A Crash Course on Coding Theory

Madhu Sudan
MIT

Topic: Applications in CS

We'll cover an assortment of applications of coding theory in computer science.

Disclaimer: Most connections to coding theory brought about in hindsight.

Computation in the presence of noise

- Influence of coding theory.
  - [von Neumann '56]
  - [Shannon & Moore '56]
  - [Elias '58]
  - [Dobrushin & Ortyukov '77]
  - [Rivest et al. '80]
  - [Pippenger '85]
  - [Spielman '97]
- Won't describe in detail.
  - Models tricky.
  - This is what codes were meant to do.
  - We will focus on less obvious connections.

Secret sharing [Shamir]

Defn: (n, k, N) Secret Sharing Scheme: "Distribute" secret $s \in [N]$ among n parties, so that no subset of size $k-1$ has any information about secret, while every subset of size k can compute secret.

Formally, SSS given by sets $\Omega, \mathcal{M}$ and

$$f : [n] \times [N] \times \Omega \to \mathcal{M}:$$

Distribution Given secret $s \in N$, pick $\omega \in \Omega$ at random and let i-th share be $f(i, s, \omega)$.

Recovery Given $\{(i, s_i) | i \in S\}$ for $|S| \geq k$, $|\{s \mid \exists \omega, \ \forall i \in S, \ f(i, s, \omega) = s_i\}| \leq 1$.

Secrecy Given $\{(i, s_i) | i \in S\}$ for $|S| < k$ $\forall s, \quad \Pr_\omega[\forall i \in S, \ f(i, s, \omega) = s_i] = \frac{1}{N}$.

Construction

Let $C = [n+1, k, n-k+2]_N$ MDS code.

Let $\Omega = [N]^{k-1}$ and $\mathcal{M} = [N]$.
f is computed as follows:
Given $s \in [N]$, and $\omega = \langle s_1, \ldots, s_{k-1} \rangle$
let $c \in C$ s.t. $(c)_i = s_i$ and $(c)_{n+1} = s$.
Then $f(i, s, \omega) = (c)_i$.

Properties

Distance $\Rightarrow$ Recovery.
Dimension $\Rightarrow$ Secrecy.

[Shamir]'s scheme:
Used polynomials = RS codes.

Probabilistic Communication Complexity

Defn: Let $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$. The communication complexity of f is the minimum number of bits that two players X and Y have to exchange to determine $f(x, y)$, where initially X knows x and Y knows y.

Parties may be forced to be deterministic or allowed to toss random coins.

An important result in the early study of communication complexity, separated probabilistic communication complexity from deterministic communication complexity, by considering the Identity function.

Prob. Comm. Complexity

Identity$(x, y) = 1$ if $x = y$ and 0 o.w.,

[Yao]: Identity has deterministic comm. complexity at least n.

[Rabin+Yao]: Identity has prob. comm. complexity $O(\log n)$.

Proof:

- Parties agree on a code $C = [2n, n, .01n]_2$.
- X picks $i \in [2n]$ at random
- X sends $(i, C(x)_i)$ to Y.
- Y sends back 1 if $C(x)_i = C(y)_i$ and 0 otherwise.
- (Repeat as desired.)

Original proof: Via Chinese Remaindering.

Pseudorandomness: l-wise independence

Limited independence:

- Concept used in reducing randomness used by randomized algorithms.
- Take algorithm using n independent random coins.
- Show algorithm works as well when given n dependent coins in which any subset of l are independent.
- Use an l-wise independent sample space. (Typically much less randomness.)

Defn: (Simplified): A set $S \subseteq [q]^n$ is l-wise independent, if for every sequence of l distinct indices $i_1, \ldots, i_j$ and $b \in [q]^l$,
$\Pr_{x \in S}[\forall j \in [l], x_{i_j} = b_j] = q^{-l}$.

Goal: Given n, q, l find smallest such S.

l-wise independence

Insight: linear independence $\Rightarrow$ independence.

Construction:

- Pick code $C = [n, k, l+1]_q$.
- Set $S = C^\perp$. (Duality!)

Correctness: Exercise!
Quality: Rate(C) large $\Rightarrow$ S small.

Well-known examples:

Pairwise independence:
Usual construction: Hadamard codes.
From above: Hamming$^\perp$ = Hadamard!

k-wise independence:
Usual construction: Polynomial evaluation
From above: RS$^\perp$ = RS codes!

Aside: Explicit constructions

Some common examples of combinatorial structures for which we seek explicit constructions:

- Error-correcting codes
- Designs (Constant-weight binary codes)
- Expanders
- Pseudo-random sequences
- Low disrepancy sequences
- Dispersers
- Extractors

Many inter-relationships.

Examples

- (m, n, t)-Design: Family of subsets of $[m]$, each subset of size n, with every intersection being of size at most t.

  Designs = constant-weight binary ECCs.

- [Nisan-Wigderson] Pseudo-Randomness: Create pseudo-random strings from hard functions. One key ingredient: Design.

- Many paths from ECCs to pseudo-randomness! (ϵ-biased spaces: another well-known application of ECCs.)

Examples (contd).

- [Trevisan] Constructs "Extractors" - a special family of expanding graphs: Main ingredients: [NW] (i.e., designs) + error-correcting codes of large list-decoding radius!

- [*.*] Interactive Proofs, Program Checking and Probabilistically Checkable Proofs. Rich collection on results over the course of last 12 years. Reliance on coding theory hidden initially, but explicit now!

(Details omitted.)

Linearity Testing

Defn: $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is linear if for every $x, y \in \mathbb{F}_2^n$, we have $f(x) + f(y) = f(x + y)$.

Problem:
Given: oracle access to $f : \mathbb{F}_2^n \to \mathbb{F}_2$.
Goal: Is function linear? I.e.,

Completeness f linear $\Rightarrow$ accept w.p. 1.

Soundness If every linear function g is at least ϵ-far in Hamming distance from f, then reject with probability δ.

Linearity Testing (contd).

Linearity Test[Blum,Luby,Rubinfeld]
- Pick x, y at random
- Accept if $f(x) + f(y) = f(x + y)$.

Performance: How do ϵ and δ relate?
In particular, is δ lower bounded by some growing function of ϵ?

- Easy: $\epsilon > 0 \Leftrightarrow \delta > 0$.
- Non-trivial: $\delta > \frac{2}{9}\epsilon$. [BLR].
- Subsequently: $\delta > \epsilon$ [BCHKS].
- The realization: Testing $\equiv$ Duality of coding theory [Kiwi].

Linearity Testing a la [Kiwi]

Let C be code whose codewords are all linear functions (where truth tables of linear functions are viewed as binary vectors).

C is the $[2^n, 2^n, 2^{n-1}]_2$ Hadamard code.

Let $C + f$ be the linear code spanned by codewords of C and f.

Realization 1: $\epsilon =$ distance rate of $C + f$.

Linearity testing (contd).

- To understand $\delta =$ need to look at dual.
- Random strings in linearity test correspond to weight 3 codewords in $C^\perp$!
- Recall $(C + f)^\perp \subseteq C^\perp$. $1 - \delta =$ fraction of weight 3 codewords of $C^\perp$ that are also contained in $(C + f)^\perp$!
- Summarizing:
  - Have partial info on wt. dist. of primal.
  - Have partial info on wt. dist. of dual.
  - Use MacWilliams Identities!
- [Kiwi] analyzes many tests! Esp. linearity testing over arbit. fields.

Worst-case to Average-case

Defn: Language L belongs to avg-P if there exists a polynomial time algorithm that decides L on "most" instances of every length n, when inputs are drawn uniformly at random from $\{0,1\}^n$.

Question: Is $\text{avg-P} = \text{NP}$?

Answer: Certainly, YES if P = NP.
But what if P $\neq$ NP?

Questions of this nature are studied under the label "Average-case Complexity"

Relationship between average to worst case open at the P vs. NP level.

Coding theory answers the questions at the EXP vs. P/poly level.

Aside: Definitions

EXP = languages decidable in exponential time. (Hard).

P/poly = languages decidable with polynomial size circuits. (Easy).)

Worst-case to Average-case

Thm: $\text{EXP} \not\subset \text{P}/\text{poly} \Rightarrow \text{EXP} \not\subset \text{avg-P}/\text{poly}$.

Proof uses following code.

Fact: $\forall \epsilon > 0, k$, $\exists$ systematic code C mapping k bits to $n = \text{poly}(k/\epsilon)$ bits, with list-decoding algorithm that, given an implicit representation of a received word $r \in \{0,1\}^n$, outputs implicit representations of all codewords within a distance of $\frac{1}{2} + \epsilon$ from the received word, in $\text{poly}\log n$ time.

(Implicit representations = oracles)

Worst-case to Average-case

Proof:[of Theorem, assuming Fact]:

Given: $g : \{0,1\}^l \rightarrow \{0,1\}$ in $\text{EXP} - \text{P}/\text{poly}$.

- Let $C = [n, k = 2^l, \delta]_2$ code (from Fact.)
- View $g = 2^l$-bit string to be encoded.
- View $h = C(g)$ as truth-table of function.
- Then h is hard almost everywhere.

Analysis:

- Let f predict h with accuracy $\frac{1}{2} + \epsilon$.
- View f as implicit rep'n of rec'd vector.
- List decoder outputs implicit representations (circuits) computing nearby codewords.
- Some nearby codeword is $h = C(g)$; hence computes g efficiently.

Hard Boolean Functions

Issue:

Suppose $\exists$ hard computation problem.
Then, do there exist hard languages?

Distinction?

Problem $= f : \{0,1\}^n \to \{0,1\}^m$.
Language $= L : \{0,1\}^n \to \{0,1\}$.

Languages are easier to work with, but insight into hardness usually comes from general functions.

Well-known answer: Obviously yes.
If $f : \{0,1\}^n \to \{0,1\}^m$ is hard, then
so is $L : \{0,1\}^n \times [m] \to \{0,1\}$,
where $L(x,i) = (f(x))_i$.

Hard Boolean Functions (contd).

- Answer satisfactory in a worst-case setting.
- But weak in the average-case setting. E.g.,

  If the Discrete Log function is almost always hard to compute, then the resulting language may be easy to compute on $1-\frac{1}{n}$ fraction of the inputs!
- Can we do better?

  (Specific answers exists. E.g., it is known that most bits of the Discrete Log function are very hard. We look for generic answers.)

Hardcore Predicates[Goldreich+Levin]

Given: One-way perm. $\pi : \{0,1\}^k \to \{0,1\}^k$.
Want: Boolean function $b = b(x,i)$ s.t.
$b(x,i)$ hard to compute given $\pi(x), i$.

(Can't compute π w.p. greater than ϵ should imply can't predict b w.p. greater than $\frac{1}{2}+\delta$.)

Abstract GL [Impagliazzo]:
- Let $C = [n,k,\delta]_2$ code w. list decoder.
- $b(x,i) = C(x)_i$ is a hardcore predicate.

Analysis:
- Let $A(\pi(x), i)$ compute $b(x,i)$ w.p. $\frac{1}{2}+\epsilon$.
- List decode f where $f(i) = A(\pi(x), i)$.
- Gives $C(x_1), \ldots, C(x_k)$ s.t. some $x_i = x$.
- Check which one using $\pi(x)$.

Application (contd.)

[GL]:
- Use C = Hadamard code.
- Oracle rep'n saves time to write f.
- Give eff. list decoding algorithm for Hadamard code.

[Impagliazzo]:
- Use Thm 2.
- Extra randomness reduces from $O(k)$ to $O(\log k)$.

Conclusion

- Many interesting applications.
- But most connections to coding theory found after the fact.
- Does explicit knowledge help? Recent results (e.g. [Trevisan]) seem to say, YES!