15-853:Algorithms in the Real World

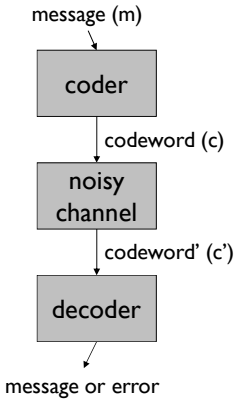
Error Correcting Codes I
- Overview, Hamming Codes, Linear Codes

Error Correcting Codes II
- Reed-Solomon and Concatenated Codes

Error Correcting Codes III (LDPC/Expander Codes)
- Expander Graphs
- LDPC (Expander) codes
- Tornado codes

Error Correcting Codes IV (RS decoding, Number thy)

Block Codes



Each message and codeword is of fixed size

Σ = codeword alphabet

k =|m| **n** = |c| **q** = $|\Sigma|$

C $\subseteq \Sigma^n$ (codewords)

Δ**(x,y)** = number of positions s.t. $x_i \neq y_i$

d = min{Δ(x,y) : x,y$\in$ C, x $\neq$ y}

Code described as: $\mathbf{(n,k,d)_q}$

Linear Codes

If Σ is a field, then Σ^n is a vector space

Definition: C is a linear code if it is a linear subspace of Σ^n of dimension k.

This means that there is a set of k independent vectors $v_i \in \Sigma^n$ $(1 \le i \le k)$ that span the subspace.

i.e. every codeword can be written as:

$c = a_1 v_1 + a_2 v_2 + \ldots + a_k v_k$ where $a_i \in \Sigma$

"Linear": the sum of two codewords is a codeword.

Minimum distance = weight of least-weight codeword

Generator and Parity Check Matrices

Generator Matrix:

A k x n matrix **G** such that: C = { xG | x $\in \Sigma^k$ }

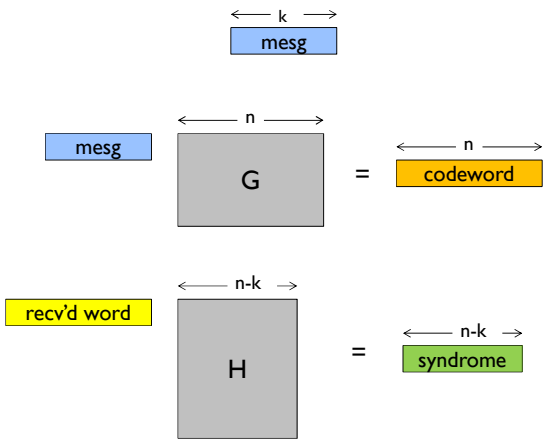
Made from stacking the spanning vectors

Parity Check Matrix:

An (n – k) x n matrix **H** such that: C = {y $\in \Sigma^n$ | $Hy^T = 0$}

(Codewords are the null space of H.)

These **always exist for linear codes**



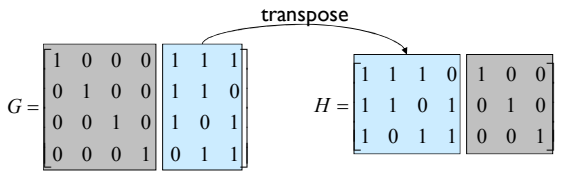
if syndrome = 0, received word = codeword

else have to use syndrome to get back codeword

Relationship of G and H

For linear codes, if G is in standard form $[I_k \; A]$
then $H = [-A^T \; I_{n-k}]$

Example of (7,4,3) Hamming code:



Examples of Codes

Hamming codes are binary $(2^r\text{-}1\text{–}1, 2^r\text{-}1\text{-}r, 3)$ codes.
Basically (n, n – log n, 3)
Hadamard codes are binary $(2^r\text{-}1, r, 2^{r-1})$.
Basically (n, log n, n/2)

Reed Solomon codes are $(n, k, n\text{-}k\text{+}1)_n$
Optimal but large alphabet
Concatenated codes can get best of both worlds

Today: another set of codes, optimized for fast (de)coding.

Why Expander Based Codes?

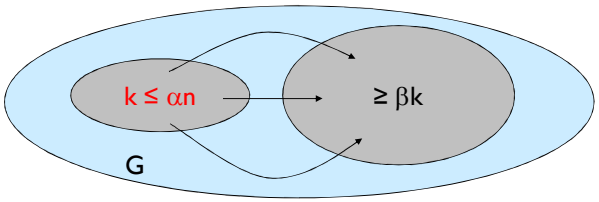
These are linear codes like RS & random linear codes

RS/random linear codes give good rates but are slow:

Code	Encoding	Decoding
Random Linear	$O(n^2)$	$O(n^3)$
RS	O(n log n)	$O(n^2)$
LDPC	$O(n^2)$ or better	O(n)
Tornado	O(n log 1/ε)	O(n log 1/ε)

Assuming an $(n, (1\text{-}p)n, (1\text{-}\varepsilon)pn\text{+}1)_2$ tornado code

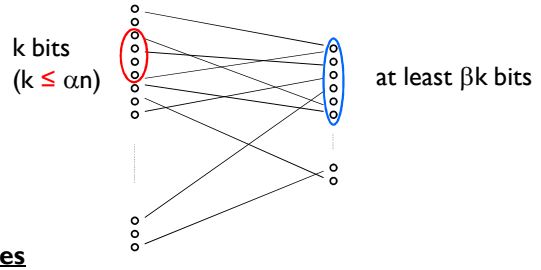
(α, β) Expander Graphs (non-bipartite)



Properties

- **Expansion:** every small subset (k ≤αn) has many (≥ βk) neighbors
- **Low degree –** not technically part of the definition, but typically assumed

(α, β) Expander Graphs (bipartite)



Properties

- **Expansion:** every small subset (k ≤αn) on left has many (≥βk) neighbors on right
- **Low degree –** not technically part of the definition, but typically assumed

Expander Graphs

Useful properties:

- Every set of vertices has many neighbors
- Every balanced cut has many edges crossing it
- A random walk will quickly converge to the stationary distribution (rapid mixing)
- Expansion is related to the eigenvalues of the adjacency matrix

Expander Graphs: Applications

Pseudo-randomness: implement randomized algorithms with few random bits

Cryptography: strong one-way functions from weak ones.

Hashing: efficient n-wise independent hash functions

Random walks: quickly spreading probability as you walk through a graph

Error Correcting Codes: several constructions

Communication networks: fault tolerance, gossip-based protocols, peer-to-peer networks

d-regular graphs

An undirected graph is **d-regular** if every vertex has d neighbors.

A bipartite graph is **d-regular** if every vertex on the left has d neighbors on the right.

We consider only d-regular constructions.

Expander Graphs: Constructions

Important parameters:size (n), degree (d), expansion (β)

Randomized constructions

- A random d-regular graph is an expander with a high probability
- Construct by choosing d random perfect matchings
- Time consuming and cannot be stored compactly

Explicit constructions

- Cayley graphs, Ramanujan graphs etc
- Typical technique – start with a small expander, apply operations to increase its size

Expander Graphs: Constructions

Theorem: for every constant $0 < c < 1$, can construct bipartite graphs with

n nodes on left,
cn on right,
d-regular,

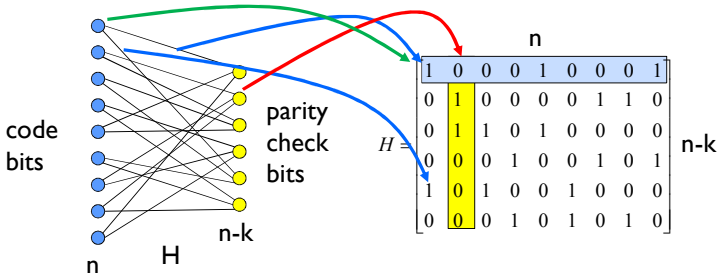
that are $(\alpha, 3d/4)$ expanders, for constants α and d that are functions of c alone.

"Any set containing at most alpha fraction of the left has (3d/4) times as many neighbors on the right"

From expanders to codes

LDPC CODES

Low Density Parity Check (LDPC) Codes



Each row is a vertex on the right and each column is a vertex on the left.

A codeword on the left is valid if each right "parity check" vertex has parity 0.

The graph has O(n) edges (**low density**)

Applications in the real world

10Gbase-T (IEEE 802.3an, 2006)

- Standard for 10 Gbits/sec over copper wire

WiMax (IEEE 802.16e, 2006)

- Standard for medium-distance wireless. Approx 10Mbits/sec over 10 Kilometers.

NASA

- Proposed for all their space data systems

History

Invented by Gallager in 1963 (his PhD thesis)

Generalized by Tanner in 1981 (instead of using parity and binary codes, use other codes for "check" nodes).

Mostly forgotten by community at large until the mid 90s when revisted by Spielman, MacKay and others.

Distance of LDPC codes

Consider a d-regular LPDC with (α,3d/4) expansion.

Theorem: Distance of code is greater than αn.

Proof. (by contradiction)

Linear code; distance= min weight of non-0 codeword.

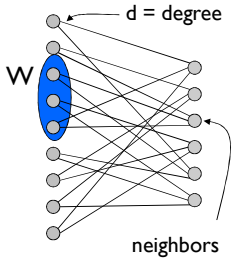
Assume a codeword with weight w ≤ αn.

Let W be the set of 1 bits in codeword

It has >3dw/4 neighbors on the right

Average # of 1s per such neighbor is < 4/3.

So at least one neighbor sees a single 1-bit. Parity check would fail!



Correcting Errors in LPDC codes

We say a vertex is *unsatisfied* if parity ≠ 0

Algorithm:

While there are unsatisfied check bits

1. Find a bit on the left for which more than d/2 neighbors are unsatisfied
2. Flip that bit

Converges since every step reduces unsatisfied nodes by at least 1.

Runs in linear time.

Why must there be a node with more than d/2 unsatisfied neighbors (if we're not at a codeword)?

Correcting Errors in LPDC codes

Theorem: Always exists a node > d/2 unsatisfied neighbors if we're not at a codeword.

Suppose not. (Let d be odd.) Let S be the corrupted bits.

Each such bit has majority of satisfied neighbors

(sat. neighbors see at least two corrupted bits on left)

(unsat. neighbors may see only one corrupted bit on left)

Each corrupt bit give $1 to each unsat nbr, $½ to sat nbr.

Total money given < 3d/4 |S|.

Each node in N(S) collects $1 at least.

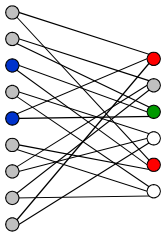
So |N(S)| < 3d/4 |S|. Contradicts expansion.

Coverges to closest codeword

Theorem: Assume (α,3d/4) expansion. If # of error bits is less than αn/4 then simple decoding algorithm converges to closest codeword.

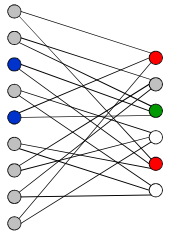
Proof: let:

- u_i = # of unsatisfied check bits on step i
- r_i = # corrupt code bits on step i
- s_i = # satisfied check bits with corrupt neighbors on step i



We know that u_i decrements on each step, but what about r_i?

Proof continued:



- u_i = unsatisfied
- r_i = corrupt
- s_i = satisfied with corrupt neighbors

$u_i + s_i > \frac{3}{4} dr_i$ (by expansion)

$2s_i + u_i \le dr_i$ (by counting edges)

$\frac{1}{2} dr_i \le u_i$ (by substitution)

$u_i < u_0$ (steps decrease u) $u_0 \le dr_0$ (by counting edges)

Therefore: $r_i < 2r_0$ i.e. number of corrupt bits cannot more than double

If we start with at most αn/4 corrupt bits we will never get αn/2 corrupt bits --- but the distance is αn. So converge to closest codeword.

More on decoding LDPC

Simple algorithm is only guaranteed to fix half as many errors as could be fixed but in practice can do better.

Fixing (d-1)/2 errors is NP hard

Soft "decoding" as originally specified by Gallager is based on belief propagation---determine probability of each code bit being 1 and 0 and propagate probs. back and forth to check bits.

Encoding LPDC

Encoding can be done by generating G from H and using matrix multiply. (Remember, c = xG).

What is the problem with this?

Various more efficient methods have been studied

Let's see one approach to efficient coding and decoding.

TORNADO CODES

Luby Mitzenmacher Shokrollahi Spielman 2001

The random erasure loss model

Erasure Model:

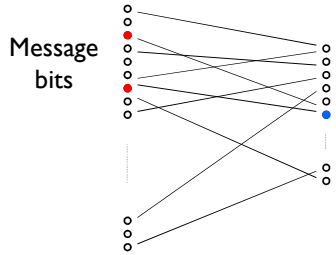
- Each bit either reaches intact, or is lost.
- We know the positions of the lost bits.

Shannon's Noise Model:

- Each bit is erased with some probability p (say ½ here)

Makes life easier for the explanation here.

Can be extended to worst-case error, and bit corruption with extra effort. [see e.g., Spielman1996]



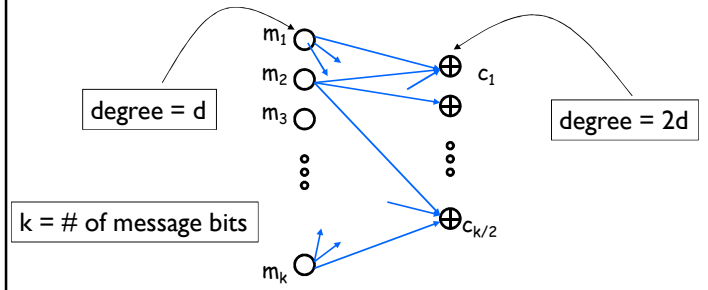
$c_6 = m_3 \oplus m_7$

Similar to standard LDPC codes but check bits are not required to equal zero.
(i.e., the graph does not represent H anymore).

Tornado codes

Have d-regular bipartite graphs with n nodes on the left and n/2 on the right.

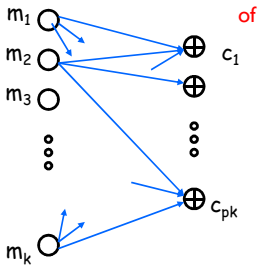
Let's again assume 3d/4-expansion.



Tornado codes: Encoding

Why is it linear time?

Computes the sum modulo 2 of its neighbors

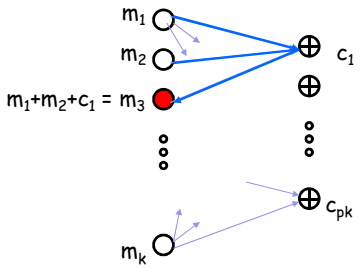


Tornado codes: Decoding

First, assume that all the check bits are intact

Find a check bit such that only one of its neighbors is erased (an *unshared neighbor*)

Fix the erased bit, and repeat.



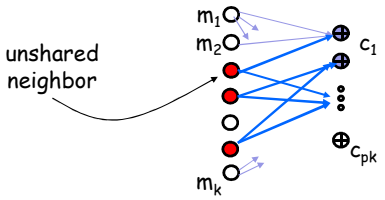
Tornado codes: Decoding

Want to always find such a check bit with the following "Unshared neighbors" property.

Consider the set of corrupted message bit and their neighbors.

Suppose this set is small.

=> at least one message bit has an unshared neighbor.



Tornado codes: Decoding

Can we always find unshared neighbors?

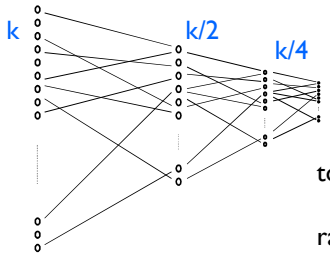
Expander graphs give us this property if expansion > d/2 (similar argument to one above)

Also, [Luby et al] show that if we construct the graph from a specific kind of degree sequence, then we can always find unshared neighbors.

What if check bits are lost?

Cascading

- Use another bipartite graph to construct another level of check bits for the check bits
- Final level is encoded using RS or some other code



stop when $k/2^t$ "small enough"

total bits $n \le k(1 + ½ + ¼ + \ldots)$
$= 2k$

rate = $k/n = ½$.

Cascading

Encoding time

- for the first k stages : |E| = d x |V| = O(k)
- for the last stage: poly(last size) = O(k) by design.

Decoding time

- start from the last stage and move left
- again proportional to |E|

So get very fast (linear-time) coding and decoding.

Construction for expander graphs

SOME EXTRA SLIDES

Expander Graphs: Constructions

Start with a small expander, and apply operations to make it bigger while preserving expansion

Squaring

- G^2 contains edge (u,w) if G contains edges (u,v) and (v,w) for some node v
- $A' = A^2$ – I/d I
- $\lambda' = \lambda^2$ – I/d
- $d' \leq d^2 - d$

Size	≡
Degree	↑
Expansion	↑

Expander Graphs: Constructions

Start with a small expander, and apply operations to make it bigger while preserving expansion

Tensor Product (Kronecker product)

- G = AxB nodes are (a,b) $\forall$ a$\in$A and b $\in$B
- edge between (a,b) and (a',b') if A contains (a,a') and B contains (b,b')
- $n' = n_1 n_2$
- $\lambda' = \max(\lambda_1, \lambda_2)$
- $d' = d_1 d_2$

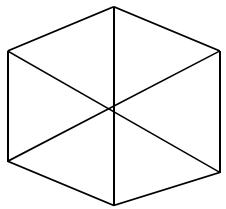
Size	↑
Degree	↑
Expansion	↓

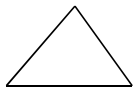
Expander Graphs: Constructions

Start with a small expander, and apply operations to make it bigger while preserving expansion

Zig-Zag product

- "Multiply" a big graph with a small graph





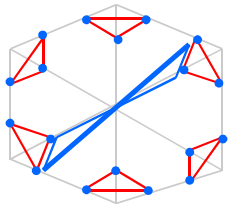
$n_2 = d_1$
$d_2 = \sqrt{d_1}$

Expander Graphs: Constructions

Start with a small expander, and apply operations to make it bigger while preserving expansion

Zig-Zag product

– "Multiply" a big graph with a small graph



Size	↑
Degree	↓
Expansion	↓ (slightly)

Combination: square and zig-zag

For a graph with size n, degree d, and eigenvalue λ, define G = (n, d, λ). We would like to increase n while holding d and λ the same.

Squaring and zig-zag have the following effects:

$(n, d, \lambda)^2 = (n, d^2, \lambda^2)$ $\equiv \uparrow\uparrow$

(n_1, d_1, λ_1) zz $(d_1, d_2, \lambda_2) = (n_1 d_1, d_2^2, \lambda_1 + \lambda_2 + \lambda_2^2)$ $\uparrow\downarrow\downarrow$

Now given a graph $H = (d^4, d, 1/5)$ and $G_1 = (d^4, d^2, 2/5)$

– $G_i = G_{i-1}^2$ zz H (square, zig-zag)

Giving: $G_i = (n_i, d^2, 2/5)$ where $n_i = d^{4i}$ (as desired)