AGENDA
1 Efficiently Decoding Concatenated Codes

Recall the GOAL from last lecture:

Recall: The Concatenated Code $C_{in} \circ C_{out} = \sum_{i=1}^{n_{out}} \cdot \min e_x$ is defined by picture...

Today’s Termite Tidbit:
Some termites build their nests in trees, instead of on/under the ground. These termites can build “shelter tubes,” tunnels built out of mud and feces that run along the trunks of trees to give the colony a protected path from their nest to food on the ground.

Goal: Obtain explicit (aka, efficiently constructible), asymptotically good families of binary codes, ideally with fast algorithms.

We just saw how to use Concatenated Codes to do all of that except the “fast algorithms” part. Now we will do that.

$C_{in} \circ C_{out}$, encoding of $x$ under $C_{out}$.

$c \in \sum_{i=1}^{n_{out}} \cdot \min e_x$, encoding of $x$ under $C_{out}$.

$c' \in \sum_{i=1}^{n_{out}}$, encoding of $x$ under $C_{in}$.
**First Try at Decoding:**

1. Decode each of these blocks: that is, find the codeword \( c' \in C_{in} \) which is the closest to the received word \( c \in \Sigma_{in}^{n_{in}} \), encoding of \( x \) under \( C_{in} \).

2. Convert the “corrected” chunks \( e \in C_{in} \) into \( \Box \in \Sigma_{out} \).

3. Decode \( C_{out} \) to get the original message.

**Claim:** The above works PROVIDED that the number of errors \( e \) is \( < \frac{d_{in} \cdot d_{out}}{4} \).

**Notice:** \( d = d_{in} \cdot d_{out} \) is the designed distance of the concatenated code.

So we'd really like \( e \leq \left\lfloor \frac{d_{in} - 1}{2} \right\rfloor \), not \( d/4 \). But let's prove the claim anyway, to understand why this approach might fail.

**Let's call a block **`BAD`** if there are more than \( \left\lfloor \frac{d_{in} - 1}{2} \right\rfloor \) errors in that block.**

If there are \( e \) errors total, at most \( e/\left\lfloor \frac{d_{in}-1}{2} \right\rfloor \) blocks are \( **BAD** \).

If a block is **NOT** \( **BAD** \), then the inner code works.

Thus we win provided \( \# **BAD** \) blocks \( \leq \left\lfloor \frac{d_{in} - 1}{2} \right\rfloor \)

aka \( \frac{e}{\left\lfloor \frac{d_{in}-1}{2} \right\rfloor} \leq \left\lfloor \frac{d_{in}-1}{2} \right\rfloor \)

\( e \leq \left\lfloor \frac{d_{in}-1}{2} \right\rfloor \cdot \frac{d_{out}-1}{2} \) or \( \frac{d_{in} \cdot d_{out}}{4} \). Indeed, that's what happens when there are exactly \( \left\lfloor \frac{d_{in}}{2} \right\rfloor \) errors in each bad block.
The proof shows that this might NOT be a good idea.

If the adversary JUST BARELY messes up as many blocks as he can, this decoder will fail on $\left\lfloor \frac{d}{2} \right\rfloor$ errors.

**WHAT ARE WE LEAVING ON THE TABLE?**

Key observation: When we decode the inner code

\[ \begin{align*}
\forall e \in \Sigma^{\text{in}} & \rightarrow e \in C, \\
\text{we learn more than just } & e \in C, \\
\text{we also know } & \left( \begin{array}{c} \forall e \in \Sigma^{\text{in}} \\ e \in C 
\end{array} \right) 
\end{align*} \]

**SOME MOTIVATING EXAMPLES:**

1. Each block either has 0 or $\frac{d}{2}$ errors. [This is the bad example from before.]

   - Each block has $\frac{d}{2}$ errors.
   - Correct each block of $C_{\text{in}}$.
   - These blocks have no errors and don't change when we decode them.
   - This block had some errors: When we decode it, it's to something at least $\frac{d}{2}$ away, because:
     - $e$ is $\frac{d}{2}$ away.
     - This distance is also $\frac{d}{2}$.

   Thus, even though the $\boxed{\text{blocks}}$ are incorrect, we can detect that they were incorrect.

So the thing we should do in this case is treat the $\boxed{\text{blocks}}$ as ERASURES. We can handle twice as many of those! So our error tolerance is actually about $\frac{d}{2}$ in this case, which is what we wanted.
Motivating Example #2. The bad guy tries to foil our previous example by adding error $d_{in}$ to some blocks, turning them into other codewords.

The bad guy tries to foil our previous example by adding error $d_{in}$ to some blocks, turning them into other codewords.

Now we can't detect anything! BUT, there are only $e/d_{in}$ corrupted blocks. Again we save a factor of 2 and can correct up to $e = d/2$ errors.

We would like to interpolate between these two extremes.

**Claim.** We can efficiently decode $RS_t(n,k)$ from $e$ errors and $s$ erasures, as long as $2e + s < n-k+1$. (aka, the distance of the RS code).

pf-ish. Throw out the $s$ erasures. You are left with $RS(n-s,k)$.

Since $2e < (n-s) - k + 1$, run Berlekamp-Welch to correct the errors. Now you have an $RS(n,k)$ codeword w/ $s$ erasures. Since $s < n-k+1$, correct the erasures (via linear algebra).

This inspires the following algorithm:

**Algorithm:** (Not the final version).

Given $\hat{w} = (w_1, w_2, ..., w_{n_{out}}) \in \left(\mathbb{F}_{q}^{n_{in}}\right)_{s_{out}}^s$, s.t. $\Delta(w_i,c) < \frac{d_{in} \cdot d_{out}}{2}$ for some $c \in C_{in} \circ C_{out}$

For each $i = 1, ..., n_{out}$:

Let $w_i^* = \arg \min_{y \in C_{in}} (\Delta(y,w_i))$

With probability $\min\left(\frac{2\Delta(w_i,w_i^*)}{d_{in}}, 1\right)$:

L Let $b_i = 1$

Else:

L Set $b_i$ s.t. $E_{in}(y_i) = w_i^*$

Run $C_{out}$'s (error, erasure) decoder on $(b_1, ..., b_{n_{out}})$, RETURN the result.
Why does this algorithm work?

Let $e_i = \lambda(W_i, C_i)$, so

$$E[2X_i, X_i] \leq 2e_i$$

Proof (sketch), let $e_i = \lambda(W_i, C_i)$, so

$$e = \sum e_i \leq 2 \cdot \sum e_i$$

Given the subclaim, $E[2X_i, X_i] = 2(1 - E[X_i^2])$, since if we didn't find $J$, then we

$$E[X_i] = \frac{1}{2} \cdot \min \{2(\Delta(w, w+1)), \Delta(w, w+2)\}$$

Case 1. $C = w+1$. As before, $E[X_i] = \frac{1}{2} \cdot \min \{2(\Delta(w, w+1)), \Delta(w, w+2)\}$, and

$$E[X_i] = \frac{1}{2} \cdot \min \{2(\Delta(w, w+1)), \Delta(w, w+2)\}$$

Case 2. $C = w$. So $X_i = 0$, and $E[X_i] \leq 2 \cdot \min \{2(\Delta(w, w+1)), \Delta(w, w+2)\}$ as desired.
Subsubclaim. If $c_i \neq w_i'$, $2e_i + \min(2\Delta(w_i, w_i'), d_{in}) \geq 2 \cdot d_{in}$

Proof: Suppose that $2\Delta(w_i, w_i') \leq d_{in}$. Then the subsubclaim reads:

$$2e_i + 2\Delta(w_i, w_i') \geq 2d_{in}$$

$$e_i + \Delta(w_i, w_i') \geq d_{in}$$

$$\Delta(w_i, c_i) + \Delta(w_i, w_i') \geq d_{in}$$

which is true since

$$d_{in} \leq \Delta(c_i, w_i') \leq \Delta(w_i, c_i) + \Delta(w_i, w_i')$$

$\uparrow$ since $c_i, w_i' \in C_{in}$ and $c_i \neq w_i'$

On the other hand, if $d_{in} < 2 \Delta(w_i, w_i')$, then the subsubclaim reads:

$$2e_i + d_{in} \geq 2d_{in} \text{ aka } e_i \geq \frac{d_{in}}{2}.$$ 

But this must be true because we are in the setting where $c_i \neq w_i'$. Indeed, if $e_i < \frac{d_{in}}{2}$, then the inner code's decoder would have worked correctly and we would have $c_i = w_i'$.

Sorry for botching this argument in class! And thanks to Shaqun for helping me fix it.
So the CLAIM implies that the algorithm works "in expectation."

We could try to turn this into a high probability result (repeat a bunch of times), but instead we will actually be able to DERANDOMIZE it.

**STEP 1.** We will reduce the necessary randomness by a little bit.

**ALGORITHM VERSION 2**

Given \( \vec{w} = (w_1, w_2, ..., w_{n_{out}}) \in \left( F_{\frac{d_{in}}{d_{in}}} \right)^{n_{out}} \) s.t. \( \Delta(w, e) < \frac{d_{in} \cdot d_{out}}{2} \) for some \( e \in C_{in} \cap C_{out} \)

**CHOOSE \( \Theta \in [0, 1] \) UNIFORMLY AT RANDOM.**

For each \( i = 1, ..., n_{out} \):

- Let \( \omega_i = \arg\min_{y \in C_{in}} (\Delta(y, \omega_i)) \)

**IF \( \Theta \leq \min \left( \frac{2 \Delta(\omega_i, \omega_i')}{d_{in}}, 1 \right) :**

  - Set \( \beta_i = 1 \)

**ELSE:**

  - Set \( \beta_i \) s.t. \( E_{in}(y_i) = \omega_i \)

Run \( C_{out} \)'s (error+erasure) decoder on \( (\beta_1, ..., \beta_{n_{out}}) \), RETURN the result.

That is, we never used the fact that our draws for \( \beta_i \) were independent. So let's make them not at all independent.

Our next step will be to search over all possible \( \Theta \)'s. In fact, we only need to look at \( n_{out} + 2 \) values of \( \Theta : \)

\[
\Theta_0 \rightarrow \Theta_1 \rightarrow \min \left( \frac{2 \Delta(\omega_i, \omega_i')}{d_{in}}, 1 \right) \rightarrow \Theta_3 \rightarrow \min \left( \frac{2 \Delta(\omega_i, \omega_i')}{d_{in}}, 1 \right) \rightarrow \Theta_{n_{out} + 1}
\]

All the \( \Theta \) values \( \omega_i \) in the same interval before the same.
This is called Forney’s Generalized Minimum Distance Decoder.

Algorithm: **Final Version**

Given $\hat{w} = (w_1, w_2, \ldots, w_{n_{out}}) \in \left( F_{b_{in}}^{n_{in}} \right)^{n_{out}}$, s.t. $\Delta(w, c) < \frac{d_{in} \cdot d_{out}}{2}$ for some $c \in C_{in} \cdot C_{out}$

**Compute the $n_{out} + 2$ relevant values of $\Theta$, $\Theta_0, \ldots, \Theta_{n_{out} + 1}$**

For $j = 0, \ldots, n_{out} + 1$:

For each $i = 1, \ldots, n_{out}$:

Let $\omega_i = \arg\min_{y \in C_{in}} \Delta(y, w_i)$

**IF** $\Theta_j < \min \left( 2 \frac{\Delta(\omega_i, w_i)}{d_{in}}, 1 \right)$:

- Set $\beta_i = 1$

**Else**:

- Set $\beta_i$, s.t. $E_{in}(y_c) = \omega_i$

Run $C_{out}$’s (error+erasure) decoder on $(\beta_1, \ldots, \beta_{n_{out}})$, to obtain $\hat{x}$

**IF** $\Delta(Enc(\hat{x}), w) \leq \left\lfloor \frac{d_{in} - 1}{2} \right\rfloor$:

**RETURN** $\hat{x}$

The fact that this algorithm is correct follows from our earlier claim. Since $\sum_{\Theta} \left( 2 \cdot (\text{#errors}) + (\text{#erasures}) \right) \leq d_{out}$, there exists some $\Theta \in [0, 1]$ so that $2(\text{#errors}) + (\text{#erasures}) \leq d_{out}$, aka so that the alg. finds the correct $\hat{x}$.

Thus, our algorithm above, which tries ALL values of $\Theta$, must find that good value and return the correct answer.
What is the running time of this algorithm?

Depends on the codes. Let’s choose our explicit construction from last time:

Recall we had \( n_{out} = q_{out} - 1 \), and \( q_{out} = 2^{k_{in}} \).

The expensive bits of the alg are:

For \( O(n_{out}) \) choices of \( \Theta \):

For \( n' = 1, \ldots, n_{out} \):

\[ \text{\cdot Decode the inner code \( (\text{length } n_{in} = O(K_{in}) = O(\log(n_{out})) \)} \]

\[ \text{\quad by brute force \quad // Time } O(n_{in} \cdot 1_{Cin}) = O(n_{in} \cdot 2^{k_{in}}) = \text{poly}(n) \]

\[ \text{\cdot Run the RS decoder \quad // Time } \text{poly}(n) \]

So altogether the whole thing runs in polynomial time.

We have proved

**THM** For every \( RG(0,1) \), there is a family \( C \) of explicit binary linear codes that lies at or above the Zyablov bound. Further, \( C \) can be decoded from errors up to half the Zyablov bound in time \( \text{poly}(n) \).

AKA, we have achieved our goal! Hooray!
To RECAP the story of Concatenated Codes:

- We considered \((RS \text{ code}) \circ (\text{Binary Linear Code on the GV bound})\)

- Because the inner code is so small, we can find a good one by brute force in time \(\text{poly}(n)\).

- We can be a little more clever with the Justesen Code, if we want something asymptotically good and STRONGLY explicit.

- \((RS) \circ (\text{Binary code on the GV bd})\) met the “Zyablov Bound,” which was defined as “the bound that these codes meet.”

- We saw how to use Forney’s GMD decoder to efficiently decode these codes up to half the minimum distance.

QUESTIONS to PONDER:

1. When does code concatenation give distance STRICTLY LARGER than \(d_{in} \cdot d_{out}\)?

2. Do there exist concatenated codes on the GV bound?
   
   **SPOILER ALERT:** YES, see [Thomesson 1983]. (It’s a randomized construction)

3. Can we decode these \(^\ast\) efficiently?
   
   **SPOILER ALERT:** ALSO YES. It uses list decoding, we may see it later.

4. Can you do better than the Zyablov bound for EXPLICIT CODES with EFFICIENT ALGS?