Next two lectures (Today & Next week)

Current directions in coding theory

- Optimizing codes wrt Shannon model.
- Optimizing codes wrt Hamming model.

Issues

- Hamming model: Issues clear
  - Can decode from p fraction error with positive rate, for every $p < 1 - \frac{1}{q}$.
  - Can reach ZP bound?
- Shannon model: Issues less clear.
  - Can already decode close to capacity with linear time algorithms.
  - What else can we ask for?

Issues in practice

- Concerns
  - Concrete complexity: Not good enough to say for large enough n, but need for small n.
  - Algorithmic complexity: Classical constraint - efficiency not the only measure of "goodness". Other issues not explicated. Best description - somebody gives you a code + decoding algorithm (which has been chosen for reasons unspecified) - analyze! Fortunately, universality of powerful microprocessors makes this a lesser prevalent direction. We're getting back to efficiency as major criterion.

- Non-concerns
  - Exponential dependence between error and block length.
  - Linear running time.
  - Formal theorems + proofs (note simulation studies are meaningful in this world).

Prominent directions in the past decade

- "Turbo" codes (Berrou, Glavieux, Thitimajshima '93).
  - Random family of codes with simple enoding algorithm.
  - Decoding algorithm of "moderate descriptive complexity": Iterations of a basic routing.
  - Empirically strong observed behavior at small block lengths: Decoding algorithm seems to find codeword after "small" number of iterations with "high" probability.
  - Analysis? Not much success. Known to have poor minimum distance. Doesn't work for high error-rates. Algorithms not known to converge in infinite time. Even exponential time algorithms not known to correct "close" to capacity.

- "Tornado" codes (Luby, Mitzenmacher, Shokrollahi, Spielman + Richardson, Urbanke).
  - Random family of codes with simple decoding algorithm.
  - Decoding algorithm provably linear time correcting errors with "exponentially" small probability of decoding error.
  - Hidden constants small enough to beat out Turbo codes for many choices of parameters, but not all.

Linear time algs. vs. Linear time algs.

- Any "asymptotic" way to understand good vs. bad linear time algorithms?
- Definition in works of Luby et al.(Alon-Luby, LMSS*):
  - Suppose transmitting at rate ϵ away from capacity of channel.
  - How do size of code/running times depend on ϵ?
  - Algorithms so far $2^{1/\epsilon}$.
  - What is possible? $\ln 1/\epsilon$!
  - Tornado codes potential have such run times.

Today

Tornado codes

- Codes or Encoding (modulo details)
- Decoding
- Handwaving about Analysis (+ details of Codes)

Outline of code construction

- Basic idea similar to Spielman.
  - LDG code specifies how to get first round of check bits from message.
  - Reinforce check bits.
- Differences
  - Can't afford Spielman-style reinforcing (too expensive in terms of rate).
  - Don't need it either - only need to handle random errors.
- Idea - Reinforce recursively (we did hint this would be ok for random errors).

Further (but not all) details

- Suppose want a code of rate $R \approx 1 - \tau$.
- Start with graph G_1 with k left vertices and τk right vertices.
- G_2 has τk left and $\tau^2 k$ right vertices.
- Etc. Till G_i has ϵk vertices. Do anything now.
- Gets rate $1 - \tau - \epsilon'$.
- G_i's have constant degree (on avg.). More details later.
- Encode left to right.

Decoding algorithm

- Decode right to left.
- When decoding some layer: Assume all check bits right (get this for free anyway).
- Decoding proceeds in iterations: Each edge has a guess on the message bit. Guesses get updated in the iterations. Initially - guess = recieved bit. Over time, first check bit looks at current guesses on incident edges and sees if it is satisfied. If so, send "don't flip signal" to all edges; else send flip signal. Next edge looks at signals neighboring check bits. If all agree do what they ask you to do. If not, retract into womb (set value to what was recieved).

- Madhu's disclaimer: Don't know analysis, so can't shed light on specifics. Presumably many choices for algorithm. Pick the one that is feasible to analyze. One important criterion: Messages being sent always depend on "new random variables".

Analysis?

- Retraction: Won't analyze this at all!
- Will hint at analysis of much simpler problem.
- Binary erasure channel. Bits erased w.p. p or preserved with probablity $1-p$.
- Well-known fact: Rate $= 1-p$.
- In other words, k bits encoded as $k/(1-p)+\epsilon k$ bits. Receive $k+\epsilon k$ bits of encoding. Must be able to recover message for most subsets of such size.

Encoding + Decoding

- Encoding simplifies to graphs with k left vertices and $pk+\epsilon k$ right vertices.
- Decoding simplifies greatly: Right vertex has single unknown neighbor, implies neighbor's value known.
- Graphical evolution model: At time t, Have residual graph $G^{(t)}i$: If $\exists$ vertices of degree one on right erase edge and both enpoints. Repeat. Stop when no such edge exists.
- Strictly speaking: decoding successful if all $G^{(t)}$ is empty.
- But we'll be happy if it has ϵk vertices. (Why? Can slap on an expander to take

care of these. Rate not optimal, but doesn't matter.)

- .... So how does all this work? Depends on G.

Choice of bipartite graph

- Don't know exactly what suffices (in terms of a polynomial time constructible/verifiable property).
- Can show that when one is picked at random from a "special" distribution, it works.
- Speciality? Not regular graph!!!
- Why?

Intuition for irregularity

- Say γ_i is fraction of right vertices of degree i initially (after erasures).
- Need $\gamma_1 > 0$ so we can make progress, but can't afford $\gamma_0 > \epsilon$ (lose too much).
- Initial distribution: Bernoulli with parameter p if right degrees regular. Convex combination of Bernoulli's if not. (Degrees we work with aren't regular anyway!)
- Can erase many of the γ_1 fraction edges.
- Now what is the new distribution? Depends on how many edges we removed! Would

like to remove lots of edges so that we can create new degree 1 vertices. But not so many as to create many degree 0 vertices.

- Moral of the story: Really intricate issue.

To cut to the chase

- Graph degrees chosen as follows:
  - Left degree distribution: Fraction of vertices with left degree i proportional to $\frac{1}{i^2}$. Truncated to degree D.
  - Right degree distribution: Fraction of vertices of right degree i proportional to $\alpha^{i-1}/i!$ for some α fixed as function of D and p.

Analysis overview

- Let $\lambda_{i,\ell}$ be fraction of edges of left degree that gain self-realization i after ℓ rounds. (Similarly ρ for right.)
- Right recurrences for $\lambda_{i,\ell}, \rho_{i,\ell}$ as function of same with subscripts $\ell - 1$.
- (Draw AND/OR tree).
- Some generating function magic says if we let $\lambda(x) = \sum_i \lambda_i x^{i-1}$ and similarly $\rho(x)$, then we make progress if $\rho(1 - p.\lambda(x)) > 1 - x$.
- Part of analysis. Rest of analysis, converts this back of envelope calculation into rigorous analysis.

References

- IEEE Transactions on Information Theory, Feb 2001 contains:
  - LMSS 1: Erasure codes.
  - LMSS 2: Error-correcting codes.
  - Richardson-Urbanke: Stronger analysis.
- Also see: LMS - SODA '98.