Explicit Time and Space Efficient Encoders Exist

Only With Random Access

Joshua Cook and Dana Moshkovitz

What is an Error Correcting Code (Notation)

A function $C: \{0,1\}^n \rightarrow \{0,1\}^m$ is a code with distance d if

For all $x, y \in \{0,1\}^n$ with $x \neq y$ we have $\sum_i [C(x)_i \neq C(y)_i] \geq d$

We call

- $\delta = d / m$ the relative distance of C
- $r = n / m$ the rate of C
- C good (or asymptotically good) if δ and r are constants greater than zero
- any algorithm computing C an encoder for C

Time T And Space S Required for Encoders of Good Codes

with random access to the input

Do Good Codes Require Encoders with S T = $\Omega(n^2)$?

Conjectured by Bazzi and Mitter [Bazzi, Mitter 2005]

Bazzi and Mitter showed time O(n) encoders required space Ω(n)

No!

Some Non-Explicit Good Codes have T = O(n log(n)) and S = O(log(n)) encoders

- Repeat Accumulate Codes [Divsalar, Jin, McEliece, 1998]
  - With random puncturing
- Depth 2 Parity Circuits of [G´al, Hansen, Kouck´y, Pudl´ak, and Viola, 2012]
  - Partially derandomized result
  - T = $O(n \log(n)^2)$

Explicit Codes With Time and Space Efficient Encoders

We give explicit good codes encodable

- By a uniform encoder running in time $T = n^{1+o(1)}$ and space $S = O(\log(n)^2)$
- By an explicit depth 2 parity circuit with size $n^{1+o(1)}$
  - Can be viewed as derandomization of circuits in [GHKPV, 2012]

These two statements seem incomparable

Our codes have constant distance arbitrarily close to 1

Our space efficient encoders are non-adaptive, and require random access to the input

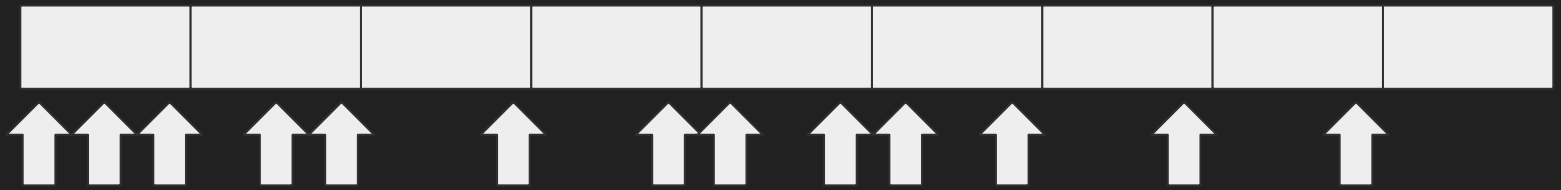
Time T And Space S Required for Encoders of Good Codes

with sequential access to the input

What is “sequential access” to the input?

Stronger than streaming

- Multiple heads
- Can only move one head one space at a time (forward or backward)
- Algorithm chooses which head to move
- Can jump heads to location of other heads



Can recognize palindromes efficiently

Why Sequential Access?

Model for access to the output of low space algorithms

Problem:

- Suppose low space algorithm A on input x outputs an O(n) bit message: A(x)
- Suppose we want to encode A(x) in code C to get C(A(x))

Solution Attempts:

- Run encoder for C, simulate A to get $A(x)_i$ when C asks for bit i
  - Small space, but slow, at least quadratic time
- Instead of simulating A from the start, store intermediate states of A
  - These are like heads in sequential access

Do Encoders (with sequential access) Require S T = $\Omega(n^2)$?

No! (for h = ω(polylog(n)))

Any good code with an encoder that runs in time T and space S with h heads has

$$hST = \Omega(n^2).$$

- This bound is tight (up to polylog(n) factors)
  - Uses a tensor code of time and space efficiently encodable codes for random access.
- If h ~ S, then $S^2T = \Omega(n^2)$
  - So almost linear time requires sqrt(n) space.
  - Not as good as random access, better than Bazzi and Mitter conjectured.

A similar result was shown for weaker streaming algorithms (h = 1) by Bangalore, Bhadauria, Hazay and Venkitasubramaniam [BBHV, 2022]

Encoder Approach

Distance of Codes as Weight

The weight of a string $y \in \{0,1\}^m$ is the number of ones in y:

$$wt(y) = \sum_{i \in [m]} y_i$$

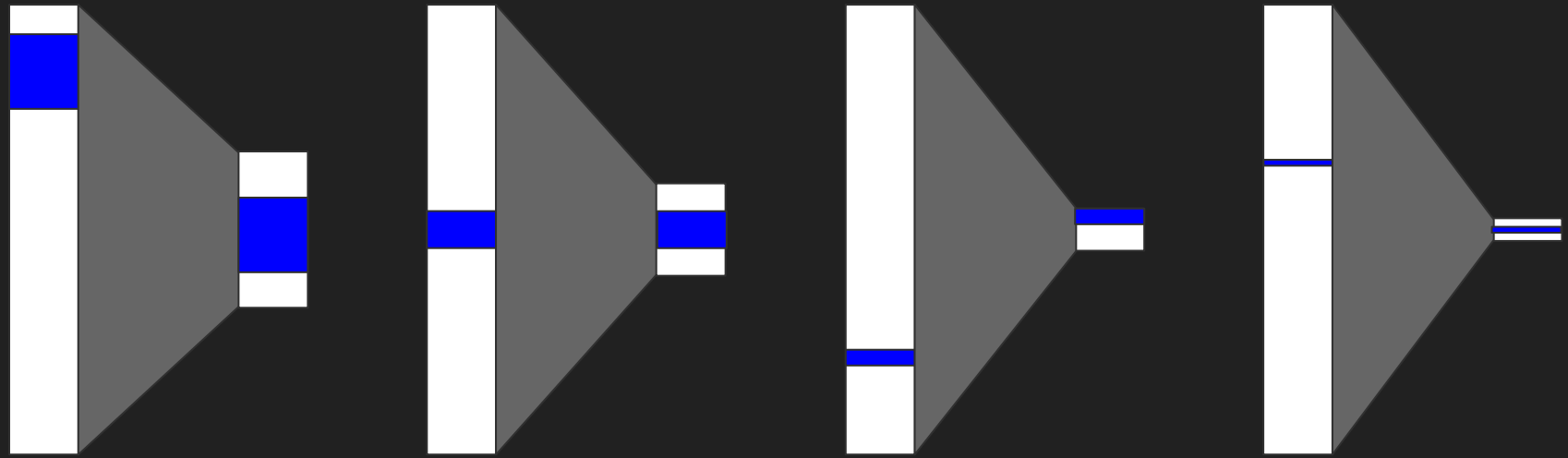
For any code $C: \{0,1\}^n \rightarrow \{0,1\}^m$ that is a linear function, the distance of C is

$$d = \min_{x \neq 0} wt(C(x))$$

So we only need to show that for all x where $wt(x) > 0$, we have $wt(C(x)) \geq d$

Suppose we knew the weight of x

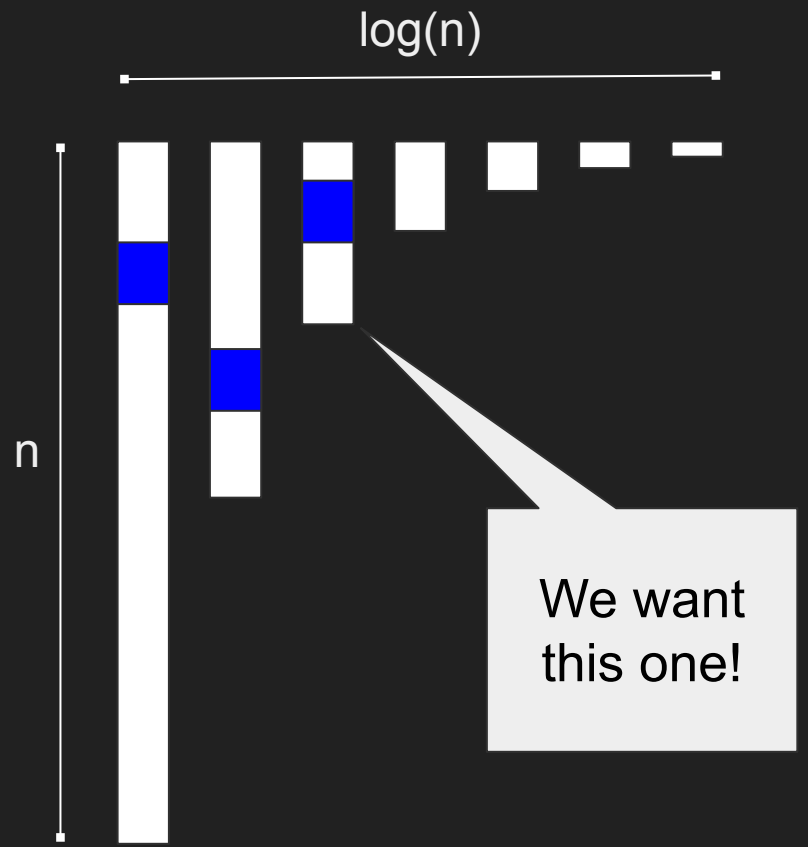
Construct a linear function that maps an input in weight range [k, 2k] to a constant relative weight δ. Call this a [k, 2k] to δ weight fixer.



Try all ranges

One output will have relative weight δ.

A small interval may have large relative weight, so can't just output this.



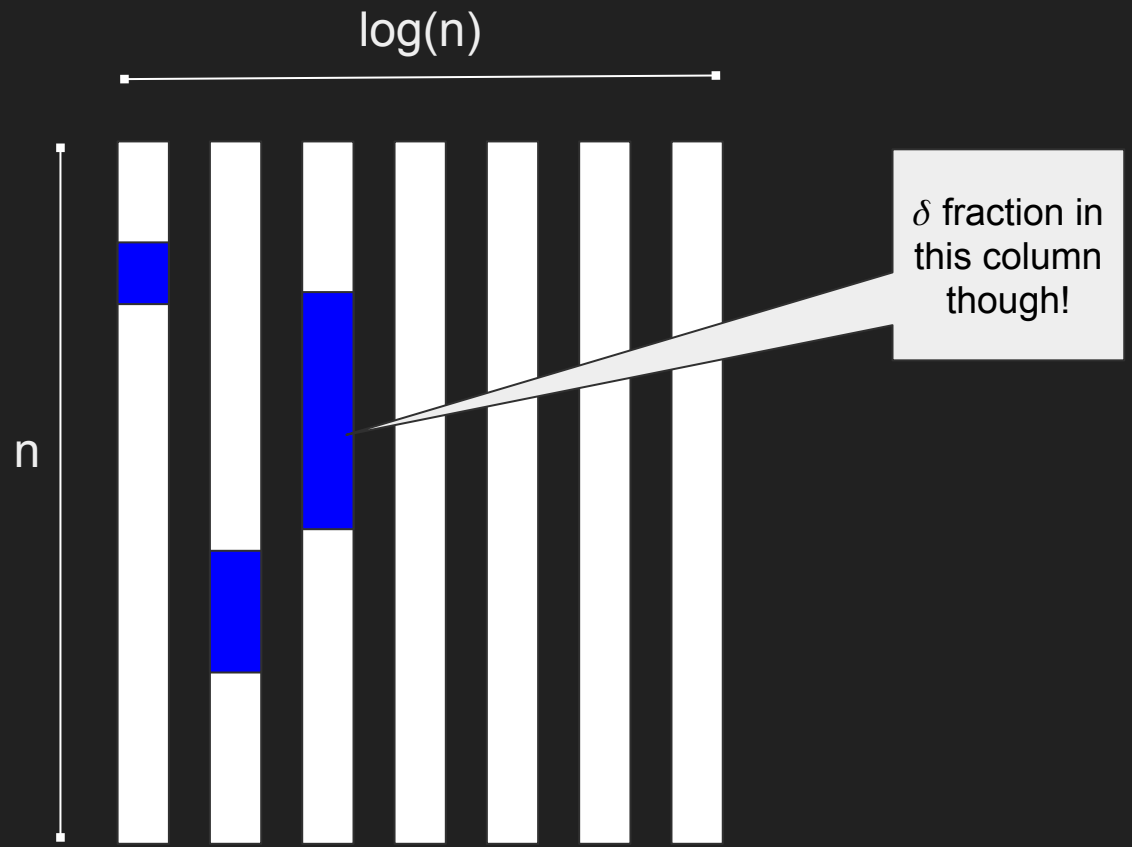
Try all ranges

One output will have relative weight δ.

A small interval may have large relative weight, so can't just output this.

Stretch all intervals to same length by repeating them.

Only has relative weight δ / log(n).



Try all ranges

One output will have relative weight δ.

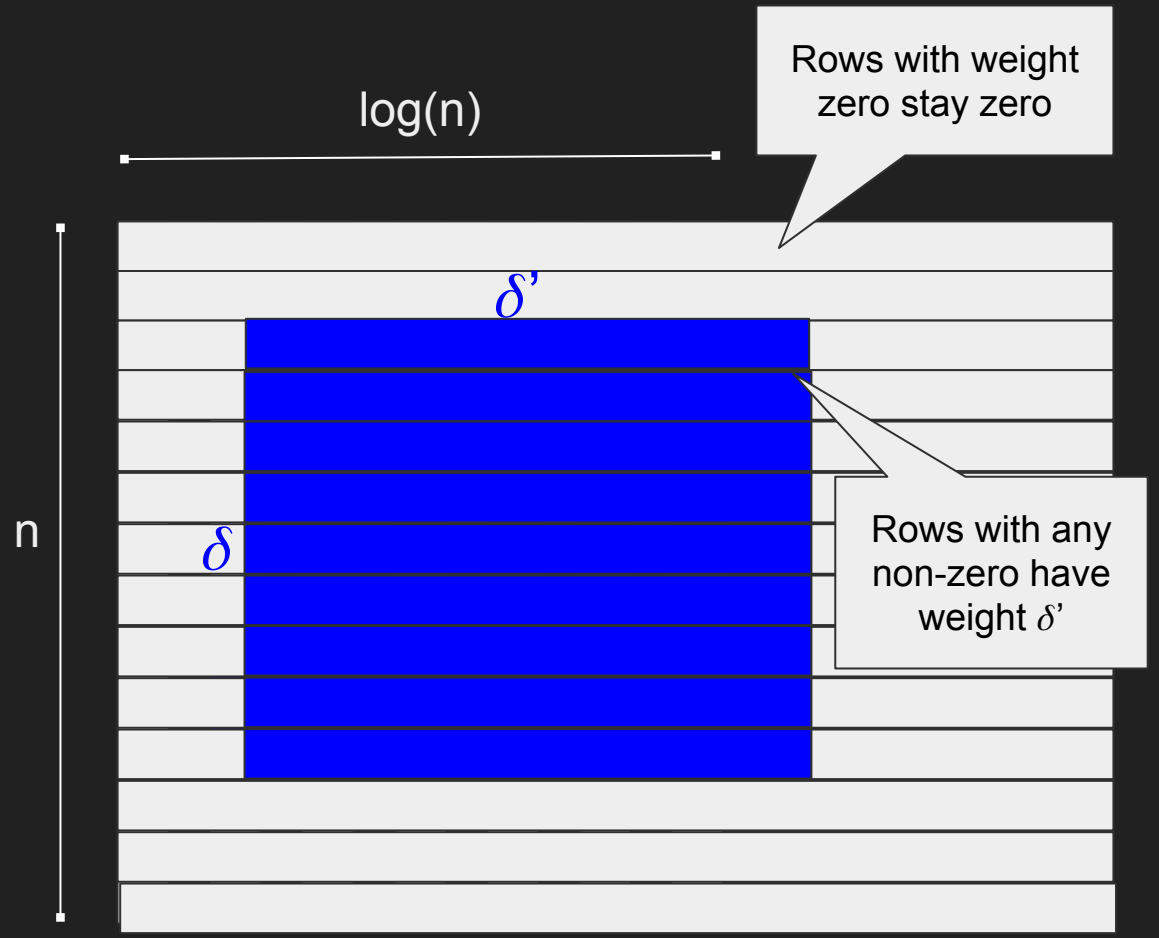
A small interval may have large relative weight, so can't just output this.

Stretch all intervals to same length by repeating them.

Only has relative weight δ / log(n).

Apply code with distance δ'

Final weight: $\delta' \delta$



Weight Fixers from Condensers

Weight fixers from condensers

Same approach used in [G´al, Hansen, Kouck´y, Pudl´ak, and Viola, 2012], but they could not construct weight fixers explicitly

We make weight fixers using lossless condensers,

AKA, bipartite lossless expanders

What is a bipartite expander?

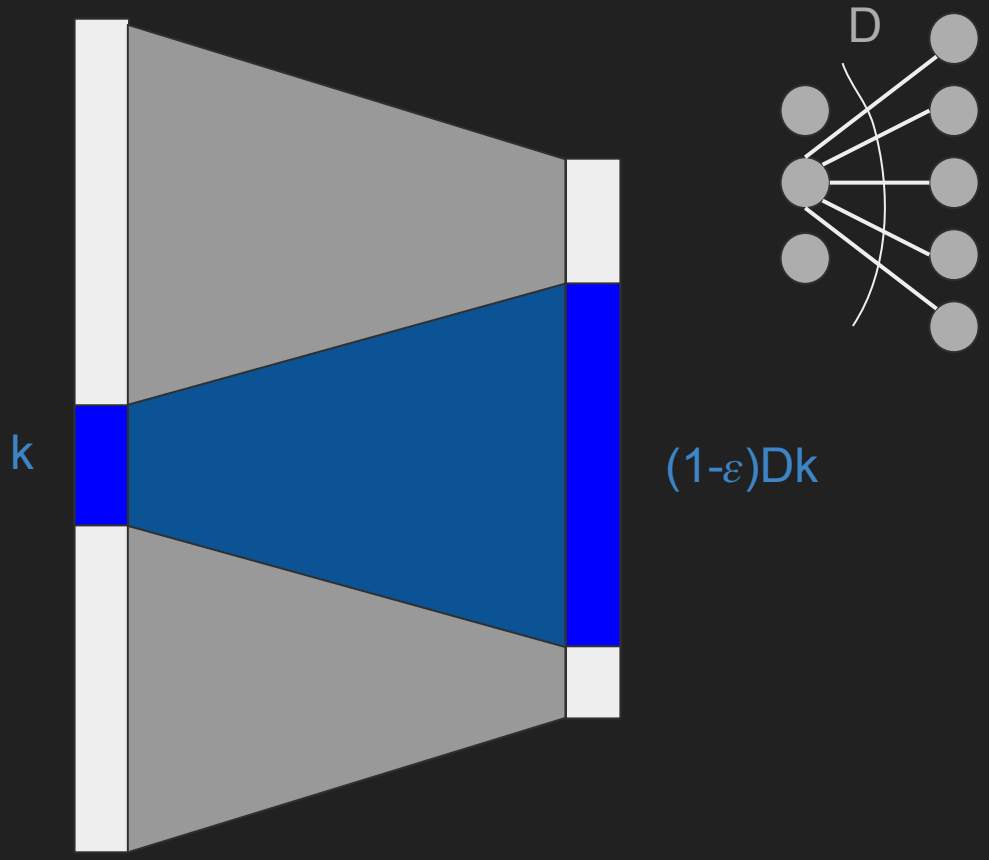
A graph such that:

Any sufficiently small subset on the left hand side

has a larger neighborhood on the right hand side.

Let D be the degree of the left hand side.

Lossless if neighborhood is near D times the size of left set.

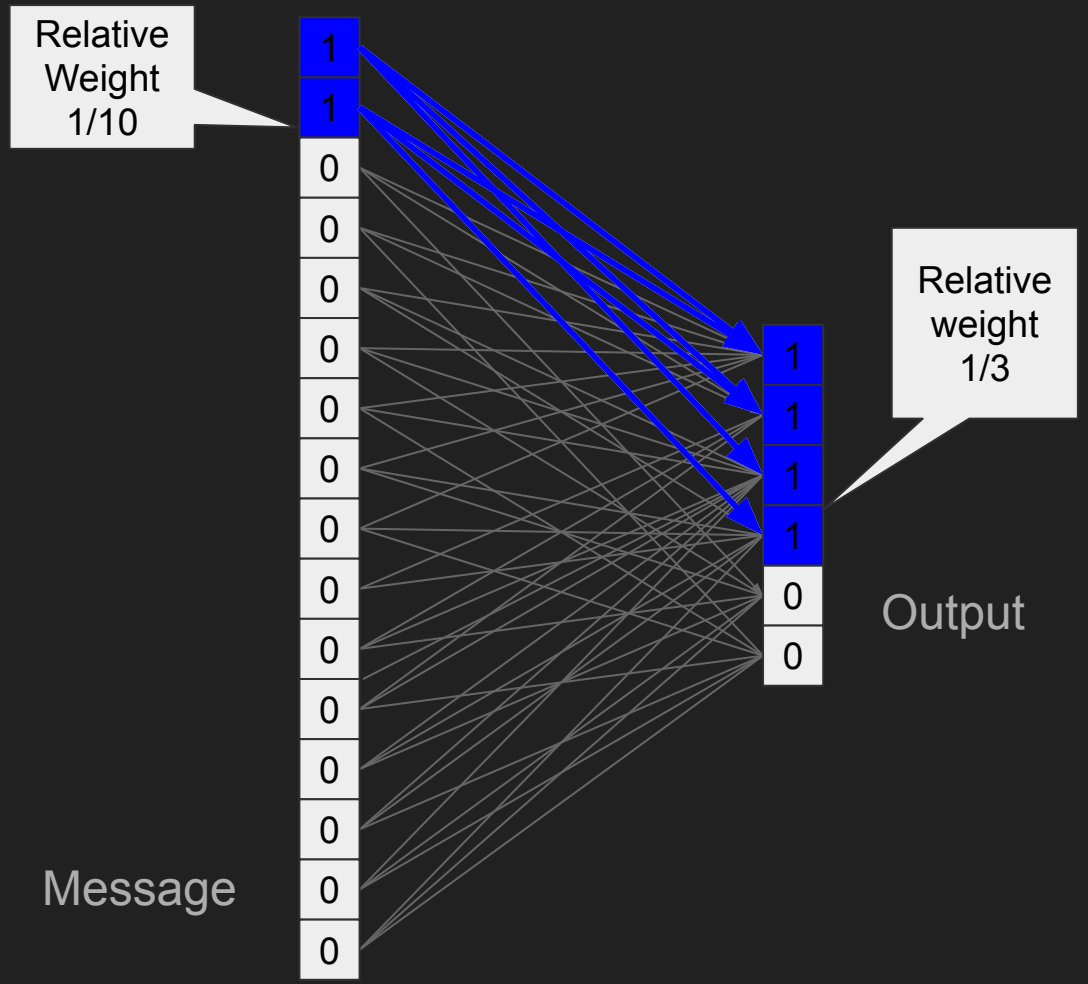


Weight Fixers From Bipartite Expanders

Identify message with left vertices, Identify output with right vertices.

Output bit is parity of adjacent input bits

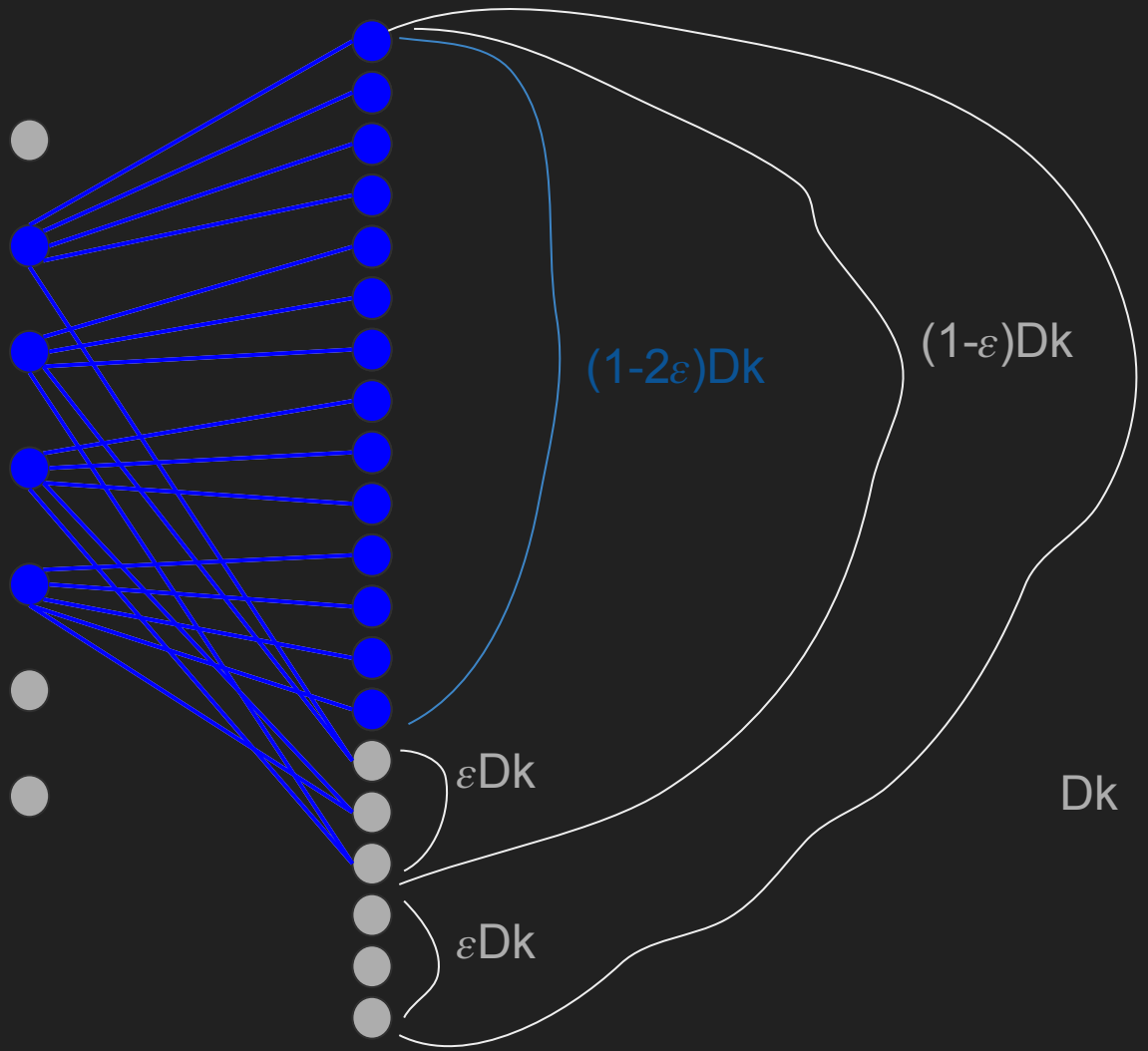
Output bit is one if a unique neighbor is one.



Weight Fixers From Lossless Expanders

If appropriate sized left hand size has many, many neighbors (it expands losslessly), then most neighbors are unique.

Left hand degree D



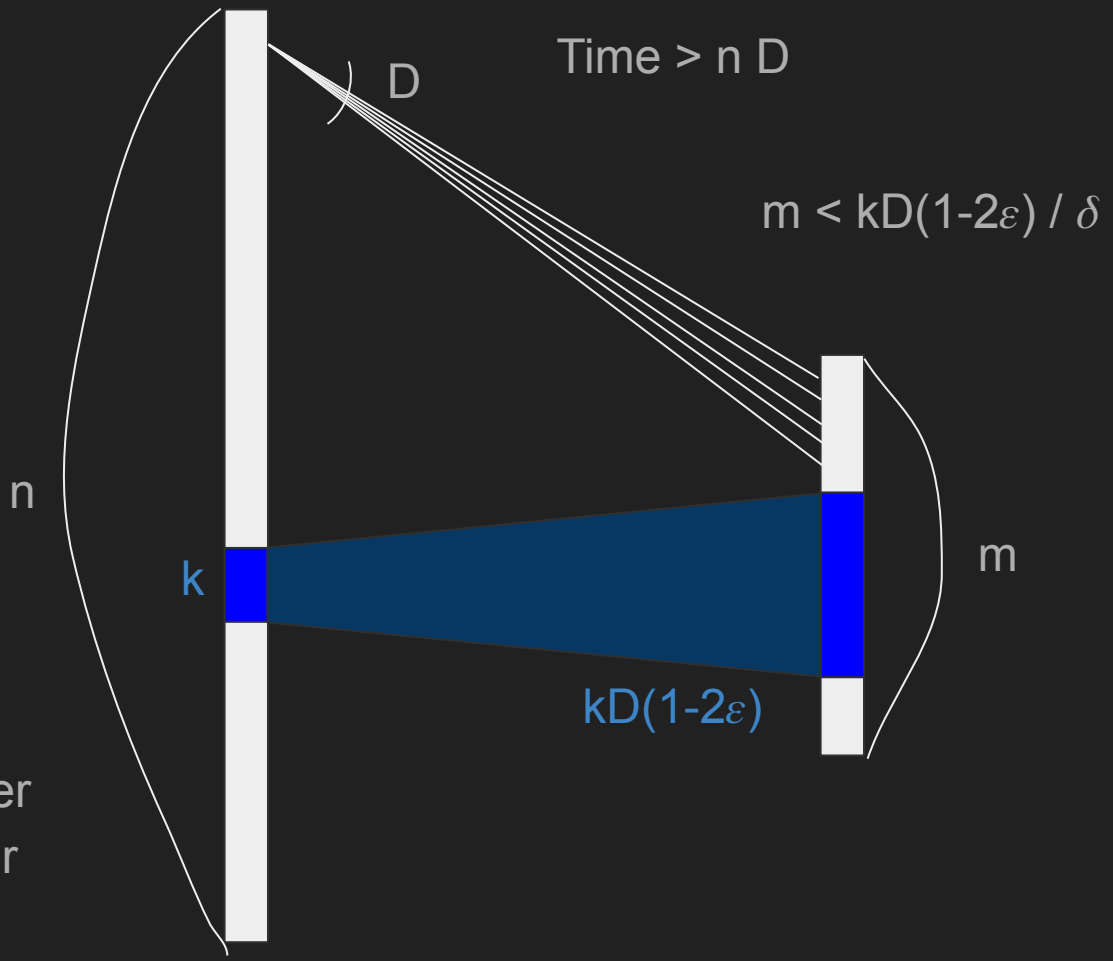
Weight Fixers From Lossless Condensers

To be efficient, need small D

But that means number of ones doesn't increase much

To get large relative weight, we need small m

This is what lossless condenser does: it condenses to a smaller space with few collisions.

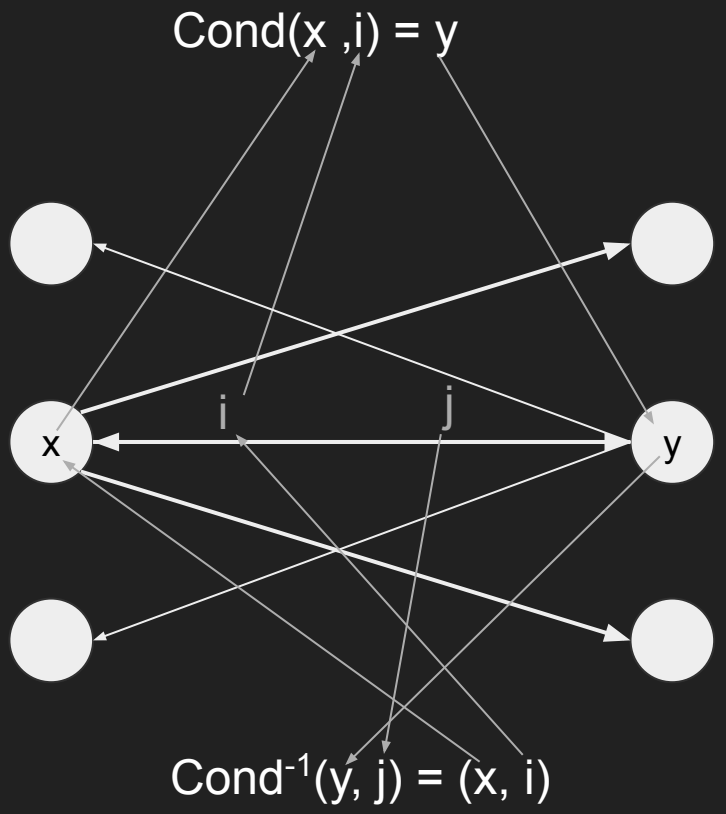


Efficiently Invertible Condensers

Generally think of condensers as a function from left hand side to right hand side.

But to be space efficient, need to map from right to left

That is, the condenser needs to be efficient to invert

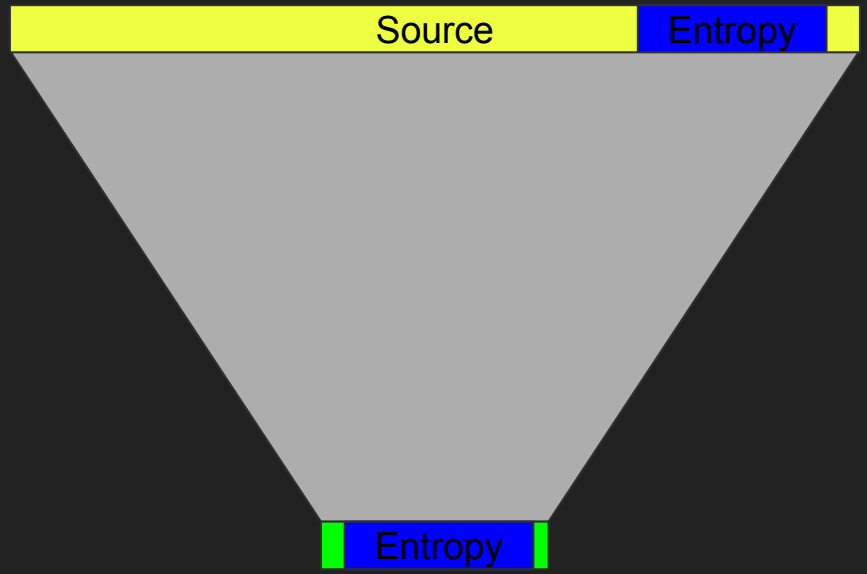


Constructing Our Lossless Condensers

Alternative View of Condensers

Functions that reduce number of bits in a source of entropy, without reducing entropy much.

Use condense and extract framework to get really good condensers



Conventional Condense And Extract

Start by condensing

Extract to further concentrate

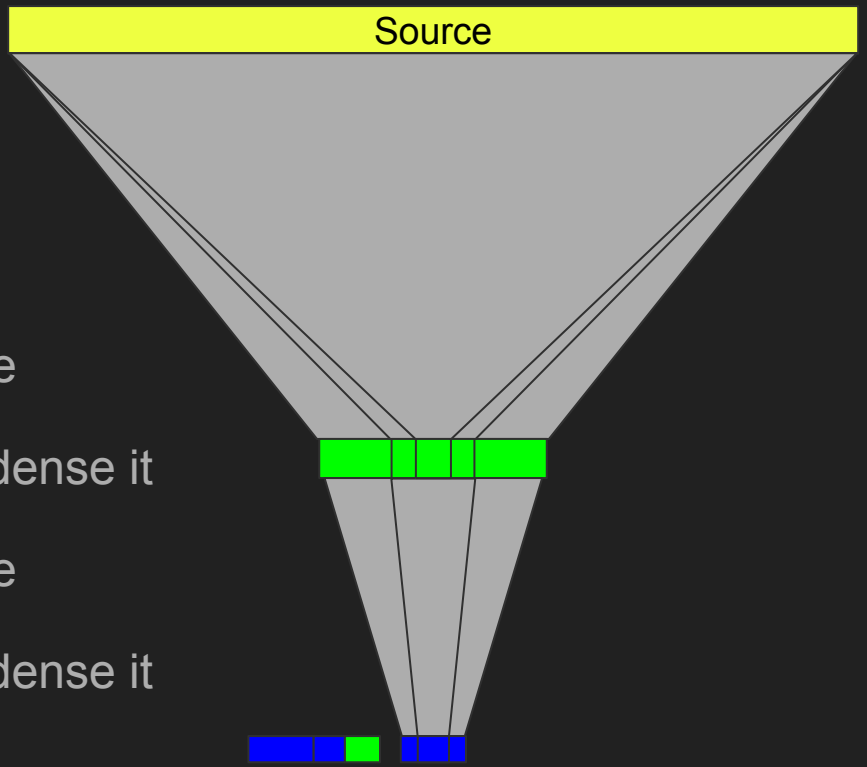
Some entropy remains, condense it

Extract to further concentrate

Some entropy remains, condense it

…

End after condense and it is a condenser

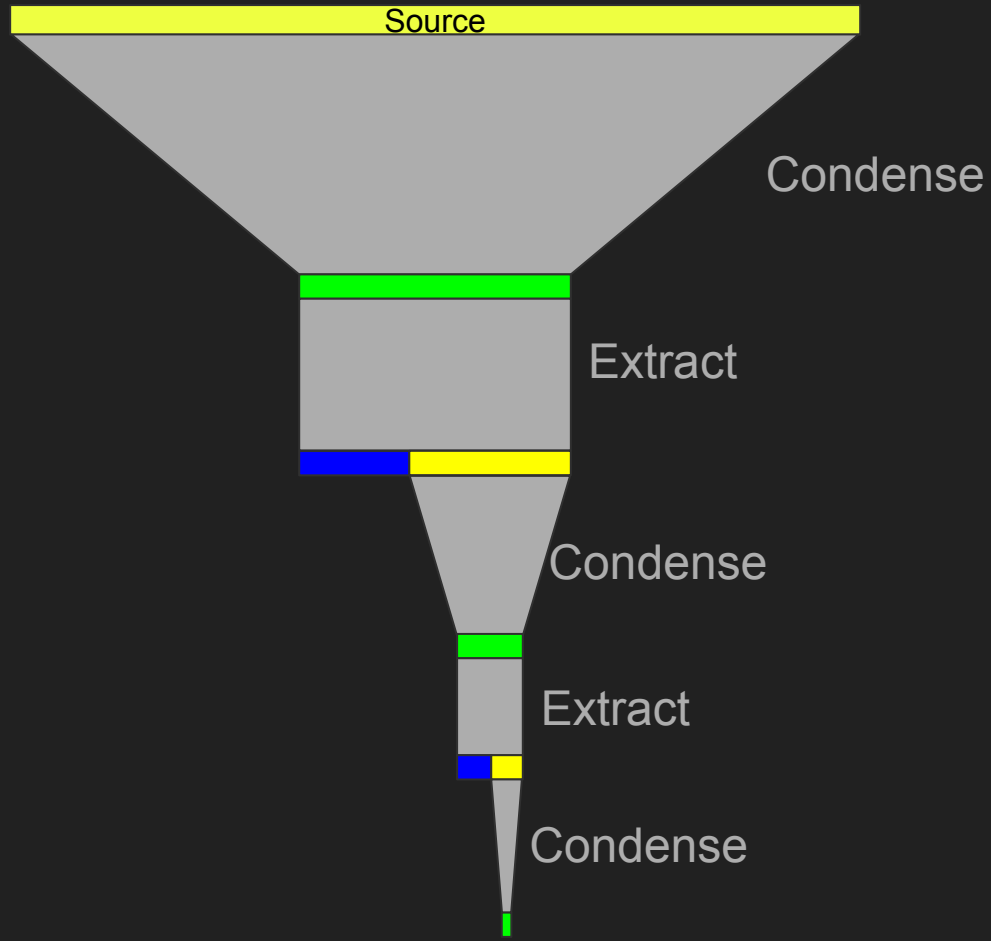


Modified to Buffered Extractors

Issue: hard to invert two functions on the same input at same time.

Solution: easy to invert two functions composed.

Make extractor output buffer, condense from buffer instead.



Constructing our Lossless Condensers

- Start with multiplicity based condensers [Kalev and Ta-Shma, 2022]
  - Not good enough
- Use a condense and extract framework [Ta-Shma, Umans, Zuckerman, 2001], [Guruswami, Umans, and Vadhan, 07]
- Use Trevisan extractor and extractor from left over hash lemma.
- Change extractors to buffered extractor and compose with the buffer
  - Similar to the lossless expanders of [Capalbo, Reingold, Vadhan, and Wigderson, 2002]

Handling large weight

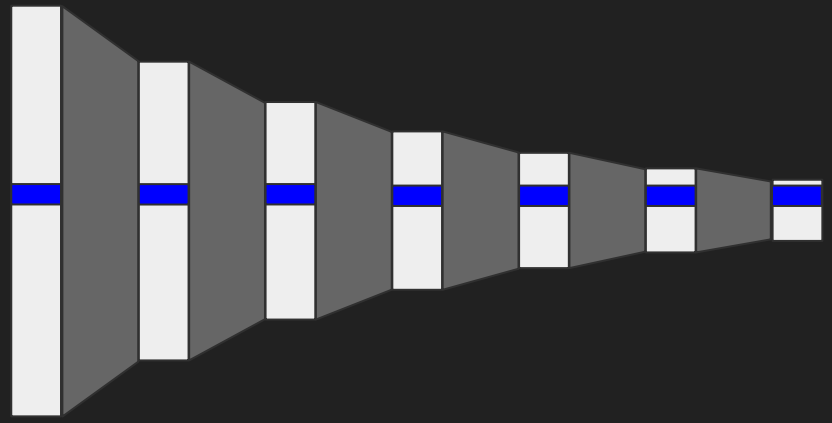
Problem: Condensers give too large outputs for input weight n / polylog(n)

Solution: Use many expanders, each increasing relative weight and decreasing size a constant factor, similar to Spielman codes [Spielman, 1995]

Why not just use this?

Time increases faster than relative weight, leads to super linear time.

Fine for few iterations.



Lower Bounds For Sequential Access

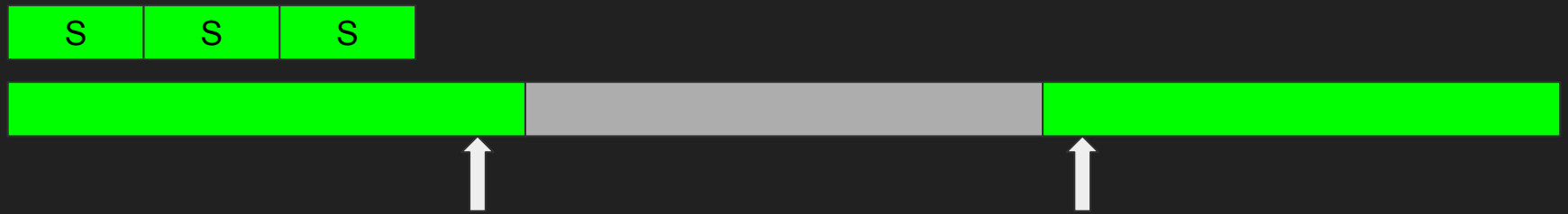
Arrange input into blocks larger than S

Idea: separate input into blocks larger than S

2 player game

- Player one sees block
- Player two outputs code most bits of the code
- Player one passes S bits when all heads leave block

If after k rounds not enough information is given, player 2 must do same thing for two settings of block. Thus code has small distance.



Lower Bound Idea

Separate input into blocks greater than S

- If block is visited few times, code can’t have good distance for that block
  - While no head is in that block,
  - Which is most blocks most of the time.
- Visiting new blocks takes a long time
  - Most blocks are visited few times
  - Excluding blocks a head was just in.

Code must have small distance!

Open Problems

Time and Space Efficient Decoding?

- Is there a code that can be decoded in time $n^{1+o(1)}$ and space $n^{o(1)}$?
  - Yes, *random* decoders for locally correctable codes [Kopparty, Meir, Ron-Zewi, and Saraf, 2016]
  - Yes, *deterministic* decoders for locally correctable codes [Cook, Moshkovitz, 2024]
- Time and space efficient **encoding** *and* **decoding** on the *same* code?
  - Time $n^{1+o(1)}$ and space $n^{1/2 + o(1)}$, achieved with tensor code.
  - Time $n^{1+o(1)}$ and space $n^{o(1)}$ is still open!

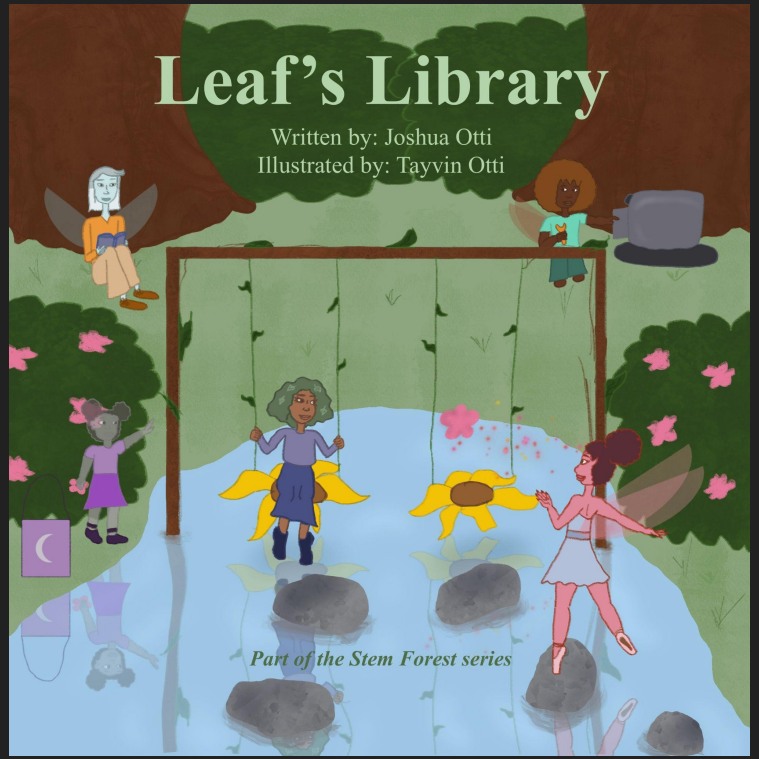
Other

- Improved Encoding Time, our time is $n\, 2^{O(\log(\log(n))^3)}$
  - Getting quasilinear time with this method requires explicit extractors with seed length $O(\log(n))$ that extract all but constantly many bits of entropy.
    - This is still open, our state of the art extractors have seed length $O(\log(n)^2)$
- Determine what code properties require $S\, T = \Omega(n^2)$
  - It is known that self dual codes do [Santhi and Vardy, 2006]
  - What about locally testable codes?
  - What about locally correctable codes?
  - etc.
- Derandomize Repeat Accumulate codes
  - As a corollary of Gal et al, our encoding technique cannot get time $o(n\, (\log(n) / \log\log(n))^2)$
  - Repeat accumulate codes, non-explicitly, achieve time $O(n \log(n))$ and space $O(\log(n))$.

Thanks for listening

Also, I wrote a picture book about binary search.

And I’m graduating soon.



https://stemforestbooks.com/leafslibrary.html