols.
  - Then 1 one bit symbol.
  - Repeat ~10 times.
- Number of bits to encode drops to ~1 million?

Multiple Symbols per Symbol

What if one bits have probability 0.000001?

- Literal one in a million

How many bits to encode 1 billion bits with this distribution?

- What if we add a fake symbol that represents 100K consecutive zeros?
  - Use this symbol ~10 times until one bit is close.
  - Then ~100K zero bit symbols.
  - Then 1 one bit symbol.
  - Repeat ~10 times.
- Number of bits to encode drops to ~1 million?

Gets better if you have more fake symbols of different run lengths.

- Powers of two will get you the previous logarithmic space usage.
- Bound powers of two needed based on $\frac{1}{1-p}$

Range Codes

Introducing this encoding for intuition before harder arithmetic codes…

Imagine we are encoding our data as 50 decimal digit numbers.

- Ignore binary encodings for now.

The range of encodings is

- 00000000000000000000000000000000000000000000000000
- 99999999999999999999999999999999999999999999999999

Range Codes

Introducing this encoding for intuition before harder arithmetic codes…

Imagine we are encoding our data as 50 decimal digit numbers.

- Ignore binary encodings for now.

The range of encodings is

- 00000000000000000000000000000000000000000000000000
- 99999999999999999999999999999999999999999999999999

Suppose the first symbol that we want to transmit is 50/50 likely to be “a” or “b”.

- We will split this range of encodings evenly between “a” and “b”.

Range Codes

The range of encodings is

- 00000000000000000000000000000000000000000000000000
- 99999999999999999999999999999999999999999999999999

Suppose the first symbol that we want to transmit is 50/50 likely to be “a” or “b”.

- We will split this range of encodings evenly between “a” and “b”.

“a”

- 00000000000000000000000000000000000000000000000000
- 49999999999999999999999999999999999999999999999999

“b”

- 50000000000000000000000000000000000000000000000000
- 99999999999999999999999999999999999999999999999999

Range Codes

Suppose the first symbol that we want to transmit is 50/50 likely to be “a” or “b”.

- We will split this range of encodings evenly between “a” and “b”.

“a”

- 00000000000000000000000000000000000000000000000000
- 49999999999999999999999999999999999999999999999999

“b”

- 50000000000000000000000000000000000000000000000000
- 99999999999999999999999999999999999999999999999999

And if after an “a”, the next character is “c” with 20% probability and “d” with 80% probability?

Range Codes

“a”

- 00000000000000000000000000000000000000000000000000
- 49999999999999999999999999999999999999999999999999

And if after an “a”, the next character is “c” with 20% probability and “d” with 80% probability?

“ac”

- 00000000000000000000000000000000000000000000000000
- 09999999999999999999999999999999999999999999999999

“ad”

- 10000000000000000000000000000000000000000000000000
- 49999999999999999999999999999999999999999999999999

Range Codes

Range codes repeatedly divide the current range of numbers according to the next symbol probabilities.

Range Codes

Range codes repeatedly divide the current range of numbers according to the next symbol probabilities.

- Encoding must stop when the range has fewer numbers than next symbol choices.

Range Codes

Range codes repeatedly divide the current range of numbers according to the next symbol probabilities.

- Encoding must stop when the range has fewer numbers than next symbol choices.
- Until that point, probabilities can change arbitrarily with each symbol.
- If all symbols encoded, transmit any number in the range?
  - Some awkwardness here because range is not empty yet.

Range Codes - One in a Million

“0”

- 00000000000000000000000000000000000000000000000000000000
- 99998999999999999999999999999999999999999999999999999999

“1”

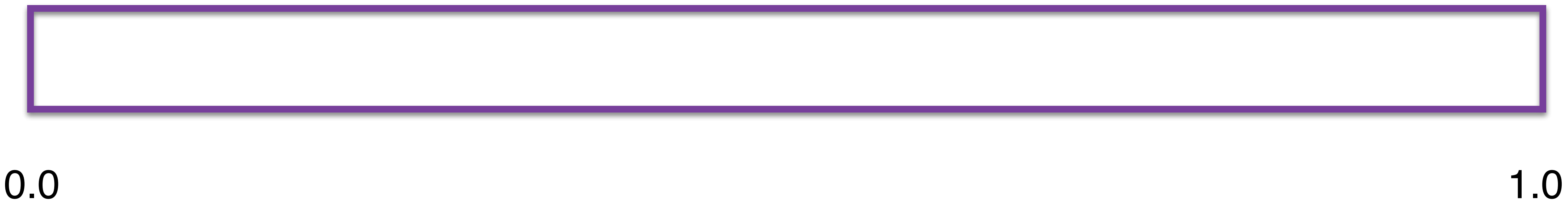
- 99999000000000000000000000000000000000000000000000000000
- 99999999999999999999999999999999999999999999999999999999

Most of the range is left on the zero side.

- No special handling needed.

Dart Throwing Model of Sampling

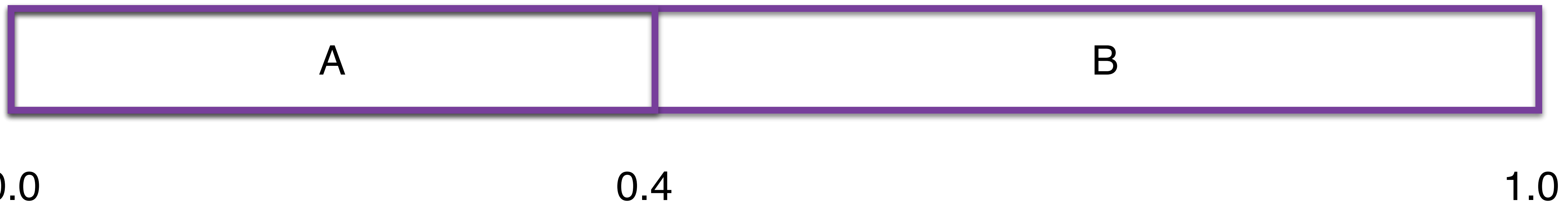
Imagine a 1D dart board labeled as follows -



Throwing a dart into the box is the same as picking a number uniformly at random between zero and one.

Dart Throwing Model of Sampling

Imagine a 1D dart board labeled as follows -

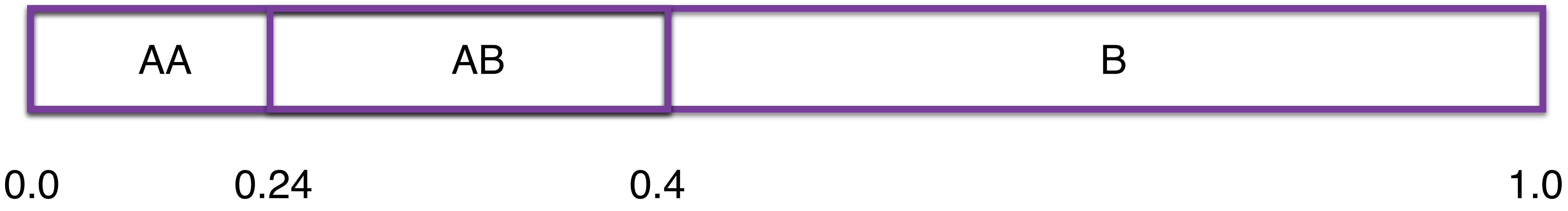


Throwing a dart into the box is the same as picking a number uniformly at random between zero and one.

- If the dart lands on the left, then A was picked.
- If the dart lands on the right, then B was picked.
- The relative lengths of those boxes determine the probabilities.

Dart Throwing Model of Sampling

Imagine a 1D dart board labeled as follows -

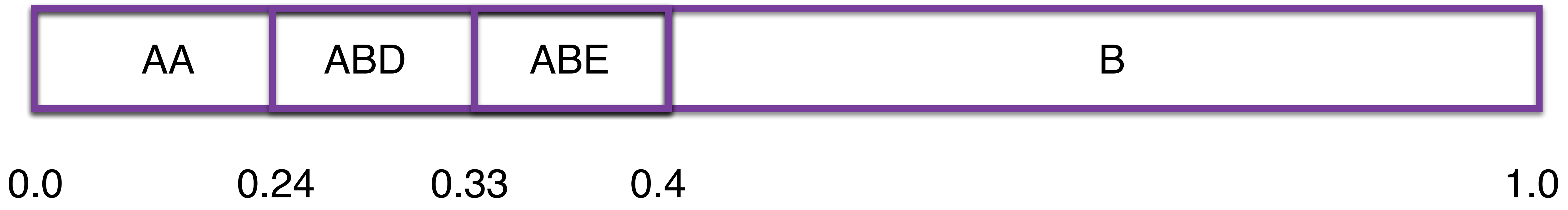


Subdividing each range allows more symbols to be encoded.

- Again, relative sizes control probabilities.
- New symbol probability is fraction of previous range's size.

Dart Throwing Model of Sampling

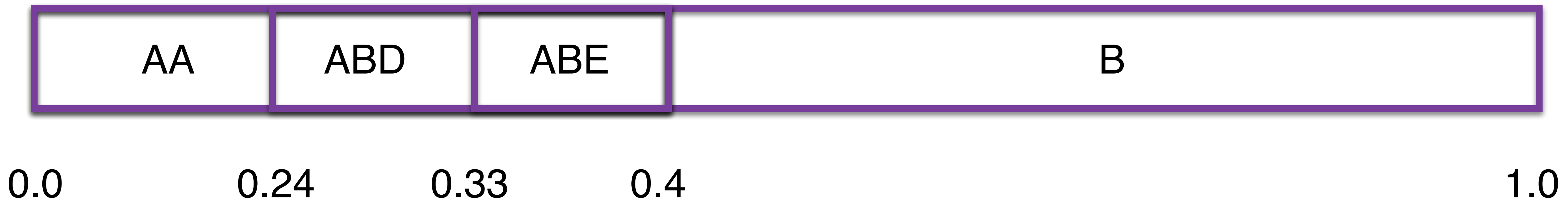
Imagine a 1D dart board labeled as follows -



Subdivisions can continue without bound.

Dart Throwing Model of Sampling

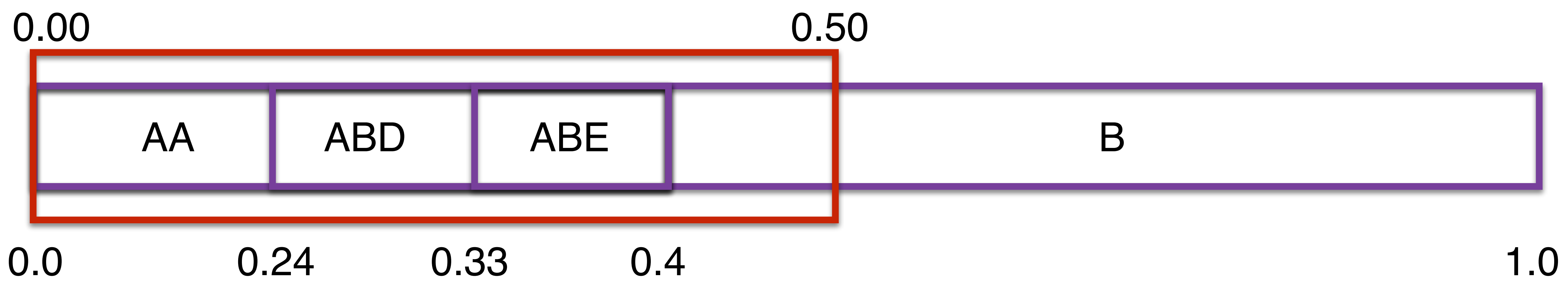
Imagine a 1D dart board labeled as follows -



If we have this layout, how do we sample?

- Pick a number uniformly at random between zero and one.
- Check which box it falls into.
- If we have “enough” precision, we can sample arbitrarily long sequences in “one” step?
- What if we can only sample one random bit at a time?

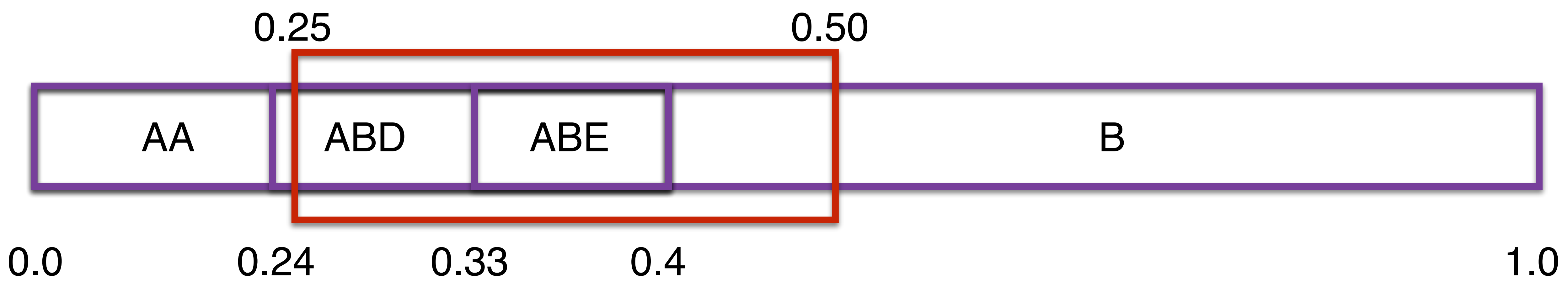
Dart Throwing Model of Sampling



If we have this layout, how do we sample?

- Pick a number uniformly at random between zero and one.
- Check which box it falls into.
- If we have “enough” precision, we can sample arbitrarily long sequences in “one” step?
- What if we can only sample one random bit at a time?
  - Each random bit halves the sample range.

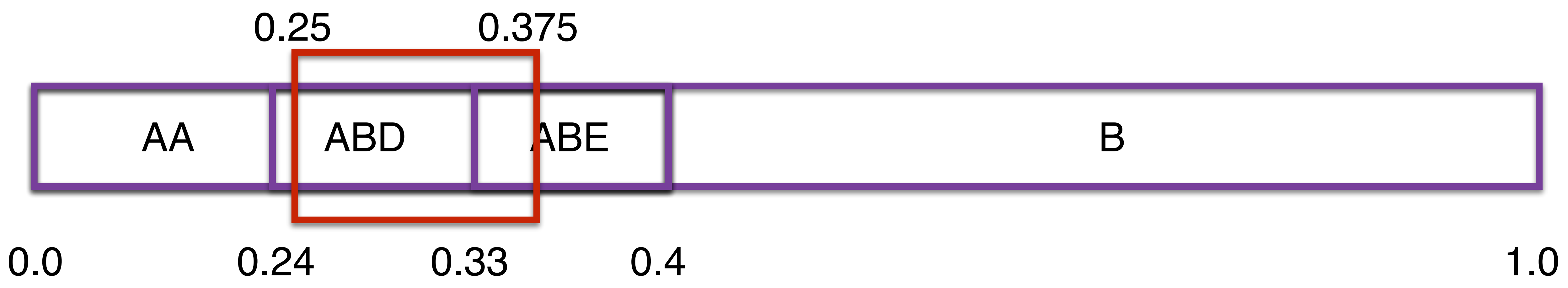
Dart Throwing Model of Sampling



If we have this layout, how do we sample?

- Pick a number uniformly at random between zero and one.
- Check which box it falls into.
- If we have “enough” precision, we can sample arbitrarily long sequences in “one” step?
- What if we can only sample one random bit at a time?
  - Each random bit halves the sample range.

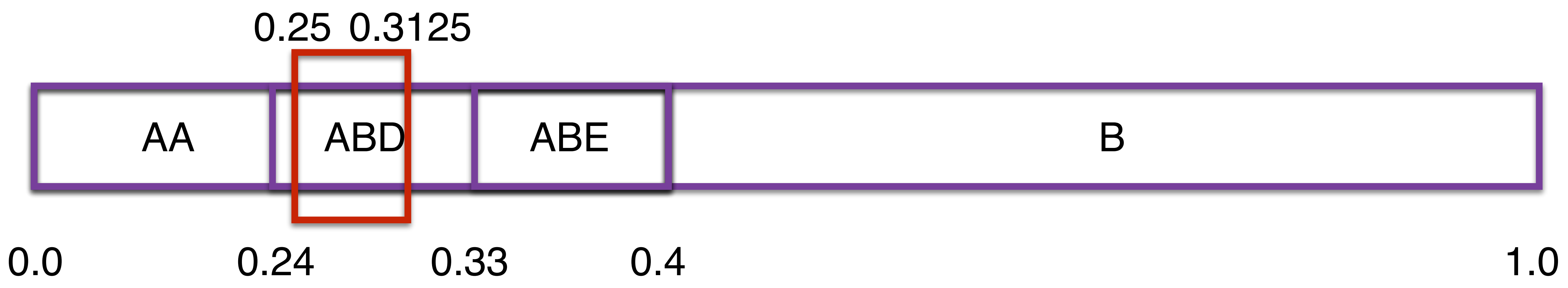
Dart Throwing Model of Sampling



If we have this layout, how do we sample?

- Pick a number uniformly at random between zero and one.
- Check which box it falls into.
- If we have "enough" precision, we can sample arbitrarily long sequences in "one" step?
- What if we can only sample one random bit at a time?
  - Each random bit halves the sample range.

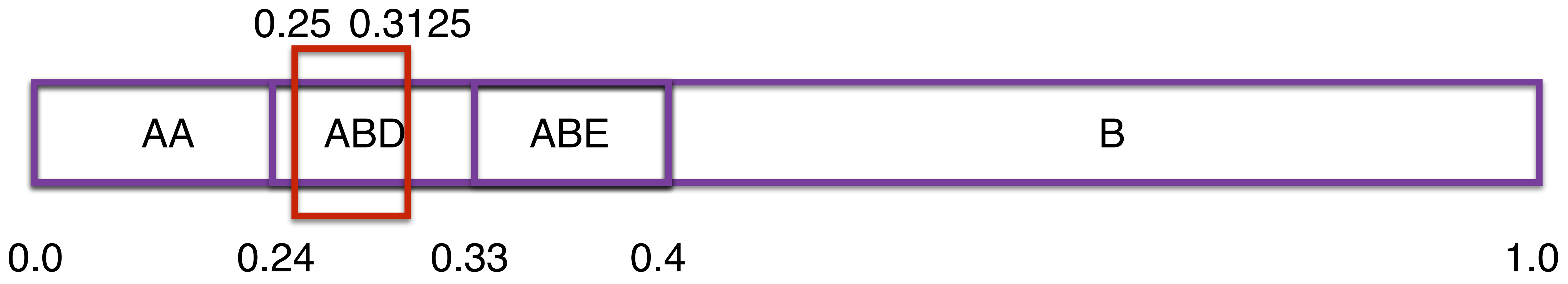
Dart Throwing Model of Sampling



Stop when the random bits shrink the sample range to fit inside one output range.

- That range's symbols are the output of this sample.
- The average number of random bits is the binary entropy.
- How do we turn this into compression?

Arithmetic Coding



The random bits that chose ABD are also the encoding of an arithmetic code.

- 0100

More specifically,

- 0.0100* always fits within the ABD range.
- Pick the shortest such prefix so that all competitions are in the target range.

Arithmetic Coding

More formally,

- Start with current range [0,1).

Arithmetic Coding

More formally,

- Start with current range [0,1).
- Each symbol choice has “bottom” and “top” probabilities.
  - a = [0, 0.25)
  - b = [0.25, 0.30)
  - c = [0.30, 1.00)

Arithmetic Coding

More formally,

- Start with current range [0,1).
- Each symbol choice has “bottom” and “top” probabilities.
  - a = [0, 0.25)
  - b = [0.25, 0.30)
  - c = [0.30, 1.00)
- To encode a symbol,
  - new_bottom =

    current_bottom + (current_top - current_bottom) * symbol_bottom
  - new_top =

    current_bottom + (current_top - current_bottom) * symbol_top

Arithmetic Coding

More formally,

- Start with current range [0,1).
- Each symbol choice has “bottom” and “top” probabilities.
  - a = [0, 0.25)
  - b = [0.25, 0.30)
  - c = [0.30, 1.00)
- To encode a symbol,
  - new_bottom =
    current_bottom + (current_top - current_bottom) * symbol_bottom
  - new_top =
    current_bottom + (current_top - current_bottom) * symbol_top
- When done encoding all symbols, pick bit prefix so that all continuations are in that range.
  - Roughly $-\log(\text{current\_top} - \text{current\_bottom})$ bits needed.

Arithmetic Coding

More formally,

- Start with current range [0,1).
- Each symbol choice has “bottom” and “top” probabilities.
  - a = [0, 0.25)
  - b = [0.25, 0.30)
  - c = [0.30, 1.00)
- To encode a symbol,
  - new_bottom =
    current_bottom + (current_top - current_bottom) * symbol_bottom
  - new_top =
    current_bottom + (current_top - current_bottom) * symbol_top
- When done encoding all symbols, pick bit prefix so that all continuations are in that range.
  - Roughly $-\log(\text{current\_top} - \text{current\_bottom})$ bits needed.
  - This basically gets us the entropy bound.

Arithmetic Coding

Implementation details

- Previous description computed final range, then picked encoding bits.
- This requires too much numeric precision.
  - Slow or will lose information.
- Real encoder picks encoding bits at the same time.
  - Adjusts range to be relative to encoded bits.
  - Reduces precision needs.
  - Makes encoding more incremental.

Arithmetic Coding

Implementation details

- Previous description computed final range, then picked encoding bits.
- This requires too much numeric precision.
  - Slow or will lose information.
- Real encoder picks encoding bits at the same time.
  - Adjusts range to be relative to encoded bits.
  - Reduces precision needs.
  - Makes encoding more incremental.

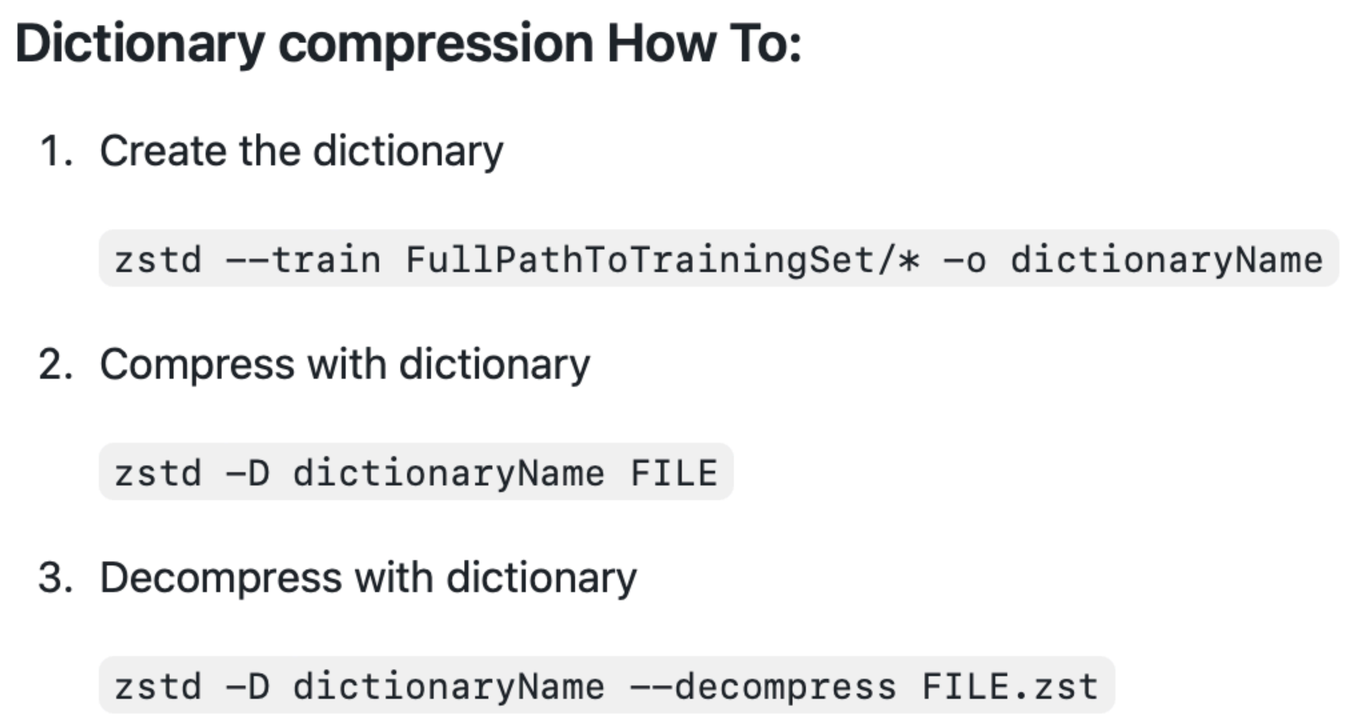
Was not actually implemented for a long time

- Because IBM had patents on it.
- Everyone wanted to use it.

Entropy Coding

Generalization of Huffman and arithmetic coding

- Compression methods that achieve the entropy bound (given method constraints)
- Some interface convergence
  - Train / model dictionary of probabilities



Dictionary compression How To:

1. Create the dictionary

```
zstd --train FullPathToTrainingSet/* -o dictionaryName
```

2. Compress with dictionary

```
zstd -D dictionaryName FILE
```

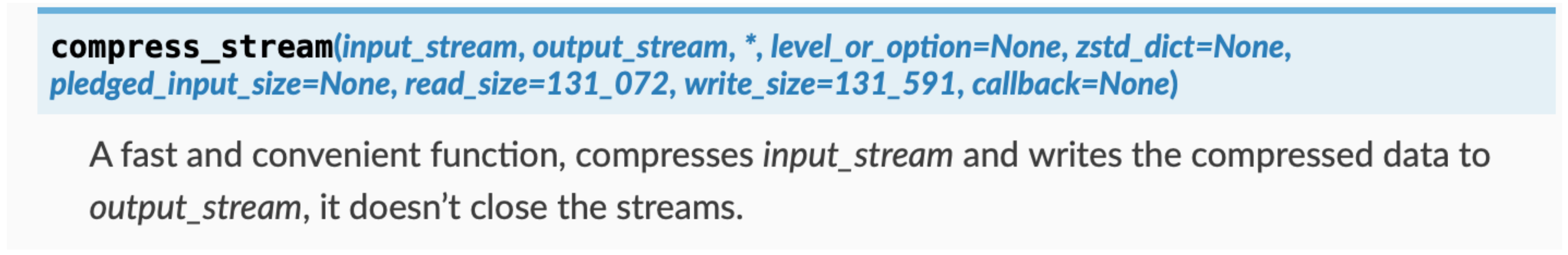
3. Decompress with dictionary

```
zstd -D dictionaryName --decompress FILE.zst
```

https://github.com/facebook/zstd/tree/dev

Entropy Coding - Advanced Interfaces

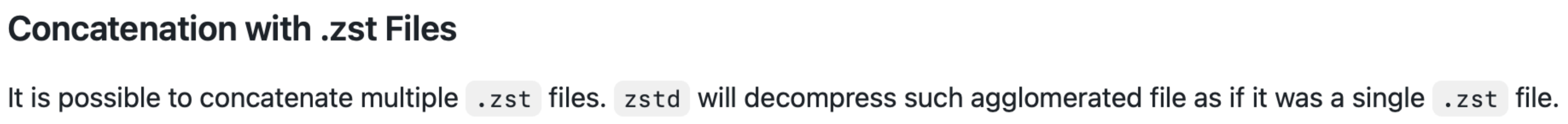
Can keep changing dictionaries



compress_stream(*input_stream, output_stream, *, level_or_option=None, zstd_dict=None, pledged_input_size=None, read_size=131_072, write_size=131_591, callback=None*)

A fast and convenient function, compresses *input_stream* and writes the compressed data to *output_stream*, it doesn't close the streams.

https://pyzstd.readthedocs.io/en/stable/



Concatenation with .zst Files

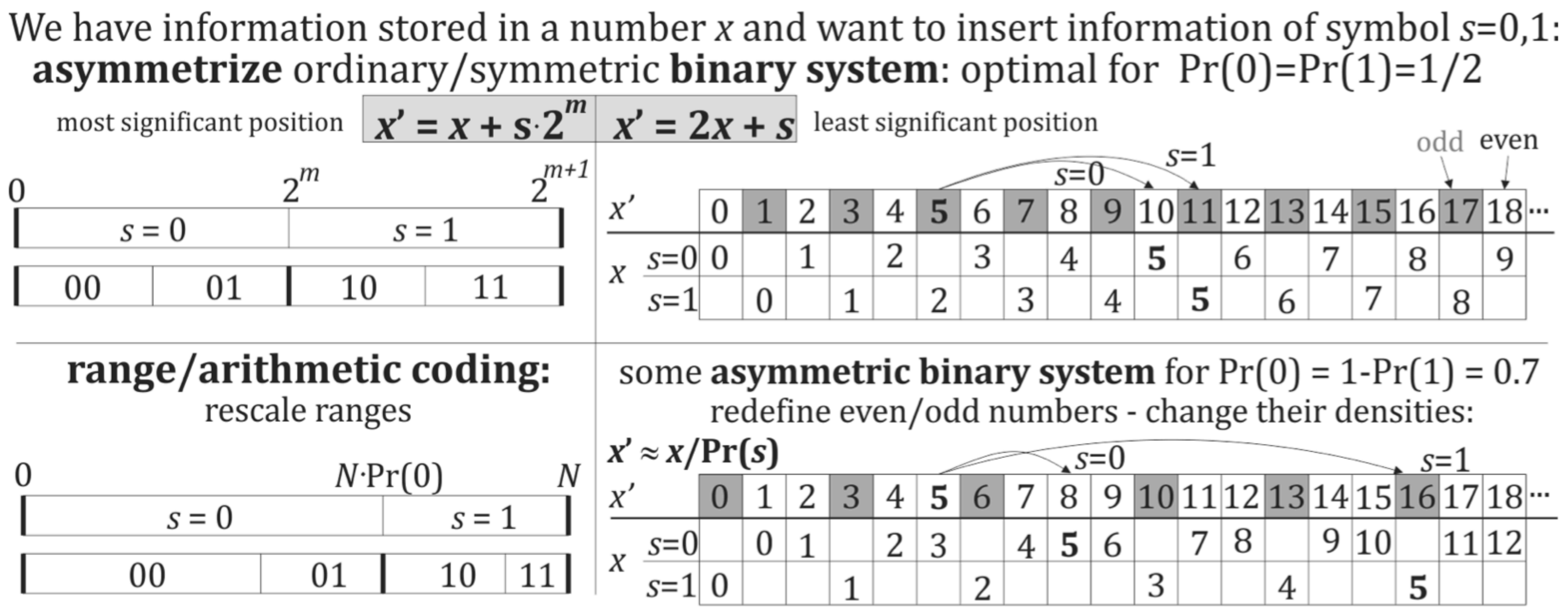
It is possible to concatenate multiple `.zst` files. `zstd` will decompress such agglomerated file as if it was a single `.zst` file.

https://github.com/facebook/zstd/blob/dev/programs/zstd.1.md

Asymmetric Numeral Systems

Won’t describe this now, but the new kid on the block for compression.

- Also hits entropy bound like arithmetic encoding.
- Faster in practice.
- Used by zstd in previous examples
  - Written by Facebook (now Meta) and open sourced


We have information stored in a number x and want to insert information of symbol s=0,1:
asymmetrize ordinary/symmetric binary system: optimal for Pr(0)=Pr(1)=1/2
most significant position
x' = x + s·2^m
x' = 2x + s
least significant position
0
2^m
2^(m+1)
s = 0
s = 1
00
01
10
11
odd
even
s=0
s=1
x'
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 ...
x
s=0: 0 1 2 3 4 5 6 7 8 9
s=1: 0 1 2 3 4 5 6 7 8
range/arithmetic coding:
rescale ranges
0
N·Pr(0)
N
s = 0
s = 1
00
01
10
11
some asymmetric binary system for Pr(0) = 1-Pr(1) = 0.7
redefine even/odd numbers - change their densities:
x' ≈ x/Pr(s)
s=0
s=1
x'
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 ...
x
s=0: 0 1 2 3 4 5 6 7 8 9 10 11 12
s=1: 0 1 2 3 4 5

“Asymmetric numeral systems: entropy coding combining speed of Huffman coding with compression rate of arithmetic coding” by Duda (2014)

Applications - File Compression

- Save disk space (duh)
- Read files faster?
  - Store compressed
  - Read and decompress on the fly
  - Depends on relative CPU speed vs disk bandwidth
  - Multi-core scenarios usually make this more favorable

Applications - File Compression

- Save disk space (duh)
- Read files faster?
  - Store compressed
  - Read and decompress on the fly
  - Depends on relative CPU speed vs disk bandwidth
  - Multi-core scenarios usually make this more favorable

- Always use this for cloud storage!!!
  - Pay less
  - Download/fetch faster
  - Usually still important with cloud compute

Applications - File Compression

- Save disk space (duh)
- Read files faster?
  - Store compressed
  - Read and decompress on the fly
  - Depends on relative CPU speed vs disk bandwidth
  - Multi-core scenarios usually make this more favorable

- Always use this for cloud storage!!!
  - Pay less
  - Download/fetch faster
  - Usually still important with cloud compute

- Some web server / browser combinations do this automatically
  - Built into HTTP protocol
  - Needs support for specific compression algorithm on both sides

Memory Saving

This used to be a near universal recommendation for Macs -

RAM Doubler 1.x | RAM Doubler 2 for Mac OS | RAM Doubler 8 for Mac OS | RAM Doubler 9 for Mac OS | RAM Doubler for Windows 3.1

Built into Mac OS as of version 10.9…

- Via integration into virtual memory system
- Both memory pages in RAM and on disk can be compressed.

Image source: https://apple.fandom.com/wiki/RAM_Doubler

Applications - Cache Efficiency

For caches where the values are large compared to the keys…

- Compress the values!
- Compression ratio is effectively an increase in cache size.
- Large caches probably sharing values over a network anyway.

Applications - Compressed Bloom Filters

What if we want something like a Bloom filter but care about size a lot more than query time?

- Emphasis on reducing transmission costs over fast access

TLDR

- Use one hash function and very big uncompressed sketch
- Get compressed false positive rate $0.5^{m/n}$ vs uncompressed $0.6185^{m/n}$
- m = number of bits in representation

One trick of note

- Break the Bloom filter into blocks and compress them separately.
- Only need to decompress one block to check the selected bit.
- Standard tradeoff to avoid decompressing whole file
  - Small, roughly constant overhead per block

Applications - Modeling

Compression efficiency is closely connected to accuracy of probabilities.

- The dynamic Huffman coding example showed how those could be manipulated for poor performance.

Applications - Modeling

Compression efficiency is closely connected to accuracy of probabilities.

- The dynamic Huffman coding example showed how those could be manipulated for poor performance.

Can use compression rates to assess predictive model quality

- Better model = better compression
- One way to evaluate language models

There are even a couple papers saying that gzip is a decent language model!

Applications - Modeling

Compression efficiency is closely connected to accuracy of probabilities.

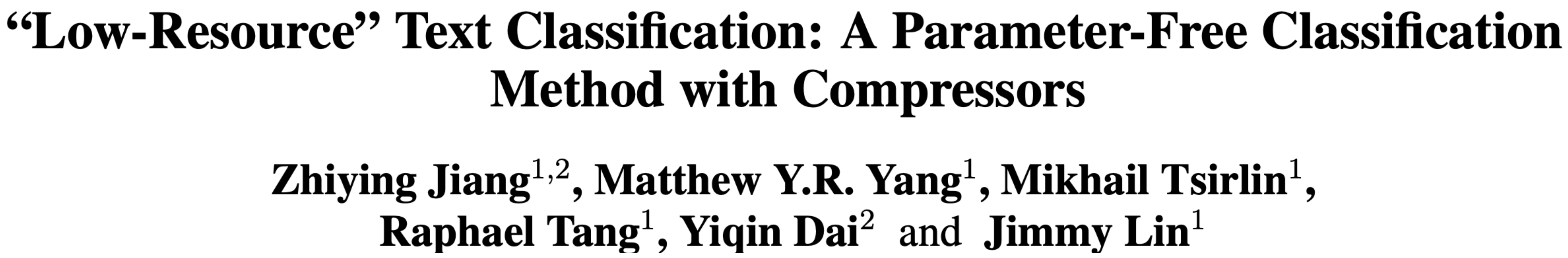
- The dynamic Huffman coding example showed how those could be manipulated for poor performance.

Can use compression rates to assess predictive model quality

- Better model = better compression
- One way to evaluate language models

There are even a couple papers saying that gzip is a decent language model!

- Half joking, but more seriously



“Low-Resource” Text Classification: A Parameter-Free Classification Method with Compressors

Zhiying Jiang[1,2], Matthew Y.R. Yang[1], Mikhail Tsirlin[1], Raphael Tang[1], Yiqin Dai[2] and Jimmy Lin[1]