Compression Algorithms

Eli Maynard

May 9, 2019

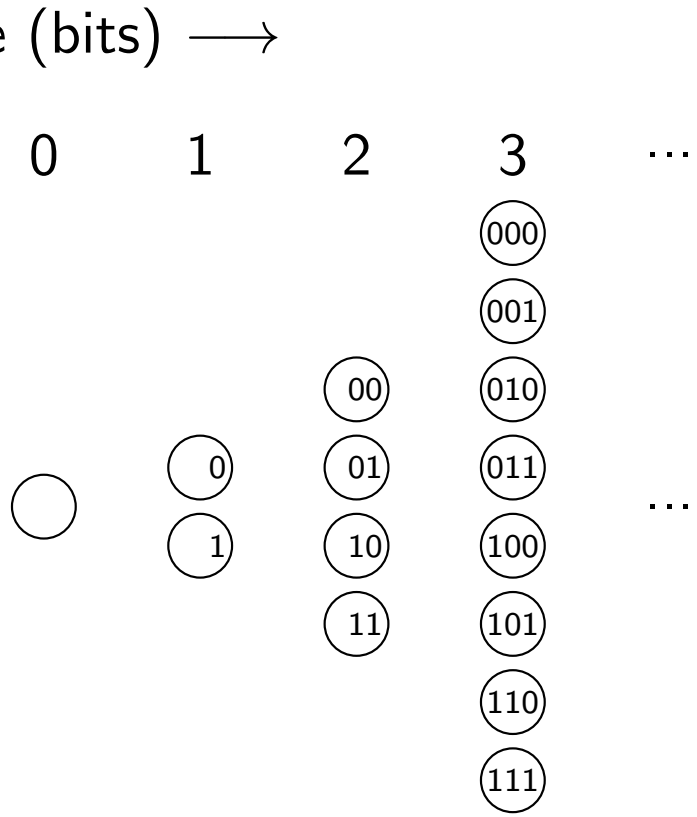
Compression Algorithms

- Makes files smaller
  - "File": string of 1s and 0s[1]
- Two types
  - Lossy: loses information during compression
    - jpg
    - mp3
  - Lossless: retains all information
    - tiff
    - png
    - gif

[1] For the purposes of this presentation. Note that we consider encodings here to be implicit and not part of the file.

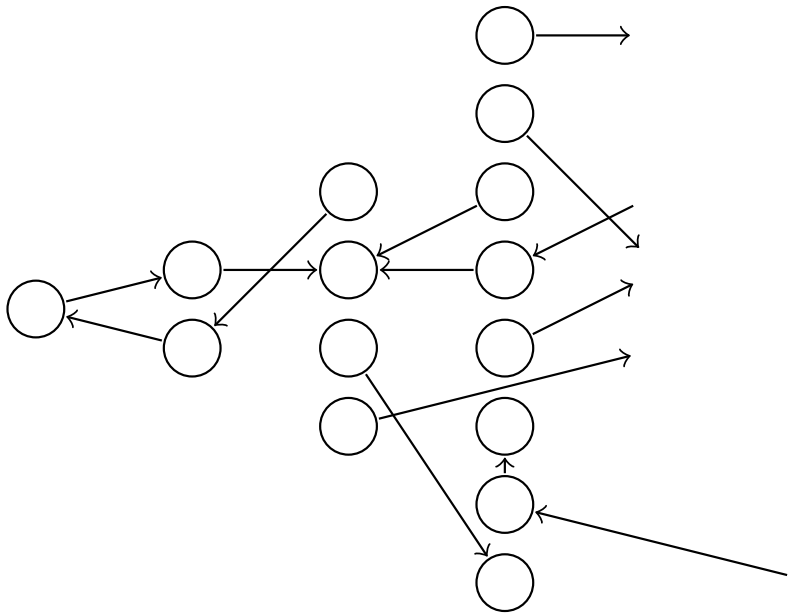
Domain

Our domain is files. We can visualize all files $\mathbb{F}$ like so...



Algorithms on Domains

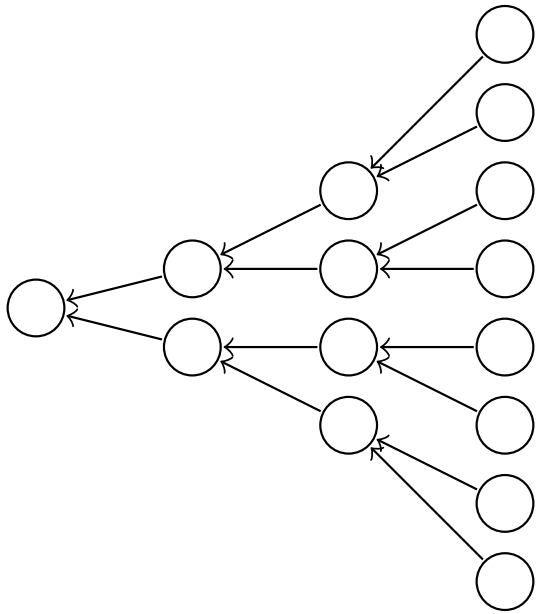
Compression algorithms move stuff around $\mathbb{F}$: you put a file in, you get a file out. $C : \mathbb{F} \to \mathbb{F}$



Goal: *compress* files: move to left $\longleftarrow$. I call moving to right *bloating*.

Lossy v Lossless

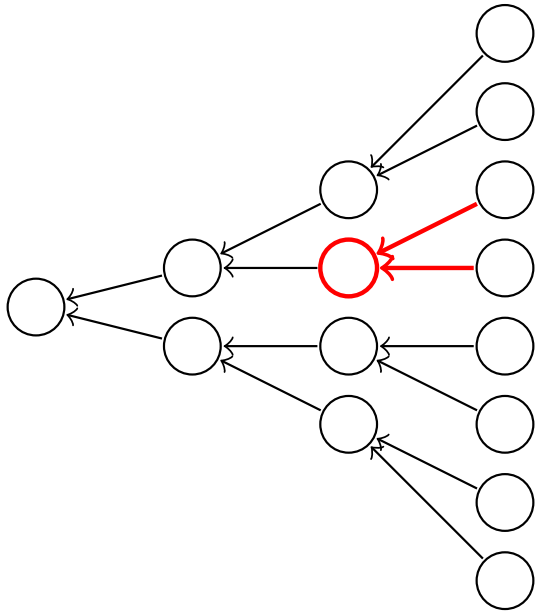
Goal: move stuff to the left $\longleftarrow$. Solution: move stuff to the left $\longleftarrow$:



Issue: overlap. Some destination files are mapped to by multiple source files. This is an issue ...

Lossy v Lossless

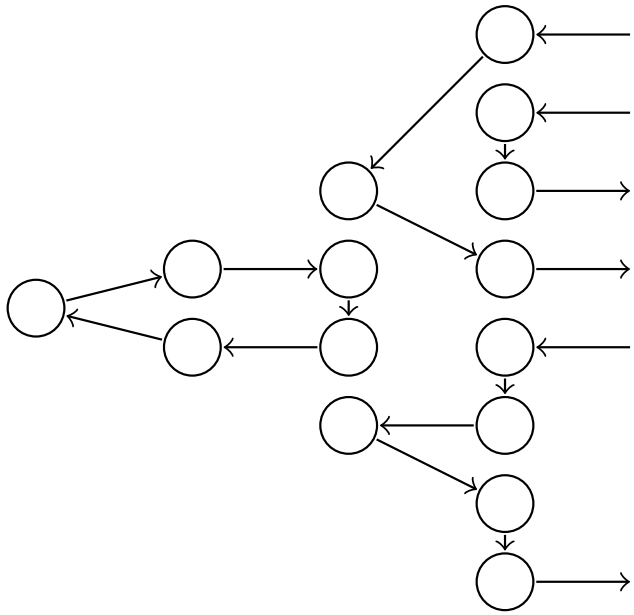
Because of overlap, we cannot tell where a file came from. Thus, when decompressing, we're not sure where to go.



Since we're losing information, we call the algorithm *lossy*.

Lossy v Lossless

We like perfect decompression. So let's retain all information. Then, no two files can map to the same file. These algorithms are called *lossless*.



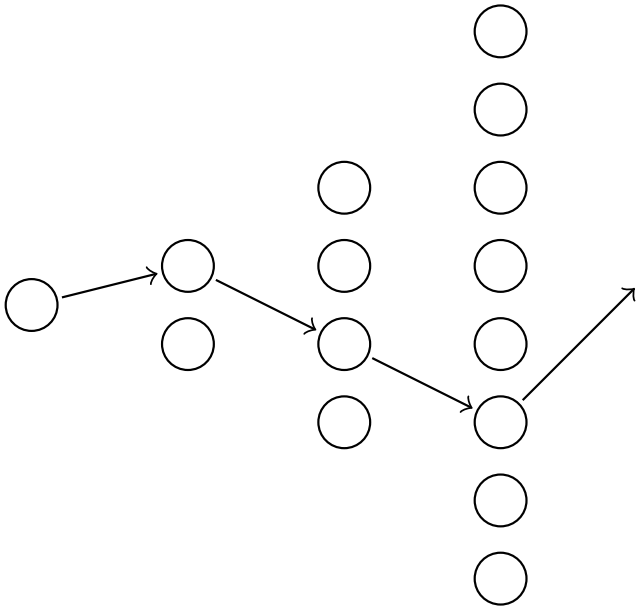
Because no two files can map onto the same file, in order for some files to be compressed, others must be bloated. More on that next slide ...

Compression Necessitates Bloating

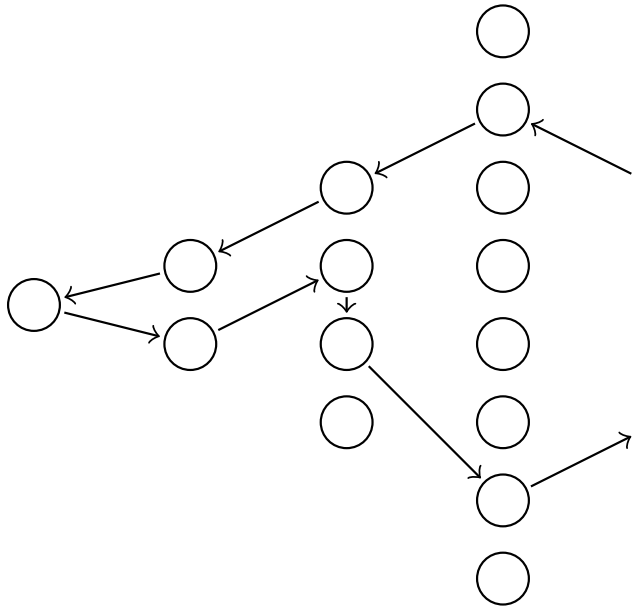
- A compression algorithm C will move files around $\mathbb{F}$
- Consider one file f
- C may move f onto g
- C is lossy $\implies$ g doesn't have to move—overlap is allowed
- C is lossless $\implies$ g *has* to move—no overlap allowed
- g may swap to f, or move to h, then forcing h to move
- Repeat to infinity $\longrightarrow$ lossless algorithms must form chains or cycles of movement
- The left of the graph is finite, and the right is inifnite.
  - C may chain infinitely far right, but
  - if it goes left, it *must* also go right
  - I.e., compression necessitates bloating

Compression Necessitates Bloating: Diagrams

"C may chain infinitely far right ..."



"... but if it goes left, it *must* also go right"



Theorem Statement

- Theorem: necessitated bloating
- If a compression algorithm C changes the size of at least one file, then it bloats at least one file
- $\forall C[\exists f(S_C(f) \neq 0) \implies \exists g(S_C(g) < 0)]$
- Proof
  - At least one file f moves
  - If f is bloated, then the proof is complete
  - If f is compressed, then another one is bloated—see previous slides

Interlude: Scoring

- May want to quantify how well C performs on f
- Let S (for 'score') be the number of bits saved by C on f
  - $S_C(f) := |f| - |C(f)|$
- Another metric is the *compression ratio* $|C(f)|/|f|$

Coolish Theorem

- Say you're trying to be clever
- Have two compression algorithms C_1 and C_2, but not sure which will work better on D
- Make an algorithm $\mathscr{C}$ which chooses the smaller one per-file, prepending a bit to differentiate:[2]

$$\mathscr{C}(f) := \begin{cases} \text{`0'} \oplus C_1(f) & S_{C_1}(f) \geq S_{C_2}(f) \\ \text{`1'} \oplus C_2(f) & S_{C_2}(f) > S_{C_1}(f) \end{cases}$$

- The empty file $e =$ '0' cannot be moved to the left
- Thus $S_{C_i}(e) \leq 0$
- Thus $S_{\mathscr{C}}(e) < 0$ due to prepended bit
- $\mathscr{C}$, our clever algorithm, is imperfect ;(
- This theorem isn't particularly impressive but hints at, I feel, deeper truths

[2] $\oplus$ denotes string/file concatenation

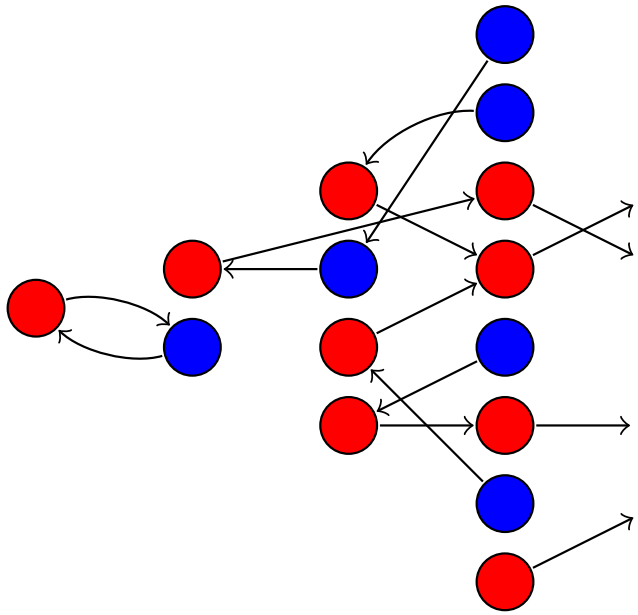
Lossless $\not\Rightarrow$ lossless

- Lossless compression can only "move stuff around"
- If one file is compressed, another is bloated
- Lossless compression is useless!?
- Two solutions[3]:
  - Subdomains
  - Infinity ∞

[3] That I have found

Solution 1: Subdomains

- Usually, we want to emphasize a particular subdomain $D \subsetneq \mathbb{F}$
  - Ex: ascii: care about sentences, not gibberish
    - The quick brown fox jumped over the lazy dog ✓
    - DgSZ31CJC7tGa0ukWgTIttIWyYPDNDaFx ×
- Compress ($\longleftarrow$) files in D; bloat ($\longrightarrow$) files not in D



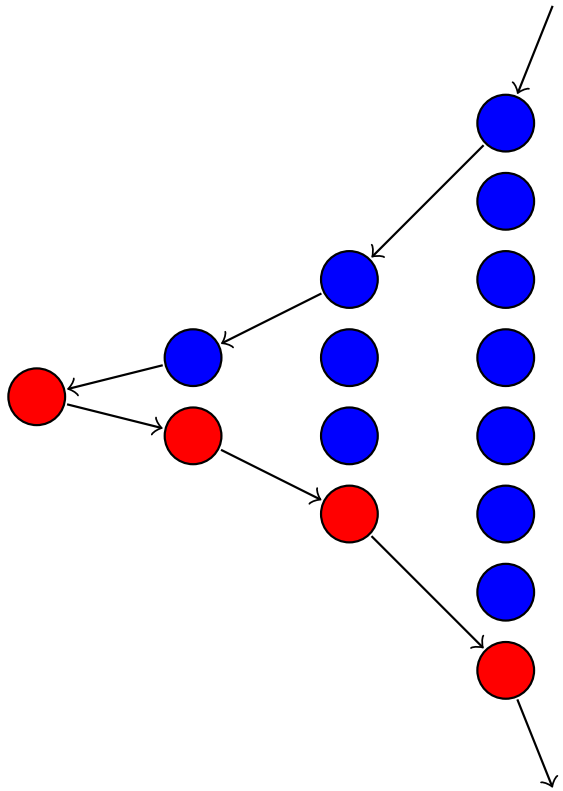
No file has two arrows pointing to it (lossless ✓); files in D moves to the left (compresses ✓). Because some files move to the left, some *must* move to the right; make those not in D move.

People Know This ✓

- LZW: Pattern-based compression
  - On 821 bytes of Lorem ipsum, $S = 80$
  - On 8399 bytes of random characters, $S = -1$
  - In general, $S(\text{random string}) = -1$
- Pattern-based compression will generally work like this
  - Encoded strings are $\mathit{dict} \oplus \mathit{payload}$
  - So if can't find patterns, small overhead for empty dict

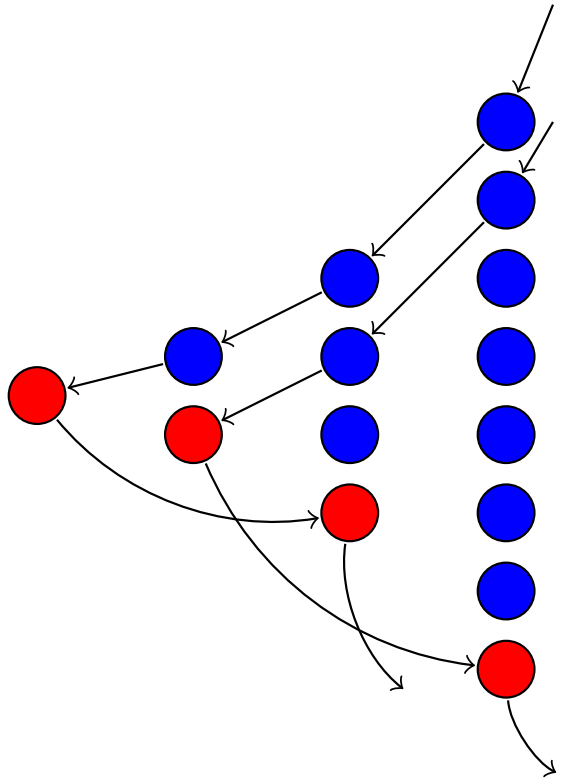
Solution 2: ∞

- Say:
  - We don't care about files that are all 1s
  - Not-all-1 files are in D and all-1 files are not in D.
  - Can do this (call it E_1):



Solution 2: ∞ (2)

But we can do better (call it E_2):



Solution 2: ∞ (3)

- Can make N as large as we want
- As $N \to \infty$, E_N gets better and better for files in D
  - Asymptotically[4]
- In general, we can punish the files we don't care about (not in D) as much as we want, rewarding the ones we do care about (in D)

[4] Technically, since I haven't defined a way to quantify how 'good' C is, I can't *truly* claim it's 'asymptotic'; however, hopefully the point is clear

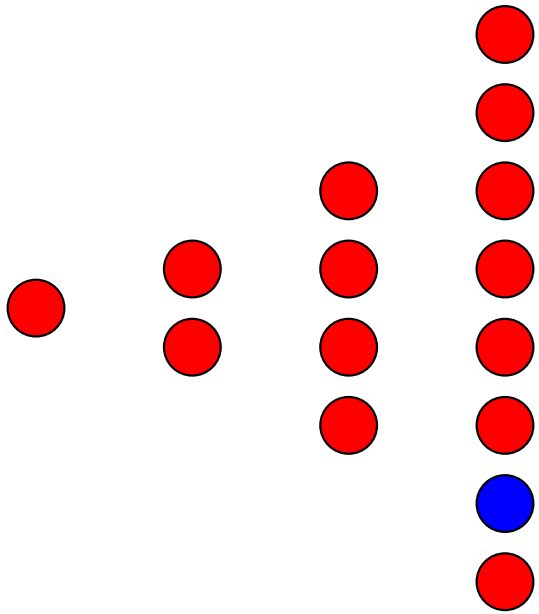
I Must Be Missing Something

- For LZW, strings with no patterns have $S \approx -1$
  - Given our new context, this should be *much* lower
  - On the string "04vbU3EDPc", $S = -10000$ seems more appropriate
- The same general behavior should happen with Huffman and any dictionary-based algorithm
- I must be missing something
  - I did almost no research for this, after all

:(

- It gets weirder
- In real life, the ratio of $|D|/|\mathbb{F}|$ is often ≈ 0
  - Almost all strings are gibberish rather than useful sentences
- And yet bzip2, called 'good'[5], gets ratios of around 20%[6].
  - Tests run on the Linux Kernel source code

Artist's rendition of a real-world D:



[5] In *Simple, Fast, and Efficient Natural Language Adaptive Compression*

[6] https://www.rootusers.com/gzip-vs-bzip2-vs-xz-performance-comparison/

Woohoo

Cool, now for some less-grounded, theoretical things

Abstracting

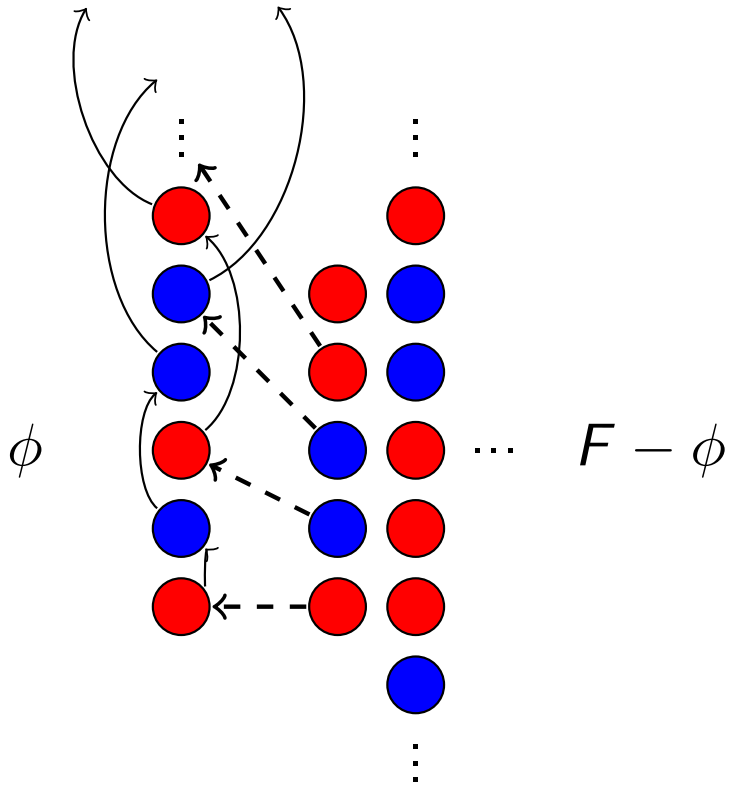
- We can abstract compression algorithms
- Instead of $\mathbb{F}$, use any set F
- Instead of file size, minimize some goal $G(f)$ for each 'file' (element of F)
- Still have D all the same ✓

Infinite Leftness

- A lot of the fun of lossless compression came from the the boundedness of the left portion of the graph
- Can break this for instance by:
  - Making G have no minimum
    - E.g. with files: $G(f) := 1/|f|$
  - Making some infinite subset of F have the same G
    - E.g. with files: If the "10 bits" column were infinitely high
    - Then we can abuse Hilbert's hotel

Infinite Subset

Example: say F is countable and an infinite subset ϕ of F all have the same G. Then our dreams come true ala Hilbert's Hotel:



Since both ϕ and $F - \phi$ are countable, it is possible for C to move all files into ϕ, thus guaranteeing constant G. Cool.

One More Thing

- D is a simplification
  - In reality, edge-cases exist
  - Should "I am a sentence with words vHnXbumapNkep2PDnQRB" be in D?
    - Has words, has gibberish
  - Should "'Twas brillig, and the slithy toves / Did gyre and gimble in the wabe"?
    - Has structure and has words... kinda?
  - It makes more sense to have a "relevance score" or weight $W : F \to [0, 1]$
  - Goal is to match lower W values with worse compression and higher with better

Thinkybois

- At this point, I am personally left with questions:
- How to measure a ‘good’ vs ‘bad’ compression algorithm?
- How to use this perspective to make a good compression algorithm?
- What if the barber doesn’t *have any* facial hair?