Contents

1 Linear coupling: an ultimate unification of gradient and mirror descent

Linear Coupling: An Ultimate Unification of Gradient and Mirror Descent

I am talking about the paper by Allen-Zhu and Orecchia `http://arxiv.org/abs/1407.1537` on explaining Nestorov's accelerated gradient descent. The paper is very elegant. The original abstract is here.

First-order methods play a central role in large-scale convex optimization. Even though many variations exist, each suited to a particular problem form, almost all such methods fundamentally rely on two types of algorithmic steps and two corresponding types of analysis: gradient-descent steps, which yield primal progress, and mirror-descent steps, which yield dual progress. In this paper, we observe that the performances of these two types of step are complementary, so that faster algorithms can be designed by linearly coupling the two steps. In particular, we obtain a simple accelerated gradient method for the class of smooth convex optimization problems. The first such method was proposed by Nesterov back to 1983, but to the best of our knowledge, the proof of the fast convergence of accelerated gradient methods has not found a clear interpretation and is still regarded by many as crucially relying on "algebraic tricks". We apply our novel insights to construct a new accelerated gradient method as a natural linear coupling of gradient descent and mirror descent and to write its proof of convergence as a simple combination of the convergence analyses of the two underlying descent steps. We believe that the complementary view and the linear coupling technique in this paper will prove very useful in the design of first-order methods as it allows us to design fast algorithms in a conceptually easier way. For instance, our technique greatly facilitates the recent breakthroughs in solving packing and covering linear programs [AO14, AO15].

1.1 Basic convex optimization

We want to find $\min f(x)$ such that $x \in Q$, where f is convex. We won't worry about the constraint Q in this talk.

In the first-order method, we are given access to $\nabla f(x)$, but we can compute it at every point. The complexity measure is the number of queries.

Let $\|\cdot\|$ be a norm, and $\|\cdot\|_*$ be the dual norm. In most cases the norms will both be the Euclidean norm $\|\cdot\|$.

Definition 1.1: We say f is L-smooth with respect to $\|\cdot\|$ as follows if for all y, x,

$$f(y) \leq f(x) + \nabla f(x)^T (y - x) + \frac{L}{2} \|y - x\|^2 .$$

I.e., we can upper-bound f by a quadratic function (with squared-term coefficient $\frac{L}{2}$).

Definition 1.2: We say f is μ-strongly convex if for all y, x,

$$f(y) \geq \underbrace{f(x) + \nabla f(x)^T (y - x) + \frac{\mu}{2} \|y - x\|^2}_{\Phi_x(y)} .$$

Without the last term this inequality is always true for convex functions. This says we can lower-bound f by a quadratic function, rather than just a linear function tangent to it.

The gradient method is a primal method; the mirror is a dual method.

We review the gradient descent method. Start at some point, and use the update rule

$$x_{t+1} = x_t - \frac{1}{L}\nabla f(x_t).$$

where L is the smoothness.

Lemma 1.3: We have

$$f(x_{t+1}) \le f(x_t) - \frac{1}{2L}\left\|\nabla f(x_t)\right\|_*^2$$

If we have a sharp gradient, we must make some progress. For the L^2 norm this is easy to see. Using the definition of the smoothness,

$$\begin{aligned} f(x_{t+1}) &\le f(x_t) + \nabla f(x_t)^T(x_{t+1} - x_t) + \frac{L}{2}\left\|x_{t+1} - x_t\right\|_2^2 \\ &= f(x_t) + \nabla f(x_t)^T\left(-\frac{1}{L}\nabla f(x_t)\right) \\ &= f(x) - \frac{1}{2L}\left\|\nabla f(x_t)\right\|_2^2 \qquad\qquad x_{t+1} - x_t = -\frac{1}{L}\nabla f(x_t). \end{aligned}$$

(Note it's also okay for $x_{t+1} - x_t = \delta_t$ for some δ_t correlated with the gradient (the angle is nontrivial). Note $\nabla {x_{t+1} - x_t}^2$ is quadratic, but $\nabla f(x_t)$ is linear, so you can tune it to be larger. In other cases you may only have approximate/stochastic queries.)

How do you get the dual norm if you work with non-Euclidean norms? If you use a general norm, then the update rule is

$$x_{t+1} = \operatorname{argmin}_y \Phi_{x_t}(g).$$

It's not clear whether you can do it in polynomial time; the norm should be nice. Then the calculation above becomes

$$\begin{aligned} f(x_{t+1}) &\le \Phi_{x_t}(x_{t+1}) \\ &\le \Phi_{x_t}(y) \\ &= f(x_t) + \nabla f(x_t)^T(y - x_t) + \frac{L}{2}\left\|y - x_t\right\|^2 \\ &\quad \text{choose } y - x_t = -\theta v \\ &\quad \text{such that } \nabla f(x_t)^T v = \left\|\nabla f(x_t)\right\|_* \left\|v\right\|, \left\|v\right\| = 1 \\ &\quad \nabla f(x_t)^T(y - x_t) = -\theta\left\|\nabla f(x_t)\right\|_*^2 \\ &\quad \frac{1}{2}\left\|y - x_t\right\|^2 = \frac{1}{L}2\theta^2\left\|v\right\|^2 = \frac{L}{2}\theta^2 \\ &= f(x_t) - \theta\left\|\nabla f(x_t)\right\|_* + \frac{L}{2}\theta^2 \\ &\quad \theta = \frac{1}{L}\left\|\nabla f(x_t)\right\|_* \\ &\quad \left\|\nabla f(x_t)\right\|_* = \max\left\{\langle \nabla f(x_t), v\rangle : \left\|v\right\| = 1\right\} \end{aligned}$$

We won't work with dual norms for the rest of the talk.

1.2 Mirror descent

Definition 1.4: Define a **distance generating function** (DGF) with respect to $\|\cdot\|$ as a 1-strongly function $w(x)$ such that

$$w(y) \geq w(x) + \nabla w(x)^T(y-x) + \frac{1}{2}\|y-x\|^2$$

The **Bregman distance** is

$$V_x(y) = w(y) - w(x) - \nabla w(x)^T(y-x)$$
$$V_x(y) = \frac{1}{2}\|y-x\|^2 .$$

(Note it's asymmetric. Think of x as a reference point; we measure the distance to x.)

Example 1.5: Take $w(x) = \frac{1}{2}\|x\|_2^2$. The Bregman distance is

$$V_x(y) = \frac{1}{2}\|y-x\|_2^2 .$$

It generates the norm square. $V_x(y)$ is distance between the linear approximation and the upper quadratic approximation.

Example 1.6: Consider a simplex

$$S = \{x : \|x\|_1 = 1, x \geq 0\} .$$

This is strongly convex with respect to the L^1 norm. Let $w(x) = \sum x_i \ln x_i$. Then

$$V_x(y) = \sum y_i \ln \frac{y_i}{x_i} = D_{KL}(y||x) \geq \frac{1}{2}\|y-x\|_{TV}^2 .$$

by Pinsker's inequality.

What mirror descent does is the following. Set

$$x_{t+1} = \text{argmin}_y\{V_{x_t}(y) + \alpha\nabla f(x_t)^T(y-x_t)\}.$$

We don't assume smoothness. Suppose we have a convex function. We don't want to minimize $f(x_t) + \nabla f(x_t)^T(y-x_t)$ in y because that would be $-\infty$. We want to find some point not too far from x_t. We believe the linear funciton is a good approximation, but it is only a good approximation when it is close to x. Hence we add in a regularizer $V_{x_t}(y)$.

Example 1.7: Let $w(x) = \frac{1}{2}\|x\|_2^2$.

$$x_{t+1} = \text{argmin}\{\frac{1}{2}\|y-x_t\|^2 + \alpha\nabla f(x_t)^T(y-x_t)\} = x_t - \alpha\nabla f(x_t).$$

We have a different step size α which in some cases can be chosen larger.

w is a convex function and $w(x)$ is a constant so $V_x(y)$ is convex. The minimizer is where the gradient is 0.

$$\nabla V_{x_t}(y) + \alpha \nabla f(x_t) = 0$$

Plugging in x_{t+1} we should get 0:

$$\begin{aligned} \nabla V_{x_t}(x_{t+1}) + \alpha \nabla f(x_t) &= 0 \\ \nabla w(x_{t+1}) &= \nabla w(x_t) - \alpha \nabla f(x_t). \end{aligned}$$

Hence we get another way to state the update rule for mirror descent,

$$\nabla w(x_{t+1}) = \nabla w(x_t) - \alpha \nabla f(x_t).$$

We have a primal space X, and we have a map ∇w which sends $x_t \in X$ to $\nabla w(x_t)$. We take a gradient step in the dual space $-\alpha \nabla f(x_t)$ to get $\nabla w(x_{t+1})$ and map it back using the inverse map ∇w^{-1} to get x_{t+1}. This is where the name "mirror" descent comes from.

For strongly convex we roughly have a lower bound $|\nabla w(x) - \nabla w(y)| \geq \mu \|x - y\|$ so that the gradient is invertible.

Lemma 1.8 (Main lemma for mirror descent)**:**

$$\begin{aligned} \alpha[f(x_k) - f(x^*)] &\leq \alpha[\nabla f(x_k)^T (x_k - x^*)] \\ &\leq \frac{\alpha^2}{2} \|\nabla f(x_k)\|_*^2 + V_{x_k}(x^*) - V_{x_{k+1}}(x^*) \end{aligned}$$

Summing we get a telescoping sum. Summing over k,

$$\begin{aligned} \alpha \sum_{k=1}^{T-1} (f(x_k) - f(x^*)) &\leq \sum_{k=0}^{T-1} \frac{\alpha^2}{2} \|\nabla f(x_k)\|_*^2 + V_{x_0}(x^*) - V_{x^T}(x^*) \\ &\leq \sum_{k=0}^{T} \frac{\alpha^2}{2} \|\nabla f(x_k)\|_*^2 + V_{x_0}(x^*) \end{aligned}$$

2 Improved noisy population recovery, and reverse Bonami-Beckner inequality for sparse functions

Authors: Shachar Lovett, Jiapeng Zhang

Abstract: The noisy population recovery problem is a basic statistical inference problem. Given an unknown distribution in $\{0,1\}^n$ with support of size k, and given access only to noisy samples from it, where each bit is flipped independently with probability 1/2, estimate the original probability up to an additive error of ε. We give an algorithm which solves this problem in time polynomial in $(k^{\log\log k}, n, 1/\varepsilon)$. This improves on the previous algorithm of Wigderson and Yehudayoff [FOCS 2012] which solves the problem in time polynomial in $(k^{\log k}, n, 1/\varepsilon)$. Our main technical contribution, which facilitates the algorithm, is a new reverse Bonami-Beckner inequality for the L1 norm of sparse functions.

The setup is the following.

Problem 2.1: There an unknown distribution π over $\{0,1\}^n$ with $|\operatorname{Supp}(\pi)| = k$. (Think of this as a mixture of k distributions.)

The input are noisy samples from π,

1. $X \sim \pi$
2. flip each x_i with probability $\frac{1}{2} - \mu$.

Output the distribution $\widehat{\pi}$ close in total variation distance:

$$TV(\widehat{\pi}, \pi) \leq \varepsilon.$$

Ideally an algorithm has runtime $\operatorname{poly}(k, n, \frac{1}{\varepsilon}, \frac{1}{\mu})$.

1. This problem was first considered by Kearns et al., who gave an algorithm in time $\operatorname{poly}(n, \frac{1}{\varepsilon}) \exp(k)$.
2. YW do this in time in $\operatorname{poly}(n, \frac{1}{\varepsilon}) k^{\ln k}$.
3. This paper does this in time $\operatorname{poly}(n, \frac{1}{\varepsilon}) k^{\ln \ln k + \ln\left(\frac{1}{\nu}\right)}$.

A related probem is lossy population recovery, where we are given

1. $X \sim \pi$.
2. Replace each x_i with '?' with probability $1 - \nu$.

This is easier; Moitra and Saks show a polynomial time algorithm for constant ν.

First we make some simple claims.

First claim: If you want to solve the noisy population recovery problem, we can assume we know the support of the distribution. Suppose we have an algorithm that works given the support. Now run it on increasing prefixes. First run it on the prefixes 0, 1. What are the probability of strings with $x_1 = 0, 1$? Now run the algorithm on $x_1 x_2$, etc.

Suppose $\operatorname{Supp}(\pi) = \{x_1, \ldots, x_k\}$, and we want to find the probabilities $p_1, \ldots, p_k$. We use the maximum likelihood estimator. Draw m samples $S_m = \{y_1, \ldots, y_m\}$. Calculate

$$\begin{aligned} p(y_1 | p_1, \ldots, p_k) &= \sum_i p_i \left(\frac{1}{2} - \mu\right)^{\Delta(x_i, y_i)} \left(\frac{1}{2} + \mu\right)^{n - \Delta(x_i, y_i)} \\ &= \sum_i p_i e^{-\Delta(x_i, y_i) C_\mu}. \end{aligned}$$

We want to find

$$\max_{\sum_i p_i = 1} \sum_{y_i \in S_m} \ln \left(\sum_i p_i e^{-\Delta(x_i, y_i) C_\mu} \right).$$

How many samples do we need to approximate this? We want to distinguish distributions which are ε apart: $TV(\pi_1, \widetilde{\pi}_2) > \varepsilon$. However we don't have perfect π_1, π_2? We want to characterize $TV(\widetilde{\pi_1}, \widetilde{\pi_2}) > a$. YW built a graph structure rather than using the MLE.

Given $\pi_1 : \{0,1\}^n \to \mathbb{R}$, the Fourier expansion is

$$\pi_1(x) = \sum_{S \subseteq [n]} \widehat{\pi_1}(G)\chi_S(x).$$

Now

$$\begin{aligned} \widetilde{\pi_1}(x) &= \mathbb{E}_e[\pi_1(x+e)] && e_i \sim B(\frac{1}{2} - \mu) \\ &= T_{2\mu}\pi_1(x) = \sum_{S \subseteq [n]} (2\mu)^{|S|}\widehat{\pi_1}(S)\chi_S(x). \end{aligned}$$

Given $f : \{0,1\}^n \to \mathbb{R}$, $f = \frac{1}{2}(\pi_1 - \pi_2)$, $\|f\|_1 \geq \varepsilon$, $\|f\|_1 = \mathbb{E}_x[|f(x)|]$, $\|T_{2\mu}f\|_1 \geq a$,

Theorem 2.2: Given $f : \{0,1\}^k \to \mathbb{R}$, $|\operatorname{Supp}(f)| = k$,

$$\|T_{2\mu}f\|_1 \geq k^{-O(\ln\ln n + \ln(\frac{1}{\mu}))} \|f\|_1 .$$

This is a reverse hypercontractivity inequality because a hypercontractive inequality goes the other way.

The idea is as follows. Given $\operatorname{Supp}(x) = \{x_1, \ldots, x_n\}$, $\widetilde{f}(x) = \sum_i v_i e^{C_\mu d(x,x_i)} \leq \frac{1}{k}\sum_i v_i$ if not close to any. If it's close to one it will get a reasonably high value. This means that looking at balls of radius $\ln k$ around each value, the balls will contain a reasonble fraction of the L^1 mass, and one will contain at least $O\left(\frac{1}{k}\right)$ of them.

Our goal is to bound $\|T_{2\mu}f\|_1$, $T_{2\mu}(f) = \sum_{S\subseteq[n]} \mu^{|S|}\widehat{f}(S)\chi_S(x)$. We use the trivial bound (for any S)

$$\begin{aligned} \|T_{2\mu}(f)\|_1 &\geq |\widehat{T_{2\mu}f}(S)| \\ &= (2\mu)^{|S|}|\widehat{f}(S)|. \end{aligned}$$

To see this note $\widehat{f}(S) = |\mathbb{E}\chi_S(x)f(x)| \leq \|f\|_1$. Our goal is just to bound one Fourier coefficient.

Lemma 2.3: Define a function $g(x) = f(x)T_{2\mu}h(x)$, where $h : \{0,1\}^n \to \mathbb{R}$ is a given function with $\|h\|_\infty \leq 1$.

Then

$$\|T_{2\mu}f\|_1 \geq \mu^{|S|}|\widehat{g}(S)|.$$

Proof. We find

$$\begin{aligned}
\widehat{g}(S) &= \mathbb{E}_x[\chi_S(x)g(x)] \\
&= \mathbb{E}_x[\chi_S(x)f(x)T_{2\mu}h(x)] \\
&= \mathbb{E}_x[h(x)T_{2\mu}\chi_S(x)f(x)] \\
&\leq \mathbb{E}_x[|T_{2\mu}\chi_S(x)f(x)|] \\
&= \|T_{2\mu}\chi_S(x)f(x)\|_1 \\
&\leq \frac{1}{(2\mu)^{|S|}} \|T_{2\mu}f(x)\|_1 \\
T_{2\mu}\chi_S(x)f(x) &= \mathbb{E}_e[\chi_S(x+e)f(x+e)], \qquad\qquad e_i \sim B(\frac{1}{2}-\mu) \\
&= \mathbb{E}_{e_S}[\chi_S(x+e)\mathbb{E}_{e_{\overline{S}}}[f(x+e_S+e_{\overline{S}})] \\
&\leq \mathbb{E}_{e_S}\left[|\mathbb{E}_{\overline{S}}f(x+e_S+e_{\overline{S}})|\right] \\
&\leq \|\mathbb{E}_{e_S}f(x+e_S+e_{\overline{S}})\|_1 \leq \frac{\|T_{2\mu}f\|_1}{(2\mu)^{|S|}} \\
f(x+e_S+e_{\overline{S}}) &= \sum_{T\subseteq[n]} (2\mu)^{|S|}\frac{(2\mu)^{|S\setminus T|}}{(2\mu)^{|S|}}\widehat{f}(T)\chi_S(T) \\
&= \frac{T_{2\mu}f}{(2\mu)^{|S|}}.
\end{aligned}$$

The same formula is true even if the noise probability is different for different coordinates. □

Choose $r \approx \frac{\ln k}{\mu^2}$, $h(x) = 1_E(x)$,

$$E = \{y \in \{0,1\}^n : d(y,x_i) < d(y,x_2) \text{ for all } x_i \text{ such that } d(x_i,x_1) \geq r\}.$$

For $g(x) = f(x)T_{2\mu}h(x)$, $|g(x_1)| \geq \frac{1}{2k}$, $g(x_i) \leq \|f\|_1 e^{\mu^2 d(x_1,x_i)}$, and

$$\begin{aligned}
g(x_1) &= f(x_1)\mathbb{E}_e h(x_1+e) \\
&= f(x_1)\mathbb{P}_e[x_1+e \in E] \\
\mathbb{P}(x_1+e \notin E) &\leq e^{-\mu^2 r} \\
g(x_1) &\geq \frac{f(x_1)}{2}.
\end{aligned}$$

We prove the existence of a large low-degree coefficient. Take $g \approx g^*$, $|\operatorname{Supp}(g^*)| \leq k$, $\operatorname{Supp}(g^*) \subseteq B(r,x)$.

Lemma 2.4: For $g^* : \{0,1\}^n \to \mathbb{R}$, $|\operatorname{Supp}(g^*)| \leq k$, $\operatorname{Supp}(g^*) \subseteq B(n,r)$, (1)

$$\|T_{2\mu}g^*\|_1 \geq k^{-\ln(4x)} \|g^*\|_1.$$

For $\|T_{2\mu}g^*\|_1 \geq \mu^{|S|}|g^*(S)|$,

Claim 2.5: There exists a low-degree polynomial $p(x) = \sum_s p_s$, $\deg(P) \leq \ln k$,

1. $P(x) = g^*(x)$ for all $x \in \mathrm{Supp}(g^*)$
2. $\|p\|_1 = \sum_S |p_S| \le kr^{\ln k} \|g^*\|_1$.

(For $x_1 = 1$, $f(x) = \left(\frac{1-\mu}{2}\right)^r \left(\frac{1+\mu}{2}\right)^{n-r}$, $\left\|\widetilde{f}\right\|_1 = \sum_i \left(\frac{1-\mu}{2}\right)^r \left(\frac{1+\mu}{1+\mu}\right)^r \binom{n}{r}$.)
How claim implies 1:

$$\langle g^*, p\rangle = \sum_x g^{*2}(x) = \sum_x g^{*2} \ge \frac{\|g^*\|_1^2}{k}$$
$$\langle g^*, p\rangle = \sum_S \widehat{g^*}(S) p_S \le \max_S |\widehat{g}(S)| \, \|f\|_1 .$$

Proof. $C_1(K), C_2(K, w)$. $C_1(K)$ minimal value such that for all $n \ge 1$, $\forall x \subseteq B(n, x)$, $\forall f : X \to \mathbb{R}$ there exists $P(x)$ such that

1. $\deg(p) \le \ln k$
2. $P(x) = f(x)$, $\forall x \in X$
3. $\|P\|_1 \le c_1(k) \|f\|_1$.

$C_1(k, w)$ is defined similarly but for $w(x) \le w$ Claim:

$$G(k, w) \le \max_{1 \le a - \frac{k}{2}} (G(k, w - a) + G(a))$$

Proof: For $f, w(x) = w$,

$$f(x) = f_1(x_1, \ldots, x_{n-1}) + x_n f_2(x_1, \ldots, x_{n-1})$$
$$f(x) = f_1(x_1, \ldots, x_{n-1}) + x_n f_2(x_1, \ldots, x_{n-1}) 1(x \notin E).$$

Partition $\mathrm{Supp}(x)$ into $x_n = 0, 1$. At most $k/2$ with $x_n = 1$. E is intersection with $x_1 = 0$. $E \subseteq \mathrm{Supp}(f_2)$.

The latter is a function on $n - 1$ variables with support $\le \frac{k}{2}$. If the support size is a,

$$\begin{aligned} G(k, w) &\le \max_{i - a \le \frac{k}{2}} G(k, w - a) + G(a).G(k) && = G(k, kx) \\ &\le \max_{1 \le a_1, \ldots, a_t \le \frac{k}{2}, \sum a_i \le kx} \sum_i G(a_i). \end{aligned}$$

Induction hypothesis $G(a_i) \le a_i r^{\ln a_i}$, $G(k) \le kr^{\ln k}$. □

Question: Given $f : \{0, 1\}^n \to \mathbb{R}$, $|\mathrm{Supp}(f)| = k$, does that there exists S, $|S| \le \ln k$,

$$\widehat{f}(S) \ge \frac{1}{\mathrm{poly}(k)}.$$

Then we have a polynomial time algorithm.

3 Hardness of signaling for Bayesian games by Shaddin Dughmi

Say we're driving to New York or Philadephia. There are many paths. It's the time via GW bridge, Lincoln tunnel, etc. But people say some route is fastest, then it becomes slower because more people go to that.

Start with the Bayesian setting $\{G_\theta\}_{\theta\in\Theta}$. Without any knowledge, players play $\mathbb{E}_{\theta\in\Theta,\theta\sim\lambda}G_\theta$ and converge to Nash equilibrium. (Nature decides θ, but is unkown to both players.)

Signalling means a distribution $\varphi(\theta,\sigma)$, $\sigma\in\Sigma$. For example, Σ could be a partition of Θ. Here we consider symmetric signalling; both players get the same information. A nondeterministic signal could do better.

The goal is to have the induced distribution on θ be given by σ, $\sum_\sigma \alpha_\sigma\varphi(\theta|\sigma)=\lambda$, where α_σ is the probability of picking σ as signal. Decomposing the prior distribution into a convex combination.

In this setting the goal is to maximize player 1's expected payoff. It could be welfare, or any monotone function on the pair. There is a quasi-polytime algorithm that works for any game, with ε-Nash equilibrium (subject to ...). (Markokis-Mehta: you can get ε-Nash.) This is based on the LMM algorithm. The idea is to create an ε-net. This is for a broader class of objective functions and games.

In our setup, $\{G_\theta\}_{\theta\in\Theta}$ is a zero-sum game. (All equilibria have the same value by the minimax theorem.)

The whole point is that signalling is as hard as finding a hidden clique of size $k=\omega((\ln n)^2)$. This matches the upper bound given by the LMM algorithm.

Define a new problem called Hidden Clique Cover ($G(n,p,k,r)$) problem, which is the following. We state it not as a decision problem but as a search problem. Let $G\sim G(n,p)$ a Erdős-Renyi random graph. Plant a clique of size k r times.

Our goal is to recover a constant fraction of these cliques.

If you can solve the search version you can make progress on the regular version. Plant the correct one at one point and plant $r-1$ garbage cliques. Every time you recover some constant fraction; hope that the algorithm will output the correct one at some point. Thus the problem is HC-hard. (HC=hidden clique.)

Given an instance $G\sim G(n,p,k,r)$, $T\subseteq V$, our goal is to output a list of cliques S such that $|S\cap T|\geq\varepsilon|T\cup S|$. Given an instance, and a set of vertices, then we can recover all cliques which have a large intersection with T in polynomial time.

It's a randomized algorithm: pick some samples. Suppose you know what $|S\cap T|$ is. Sample $\ln n$ samples from T; a constant fraction of them is uniformly distributed from $S\cap T$. Look at the vertices that have it as a common neighbor (?). Now you have a larger subset; cut out the vertices which don't have at least k neighbors. This returns the list of cliques with high probability.

If you have a subset guaranteed to have constant fraction intersection with clique S_i then you can fully recover S.

The game is as follows. Given $G\sim G(n,p,k,r)$, a state is $\theta\in V$. There are 2 players, attacker and defender. The attacker wants to destroy edges (ex. network from a server). He can hit the vertex or one of its neighbors. Hitting a vertex severs all incident edges.

The attacker picks $v \in V$. If $(v, \theta) \in E$, then the attacker gets a point.

The defender picks $D \subseteq V$, defends edges incident on D. Actually this isn't quite true: the defender picks a vertex with weight d; then mixed strategies include all subsets of size d.

If the attacker picks an edge but the defender defended the edge, then the attacker gets $-\rho$, where $\rho d = \Theta(k)$. This rules out FPTAS but not PTAS.

The attacker's payoff is

$$A^\theta(a, D) = |\{(\theta, a)\} \cap E| - \rho |D \cap \{\theta, a\}|.$$

The subsets are mixed strategies over points.

Observations:

- There exists a signalling scheme that gives ≥ 0.8 payoff for the attacker.

  The hidden clique cover gives a partition of the graph, defined as follows. Letting S_i be the ith planted clique, define

  $$\widehat{S_i} = S_i \setminus \bigcup_{j=1}^{i-1} S_i,$$
  $$\widehat{S_0} = V \setminus \bigcup_{i=1}^{r} S_i.$$

  θ falls into one of these partitions. Send the corresponding set in the partition to both players.

  Note that $\widehat{S_0}$ has to be small if $kr = \Theta(n)$. Most of the vertices are in one of the good partitions. It has to be a clique. There the attacker wins if he picks any vertex—he only cares about whether the defender defends successfully; it doesn't happen that much: if $d = |D|$ the probability of successful defense is

  $$\mathbb{P}(a \in D) + \mathbb{P}(\theta \in D) \leq \frac{2d}{|S_i|}.$$

  This is small in expectation.

- Given a signalling with payoff $\geq 0.5 + \varepsilon$, you can output $\Omega(\varepsilon)$ planted cliques in G (with high probability).

  This is harder. We want to massage the signal into behaving like in the first observation.

  Given input A_G and φ,

  1. for each $\sigma \in \Sigma$, compute minimax strategy y_σ (defender) and x_σ (attacker).
     Let

     $$x_\sigma = \begin{cases} x_\sigma, & \text{if } x_\sigma \leq \frac{2}{\rho d} \\ 0, & \text{otherwise} \end{cases}$$
     $$\widehat{y_\sigma} = \ldots$$

2. Let z_σ be the solution to the linear program

$$\max_{\|z_\sigma\|_\infty \le \frac{2}{\rho d}, \sum z_\sigma(v)=1, z_\sigma(v)\ge 0} (\widehat{x_\sigma})^T A_G z_\sigma.$$

Depending on the attacker's strategy, guess what θ is; get a distribution on nature. Tke the support on nature. If the attacker does well there is a way to predict nature, and you can do well.

- Return $\mathrm{Supp}(z_\sigma)$ for all σ.

$\widehat{x_\sigma}$ not behaves like some subset, and z_σ also behaves like a subset. It finds z that is correlated with x on the graph. If the payoff is larger than what's expected, there has to be a constant fraction of the planted cliques.

The rest is calculation.

An open problem is the following. ETH-hard to get a good ε-approximate Nash. Come up with a zero-sum game from worst-case settings, so it's hard to distiguish between

- YES: there exists a good signal.
- NO: for all signals the payoff is bad.

4 ETH hardness of DkS (with perfect completeness

We show that, assuming the (deterministic) Exponential Time Hypothesis, distinguishing between a graph with an induced k-clique and a graph in which all k-subgraphs have density at most $1-\epsilon$, requires $n^{\tilde{\Omega}(\log n)}$ time. Our result essentially matches the quasi-polynomial algorithms of Feige and Seltser [fs97] and Barman [Barman14] for this problem, and is the first one to rule out an additive PTAS for Densest k-Subgraph. We further strengthen this result by showing that our lower bound continues to hold when, in the soundness case, even subgraphs smaller by a near-polynomial factor ($k' = k \cdot 2^{-\tilde{\Omega}(\log n)}$) are assumed to be at most $(1-\epsilon)$-dense.

Our reduction is inspired by recent applications of the "birthday repetition" technique [AIM14,BKW15]. Our analysis relies on information theoretical machinery and is similar in spirit to analyzing a parallel repetition of two-prover games in which the provers may choose to answer some challenges multiple times, while completely ignoring other challenges.

Definition 4.1: DkS, the **densest k-subgraph** problem, is as follows. Given a graph $G = (V, E)$, define $\mathrm{den}(S) = \frac{|(S\times S)\cap E|}{|S\times S|}$. The decision problem is to decide between the following 2 cases. ($C > S$)

1. YES: there exists S, $|S| = k$ such that $\mathrm{den}(S) \ge C$.
2. NO: for all $|S| = k$ such that $\mathrm{den}(S) \le s$.

In the algorithmic (search) version, actually find the S.

Upper bound: The state of the art gives a $O(n^{\frac{1}{4}})$-approximation (BCCFV) using SDP. This is the best that can be obtained using LP.

Lower bound:

- DkS is $\left(1+\frac{1}{c(k)}\right)$-approximable in time $2^{(\lg n)^k}$ assuming RETH. [Khot 2001]
- Any constant inapproximable assuming variant of SSE (small set expansion) [RS10]
- Any constant inapproximable assuming Feige's conjecture/hidden clique [Alon et al, 2011].

Here it's not distinguishing between dense and less dense graph; it's distinguishing between cliques and less dense cliques.

Our result is the following. It is ETH-hard to distinguish in $n^{\widetilde{O}(\lg n)}$ time the following.

- YES: $\exists S$, $|S| = n^{1-o(1)}$ such that $\text{den}(S) = 1$,
- NO: $\forall S$, $\text{den}(S) \leq 1 - \varepsilon$.

This matches the upper bound: there exists a dynamic programming algorithm in $n^{O(\ln n)}$ time [Feigel, Setter 97]. Barak 15 gives an additive approximation [Barak 15].

The reduction is simple, but the analysis is hard.

The idea is birthday repetition. With high probability, two $O(\sqrt{n})$-size subsets of a n-sized set share an element.

Start with Dinur's instance of 2CSP: Given a 3SAT instance, there is a CSP with gap $(1-c, 1)$. Each vertex corresponds to assignments on $\sqrt{n}$ variables. The number of vertices is $|V| = \binom{n}{\sqrt{n}}|\Sigma|^{\sqrt{n}}$. Edges (u, v) are between vertices that

1. are equal on the variables they share (consistency), and
2. they do not violate any 2CSP constraints.

There are clusters labeled by variables we're giving assignments to.

If the prover is limited to choosing at most 1 vertex from each cluster, then the constant gap is inherented into the instances.

Choosing 2 random clusters, you have likely have to check something: consistency or constraints.

Idea: If you choose 2 vertices from the same cluster they can never give a match; you lose some edges.

The main technique/observation is that in this setting.

Fix a subset of vertices S of size k. Choose a vertex randomly from S, and define the following random variables: $X_i = 1$ if the ith variable participates, and Y_i is the assignment. The entropy of (X, Y) is $H(X, Y) = \lg k$ because the variables X, Y give a full description of the vertex; every vertex corresponds to a unique value of (X, Y).

$$\lg k = H(X, Y) = \sum_i H(X_i | X_{<i} Y_{<i}) + \sum_i H(Y_i | X_{\leq i} Y_{<i}).$$

The first sum is the entropy from the choice of variables, and the second sum is the entropy from the assigment. If one is large, the other is small.

1. If the second sum is large, then one must lose from consistency. (You're choosing multiple assignments for a given set, so you're losing in the consistency checks.)

   This is trickier. We'll sketch the proof below.

2. If the first sum is large, then the 2CSP value must carry over. (This is exactly what we want. Each variable occurs as it should be with some slack. If they occur around where they should be, the constraints appear as they should be so carry over the value of the 2CSP.)

Assume $\sum H(Y|X_{\leq i}, Y_{\leq i}) > 5\varepsilon_1 \left(\frac{\ln k}{\ln n}\right)$.

Definition 4.2 (Good variables)**:** i is good if

$$H(X_i|X_{<i}, Y_{<i}) \geq H(\mathbb{P}[X_i = 1]) - 2\mathbb{P}[X_i = 1] \ln |\Sigma|$$
$$H(Y|X_{\leq i}, Y_{<i}) \geq \frac{1}{2}\varepsilon_1 \mathbb{P}[x_i = 1].$$

We get

$$\sum_{\text{good } i\text{'s}} \mathbb{P}[X_j = 1]^2 \geq \frac{\left(\frac{1}{5}\varepsilon_1 k\right)^2}{n(\ln |A|)^2}$$

We say ω_{i-1} is good if

- i is good,
- $\sum_{\sigma_{i-1} \in N(\omega_{i-1})} \mathbb{P}[X_i = 1|\sigma_{i-1}] \geq (1 - \varepsilon_2)\mathbb{P}[X_i = 1]$. There is substantial mass on the neighborhoods.
- $H(Y_i|\omega_{i-1}) \geq \varepsilon_3 \mathbb{P}[X_i = 1|\omega_{i-1}]$.

$\text{den}(S) < 1 - \delta_1$, $\text{den}(S) < 1 - \delta_2$.

We tried applying this to small set expansion Here $k = \binom{n}{\sqrt{n}}$, $|V| = \cdots |\Sigma|^{\sqrt{n}}$. But in small set expansion we need k to be linear.

Bounding integrality gaps: the amount you can cheat if you assign something to a local set. Nice technique for "pick k" problems. Sparse PCA? But in the hidden-clique based reduction, we rely heavily on the background noise being random.

5 Bitcoin and game theory

I'll speak tomorrow about some game theoretic attacks against bitcoin from these two papers (and give a brief high-level overview of bitcoin first):

`http://arxiv.org/abs/1311.0243`

`http://arxiv.org/pdf/1411.7099v2.pdf`

Bitcoin is "just" a ledger. (Don't think of bitcoins as digital objects moving. Because people agree on a ledger, the coins have value.)

Every node i has a ledger L_i.

Goal: get all nodes to agree on L.

1. If two parties want to transact, they announce it to the world.
2. At every time t, the network selecs a random node i (miner) and this node proposes a list of transactions and a pointer to L_i.
3. Every other node validates all these transactions. Update L_j to be the ledger that has been updated the most. (Accept updates from the "correct" node.) (Longest blockchain.)

We want to prevent the double-spend attack.

1. Assume at least 51% of the network is honest.
2. Wait k steps after a transaction is proposed before considering it safe.

In order to undo a transaction from time t, for some $C \geq k$, the attacker gets chosen more often than the rest of the network.

"Proof of work." We replace the above steps by the following.

0. Everyone agrees on a collision-resistant hash function H with 256-bit output.

2'. At any time any node can propose updates$|L_i|$nonce.

3'. Accept updates if H (updates—L—nonce) $< 2^c$ for some agreed c.

For some agreed C. Set L_j =most updated ledger.

If you successfully propose transactions, you get to add a special transaction creates coins for you. This is called "mining."

5.1 Selfish mining

A selfish update his own blockchain without releasing it until later, so the rest of the network is wasting its time. Then you have a higher proportion of effective power in the network.

If you are in a position where you can always win a tie: immediately report when someone else reports.

Define lead= $l - k$. There are 2 ways to have a lead of 0: (0.) you agree, or (0′.) you found both found the same number of blocks.

Let α be your fraction of computing power. From 0 you can get to 1. Let $\gamma = \mathbb{P}$ (choose to extend yours).

- $0 \xrightarrow{1-\alpha} 0$.
- $1 \xrightarrow{1-\alpha} 0'$.
- $1 \xrightarrow{\alpha} 2$.
- $2 \xrightarrow{\alpha} 1$.
- $0' \to 0$: $\gamma(1-\alpha)$ win, $(1-\gamma)(1-\alpha)$ lose, α you found another block and win.

Find the stationary probabilities; how much does each person get?

The fraction of revenus is $> \alpha$, if $\alpha > \frac{1-\gamma}{3-2\gamma}$.

6 Lifelong learning

6.1 Intro

In PAC learning for boolean functions, we consider (feature domain) $X = \{0,1\}^n$, $Y = \{0,1\}$, distribution D over X, $t : X \to Y$. Given $\{x_i : t(x_i)\}_i \sim D$ iid, we want $h(x)$ such that

$$\mathbb{P}(h(x) \neq t(x))$$

is small.

In lifelong learning, we have a sequence $(t_i, D_i)_{i=1}^m$ (arriving in online fashion), t_i sharing meta-features $\{m_j\}$ (common structure).

There are different learning tasks with common structure. There may be common features involved in identifying dogs and identifying cats.

The goal is to learn the meta-features $\{m_j\}$ and the target function $\{t_j\}$.

Example 6.1: t_i are conjunctions. There exist k monomials such that t_i is a conjunction of some subset of the monomials.

For example, $t_1 = x_1x_2x_3$, $t_2 = x_2x_3x_4$, and $m_1 = x_1x_2, m_2 = x_2x_3, m_x = x_4$. Here $t_1 = m_1m_2$, $t_2 = m_2m_3$.

Example 6.2: t_i is a linear separator, and $\{t_i\}$ are in a k-dimensional subspace.

Because the set of meta-features is not unique, it suffices to output $\{\widetilde{m_j}\}$ that has equal representational power.

6.1.1 Conjunctions

First consider the **consistency** problem: given $\{t_i\}_{i=1}^m$, find k monomials $\{\widetilde{m_j}\}$. This is the set basis problem which is NP-hard. Thus we make the **anchor variable** assumption: for all m_j, there exists $y_j \in m_j$, $y_j \notin m_{j'}$ for $j' \neq j$.

Once you've found the anchor variables it's easy to find: let $\widetilde{m_j} = \bigcap_{t \ni y_j} t$.

The algorithm is as follows. Let $TS = \{t_i\}_{i=1}^m$,

$$NS(TS, Z) = \{t \in TS : z \in t\}$$
$$H(t) = \bigcup \{\widetilde{m_j} : \widetilde{m_j} \subseteq t\}.$$

Repeat:

1. Set $t = 0$, $\widetilde{M} = \phi$ (metafeatures selected so far).
2. While $t \in TS$ cannot be represented by the current metafeatures, $t \neq H(t)$,
   (a) Choose t with minimal index.
   (b) Minimal: Choose z_{k+1} from $t \backslash H(t)$ there doesn't exist $z' \in t \backslash H(t)$, $N(TS, z') \subseteq N(TS, z_{k+1})$.

3. Set $\widetilde{m_{k+1}} = \bigcup_{T \ni z_{k+1}} T$.

Definition 6.3: Say $\widetilde{m} \equiv m$ if

1. (larger than ground truth metafeature) $m \subseteq \widetilde{m}$
2. (not too large) for all $t \in TS$, if $t \supseteq m$, then $t \supseteq \widetilde{m}$.

The guarantee of the algorithm is the following.

Lemma 6.4: Let $\{\widetilde{m_j}\}$ be the output of the algorithm. For $\widetilde{m_j}$ there exists m_{t_j},

1. $\widetilde{m_j} \equiv m_{t_j}$ (each constructed feature maps to a ground truth metafeatures)
2. $m_{t_j} \neq m_{t_{j'}}$, $j \neq j'$.

(In particular, the number of ground truth metafeatures is at most k.)

Proof. Induct. The base case is clear.

Assume that it's true for $i' \leq i$. Now we show the theorem for $i+1$.

We have

$$\widetilde{m_{i+1}} = \bigcap_{z_{i+1} \in T} T.$$

$m_{t_{i+1}}$ is the m containing z_{i+1}, $m \subseteq t$.

2. It's not covered by the previous: OK.

1. Problem: z_{i+1} may not be an anchor variable. Suppose z_{i+1} is not an anchor variable. Then $y_{t_{i+1}} \in \widetilde{m_{i'}}$. We claim that $m_{t_{i+1}} \subseteq \widetilde{m_{i'}}$. Now $\widetilde{m_{i'}} = \bigcup_{z_{i'} \in T} T$, so $y_{t_{i+1}} \in T$. So $m_{t_{i+1}} \subseteq T$.

   We claim $N(TS, z_{i+1}) = N(TS, y_{t_{i+1}})$. Choose the minimal variable in the sense that if you want to construct the metafeatures...; pick anchor variable at each step and construct a metaf equivalent to ground truth metaf.

□

Given such a subroutine, each time you receive a new learning task, try to construct it from the previous metafeatures. If you succeed with error 0, go to the next task. Otherwise, take all the learning tasks up to now, and run this algorithm again. Whether this algorithm is efficient depends on whether you can learn the conjunction under the previous metafeatures. Under some query models you can do this efficiently.

Can you learn the new conjunction based on the previous metafeatures?

6.1.2 Neural nets

Consider the t_i as images. Metafeatures are ANDs of corresponding variables. If there is a pattern in the image, the corresponding feature is activated. Then take OR of the metafeatures.

Think of the metafeatures as the hidden layers; this is the corresponding boolean autoencoder. Here the hidden layer is the same size. What if it's large, but we require sparse coding. We can only use a few metafeatures to reconstruct. Each t_i is a conjunction of at most k.

Sparse boolean autoencoder: $\{t_i\}_{i=1}^m$, $\{m_j\}_{j=1}^L$, $L = O(n)$, t_i is a conjunction of $\leq k$ of the m_j. They consider $|m_j| \leq C$.

The assumption is that for all m_j, there exists $y_j \subseteq m_j$, such that for all $t \supseteq y_j$, $t \supseteq m_j$, $|y_j| \leq C$. Any containing this set must contain the whole metafeature. This is more general than the anchor variable assumption (for which $|y_j| = 1$). $|y| \leq c, \widetilde{m_y} = \bigcup_{y \in T}$. This corresponds to learning polynomials. (Anchor words assumption is stronger than the unique neighbor property. Fix m_j, there is x_i only appearing in it.)

7 Noise in computation

We'll talk about the circuit computation model. (There might be other models that are appropriate but this is the most obvious.)

Some motivations:

- The relation of communication to computation.
- Understanding computation.
- Practical considerations.

The short history: people (von Neumann) cared about it in the 50's, people made gates that have high reliability and stopped thinking about it. In practice most architectures assume the computation is infallible. However there is room for improvement in terms of power consumption.

The basic setup is that we have a circuit with some gatewise noise. Define

$$\mathrm{NAND}_\varepsilon(x, y) = \begin{cases} \neg x \wedge y, & wp\, 1 - \varepsilon \\ x \wedge y, & wp\, \varepsilon \end{cases}$$

What/how is this good for?

Given a circuit C of NAND's is there a C' of $NAND_\varepsilon$'s that does the same thing and is there a meaningful expansion parameter? We can allow C' plus $o(|C|)$ noiseless gates (or allow some noise). Spielman in the 90's had the following solution: store each word as an error correcting code. Do operations on these words. This causes an expansion of $\ln |C|$ (the length of the word). You have to allow a threshold gate at the bottom.

Can we get constant in some meaningful regime? There is no inherent reason why you shouldn't be able to.

We start with the communication connection. We can ask this problem in any setting. One successful setting is data transmission and storage. (We can't assume long-term memory is noiseless!) There is the concept of channel capacity. The capacity

$$Cap(BSC_\varepsilon) = 1 - H(\varepsilon).$$

This tells us how much more communication we need given an amount of noise. The profound fact is not that this is the conversion factor, but that there is one. This is the big discovery of Shannon. You can separate the encoding from the application; it wasn't obvious in the 40's. Is this true here?

A more accessible model is the interactive error-correcting codes: you can assume there is a lot of communication, but each communication is small. You carry a constant loss. This is smaller than one-way capacity. The capacity hasn't been formally defined. Given a generic pointer problem, what's the factor you need to blow up? It's a constant factor.

This is connected via the Kharmarcher-Wigderson games. This is a connection between boolean formulas and interactive protocols. A boolean formula $f(x_1, \ldots, x_n)$ corresponds to the interactive protocol: A is given $x \in f^{-1}(0)$ and B is given $y \in f^{-1}(1)$. Output i such that $x_i \neq y_i$. This is still the most promising attack to get lower bounds. If you prove a lower bound on the game you prove a lower bound on the depth. Starting at a disagreement at the bottom of the circuit (output), move up. Each bit they jump 1 layer by sending the index of the 1 or 0 in the previous layer (depending on whether it's AND or OR).

This protects against short-circuit errors (Kalai, L, R 2013). A short-circuit error means that we can have $AND(0,1) = 1$ but not $AND(0,0) = 1$. It doesn't protect against flips.

Interactive error correction is just an inspiration for the kind of results we want.

We don't know the error for the simplest channels. It's bounded by above by the one-way capacity; we don't know how big the gap is.

Defining a constant and asking whether it's positive is interesting.

The problem we gave is analogous to channel capacity. This correspond to asking does the conversion to $NAND_\varepsilon$ have constant rate.

What is the analogue of entropy? Given T, a circuit C, what does $\frac{C(T^n)}{n}$ approach? What is going on for multiparty computation. This would be a great deal: try to break out of discreteness. Coding theory and information theory are more successful than complexity theory because they are not discrete.

Next problem: The more abstract problem is that given $f : [0,1] \times [0,1] \to [0,1]$, when can you use it for computation? Given $f : [0,1] \times [0,1] \to M[0,1]$. (The output is a distribution on the interval.) Is it meaningful to talk about the log factor? The relationship between error and power (energy consumption of a circuit) is $e^{-\Theta(p^2)}$.

For a lot of time the limiting cost was hardware but this is changing now: hardware is cheap relative to power. We cannot hope to reduce it by much. Reducing it from 10^{-18} to 10^{-9} is a 30% reduction. (Error in lifetime vs. error per second.) Checksum for interesting computation. Ex. A multiplier that can withstand three errors. You're not allowed to blow up the size of the circuit. Ideally verify the computation in sublinear amount of gates compared to the original computation. 1 error: checksum. PSPACE solver. Usually call more than once. Call the solver once. Allowed sublinear overhead. Only need to correct few errors, not blackbox. Multiplier takes superlinear number of gates.

In terms of approachability, the three most approachable directions are

1. abstract: compute capacity (analogue of channel capacity)
2. specific problems: checking for low complexity classes. Merge PCP theory with low-level complexity classes?
3. practical stuff. (Dead for a while, but revived because of the power problem.)

Solved problem in wrong regime.

Seems unrelated: A simple proof of the following fact. A cellular automata can simulate a Turing machine. Can you do it robustly? In one dimension you can keep one bit of information (Gacs, 70s). With a more modern understanding, simplify? Godel's theorem for logic of computation is a trivial fact (there's pain in proving the arithmetic version). 1-dimension: store a bit. Trying to fight mutation. Build an immune system. Mutation with mechanism for doing exactly what you're trying to do: once in 2^{-n} steps have a mutation of size n. In 2+ dimensions you can do robust computation. Two states that don't mix. Kill all neighbors not like you, etc. What if overpower, etc.

8 Dictionary learning with VERY few examples

Authors: Klye Luh, Van Vu in FOCS 2015

We consider the equation

$$Y = AX$$

where Y is $n \times m$, the input matrix; A is $n \times n$. For dictionary learning, X is sparse and A are the items in the dictionary. When X is extremely sparse (each column has constant number of zeros) this problem is also called sparse coding.

Theorem 8.1: If A is full rank and X_i are iid sampled from $\begin{pmatrix} 1-\theta & \theta \\ 0 & \xi \end{pmatrix}$ where ξ is such that $\mathbb{E}\xi = 0$, $\theta \in [\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}]$. $\mathbb{P}(\xi \geq t) \leq e^{-\frac{t^2}{2}}$.

There exists a polynomial time algorithm that exactly recovers A, X when $m \geq n(\ln n)^4$. Information theoretically we can go to $n \ln n$ because we need that many equations to certify the variables.

(n^2 many variables, $\ln n$ bits of entropy from choice in column.)

$\text{span}(Y') = \text{span}(X')$. Each row of X is a sparse vector. Dimension much higher than number of vectors. It makes sense that the sparsest vectors in the span are the rows of X. If we can solve

$$\min_{b \in \mathbb{R}^n} \left\| b^T Y \right\|_0,$$

$b \neq 0$, $b^T y X^i c$. We can recover at least one X^i. But these are nonconvex constraints. The first constraint is easy: the $L_0 \to L_1$ relaxation,

$$\min_{b \in \mathbb{R}^n} \left\| b^T Y \right\|_1.$$

The second constraint is problematic. (Dropping it we get $b = 0$ as the solution.) We restrict $\langle b, v\rangle = 1$. This is a trick from Spielman. v is a column of Y.

Ideally we want $b^T Y = X^i$. $b^T AX = X^i$. We should have $b = (A^T)^{-1} e_i$. It's natural to put $v = A^T e_i$. We cannot really get this v because we don't know A. The analogue of v is the column $Y_j = AX_j$. The jth column is quite sparse, so it's like e_i. If we choose $v = Y_j$, can we solve the problem?

The naive approach is to minimize $\left\|b^T Y\right\|$ such that $\langle Y_j, b\rangle = 1$. Make a change of variables $b = (A^T)^{-1} r$. Then equivalently we want min $\left\|r^T x\right\|$ such that $\langle x_j, r\rangle = 1$.

Observation: for all r,

$$\left\|r^T X\right\|_1 \approx mf(\|r\|_1).$$

f is monotonically increasing. If X is 1-sparse, equal to 1-norm. If X is sparse, roughly proportional. m summations of inner product, grows linearly as m.

For the simple case, consider $\left\|r^T x\right\|_1 = mf(\|r\|_1)$. Then the convex program is equivalent to

$$\min \|r\|_1$$

such that $\langle r, X_j\rangle = 1$. X_j is a sparse column vector with $\pm 1, 0$. Putting $\frac{1}{2}$ on $1, 0$... This convex program does not recover ...

The trick: don't naively consider $\langle r, X_j\rangle = 1$. Consider $\langle r, X_p + X_q\rangle = 1$. Similar argument with unique neighbor: with decent problem have unique common nonzero place. With high probability same sign. A 2. This gives b_i. You can put $\frac{1}{2}$ on the 2. Norm can be as small as $\frac{1}{2}$. This is robust to approximately equal. Suppose only have approximately equal. We want to minimize

$$\min \sum (1 \pm S_i)|r_i|.$$

such that $2r_1 + r_3 - r_6 = 1$ (for example). No matter what δ is, still unique solution: put $\frac{1}{2}$ at r_1.

Final algorithm: go through p, q,

$$\min \left\|b^T Y\right\|, \langle b, Y_p, Y_q\rangle = 1.$$

$X_{pq} = b_{pq}^T Y$. Add X_{pq} from $\|X_{pq}\|_0$ smallest to largest. ...sparsest until rank is n. Recover X can recover A because whp X is right-invertible. Actually recover coefficients of dictionary first, use the representation to recover the dictionary. Until rank$= n$.

Observation: for $\|r\|_0 \le \sqrt{n}$ (sparse r), $\mathbb{E}\left\|X^T r\right\| = mf(\|r\|_1)$. Consider X is 1-sparse, just 1 norm of r. If $\sqrt{n}$ sparse...

Also using unique neighbor property hit one value.

For dense r, do some calculation, still increasing function of r. Expectation calculation.

Step 2: for all r, $\left\|\gamma^T X\right\|_1$ concentrate to $\mathbb{E}\left\|r^T x\right\|_1$. (Spielman used Bernstein, this is fancier.)

$$\mathbb{P}(\exists r \left\|\gamma^T x\right\|_1 - \mathbb{E}\left\|r^T x\right\| \ge \delta \mathbb{E}) \le o(1).$$

target solution is 1-sparse, first step rule out $\sqrt{n}$. How prove theorem?

How did Spielman prove this theorem? $\left\|r^T x\right\| = \sum_{i=1}^m |\langle r, x_i\rangle|$.

$$\mathbb{P}(\left\|r^T x\right\|_1 - \mathbb{E}\left\|r^T x\right\|_1) \leq \exp\left(-\frac{t^2}{\operatorname{Var}\|r^i x\|_1}\right),$$

$$\operatorname{Var}\left\|r^T x\right\| = \sum_{i=1}^m \operatorname{Var}|\langle r, X_i\rangle| = \sum_{i=1}^m \langle r, X_i\rangle = \sum_{i=1}^m \sum_{j=1}^n \mathbb{E}(X_{ij}^2 r_j^2).$$

Square, cross term cancel. With prob θ nonzero, once nonzero

$$= \sum_{i=1}^n \theta \sum_{j=1}^n r_j^2.$$

Why old thing cannot give optimal. Because expression in $\mathbb{P}$ is scaling-invariant, if true for 1-norm $= 1$, true for all r. Here if the 1-norm $= r$, then value < 1.

$$\leq \theta r n.$$

Then

$$\mathbb{P}(... \geq t) \leq \exp\left(\frac{-t^2}{\theta m}\right).$$

How small can the expectation be? $\mathbb{E}\left\|r^T x\right\|_1 = m\mathbb{E}\|r_i, x_i\|$. $(\frac{1}{\sqrt{n}}\cdots)()$. WHen nonzero iid ± 1. Sum of ± 1s are $\sqrt{\theta n}$.

$$\min \mathbb{E}\left\|r^T x\right\|_1 = m\sqrt{\frac{\theta}{n}}.$$

$t = \sqrt{\frac{\theta}{n}}$.

$$\exp\left(\frac{-t^2}{\theta m}\right) \leq \exp(-m/n).$$

Union bound: e^m many r's. Only consider rs differing by $\geq \frac{1}{n^3}$. $\frac{1}{n^3}$ ε-net, contain e^n? many points.

$r_i = \frac{k}{n^4}$ where $k \in \mathbb{N}$. How many with $\|r\|_1 = 1$? e^n. $= 2^{n^4}$.

Union $e^n e(-\frac{m}{n}) \approx o(1)$ so get n^2.

Variance bound only tight when 1-sparse: that's how they improve analysis.

When take union bound, $\mathbb{P}(A \cup B) = \mathbb{P}(A) + \mathbb{P}(B) - \mathbb{P}(A \cap B)$. For a union bound we ignore the last term. But it's large for many different r's. Two close by points would be close in value whp.

$$\left|\left\|r_1^T x\right\|_1 - \left\|r_2^T x\right\|_1\right| \leq \left\|(r_1 - r_2)^T x\right\|_1$$

Consider this term, apply concentration on this term. $r_1 - r_2$ has property that each term is small. Concentrate better than naive approach.

Why does the intersection help here? When apply concentration bound... singular vectors is 2-norm. When 2-norms only shaves log at best. Like ARV. Now the difference is $\frac{1}{n^4}$.

Coarser net 2ε-net.

$$\mathbb{P}(\exists \text{bad point}) = \mathbb{P}(\exists 2\varepsilon\text{-bad point}) + \mathbb{P}(\exists \text{bad point} | \exists 2\varepsilon \text{ points all good}).$$

Much smaller than union bound. just consider 2 adjacent. Repeat again and again. Make sure the second term is very small.

Amazing property of 1-norm: if you build an α-net on $\|r\|_1 = 1$, $r_1 = \alpha\mathbb{Z}$, then the number of net points is $\exp\left(\frac{\ln}{\alpha}\right)$. If $\alpha = \frac{1}{2}$, only polynomially many points. L_∞ net because each index. View L_∞ net on L_1 norm, get exponentially decrease in number of points. ... When net becomes coarse you eventually get polynomially many.

What about L_2? L_2 ε-net.

From ε net $\|r_1 - r_2\|_\infty \leq \alpha$. Goal:

$$\mathbb{P}(\left\|r_1^T x\right\|_1 - \left\|r_2^T x\right\| \geq m\sqrt{\frac{\theta}{n}}) \leq \exp\left(-\frac{100\ln n}{\alpha}\right).$$

Summation from $\frac{1}{n^4}$ to 1 still small. To show the inequality, cut the net according to expectation. Concentrate to expected value. Expectation $m\sqrt{\frac{\theta}{n}}$ to $m\theta$. Observe

$$\mathbb{E}(\left\|r_1^T x\right\|_1 - \left\|r_2^T x\right\|_1) \ll m\sqrt{\frac{\theta}{n}}.$$

$$\left\|r_1^T x\right\| - \left\|r_2^T x\right\|_1 \leq \left\|(r_1 - r_2)^T x\right\|_1.$$

Why use L_∞ ε-net. Want to use α.

$$\sum_j^n (r_1 - r_2)_j^2 < \alpha^2 n.$$

Move the summation to 1-norm; know 1-norm is 1. Use L_2 ε-net. Now we get

$$\mathbb{P}(\left\|r_1^T x\right\| - \left\|r_2^T x\right\| \geq m\sqrt{\frac{\theta}{n}}) \leq \exp\left(\frac{\left(m\sqrt{\frac{\theta}{n}}\right)^2}{\theta m\alpha}\right) = \exp\left(\frac{m}{n\alpha}\right)$$

L^2 gives α^2. The number of points is $\frac{1}{\alpha}$, the concentration is also $\frac{1}{\alpha}$. Real paper have to bound the maximum.

9 High-rate locally-correctable and locally-testable codes with sub-polynomial query complexity

Abstract:

Locally-correctable codes (LCCs) and locally-testable codes (LTCs) are special families of error-correcting codes that admit extremely efficient sub-linear time algorithms for error correction and detection, respectively, while making a few queries to the received word. These codes were originally studied in complexity theory because of their close relationship to program checking and probabilistically-checkable proofs (PCPs), but subsequently they were extensively studied for their own sake.

In this work, we construct the first locally-correctable codes and locally-testable codes with constant rate, constant relative distance, and sub-polynomial query complexity and

running time. Specifically, we show that there exist binary LCCs and LTCs with block length n, constant rate (which can even be taken arbitrarily close to 1), constant relative distance, and query complexity and running time $\exp(\sqrt{\log n})$. Previously such codes were known to exist only with query complexity and running time n^β (for constant $\beta > 0$), and there were several, quite different, constructions known.

Along the way, we also construct LCCs and LTCs over large (but constant) size alphabets, with the same query complexity and running time $\exp(\sqrt{\log n})$, which additionally have the property of approaching the Singleton bound: they have almost the best-possible relationship between their rate and distance. This has the surprising consequence that asking for a large alphabet error-correcting code to further be an LCC or LTC does not require any sacrifice in terms of rate and distance! Such a result was previously not known for any sub-linear query complexity.

If time permits I will also discuss a very recent work that substantially improves the query complexity of LTCs to quasi-polylogarithmic function of the form $(\log n)^{O(\log \log n)}$.

Joint work with Swastik Kopparty, Or Meir and Shubhangi Saraf

An error-correcting code is a map $C : \Sigma^k \to \Sigma^n$. Σ is the alphabet, k is the message length, and n is the codeword length.

Two basic parameters are the rate $\frac{k}{n}$ and the distance

$$\min_{u,v \in C} \{d(u,v) := \mathbb{P}_{i \in [n]}(u_i \neq v_i)\}$$

It equals twice the proportion of errors the code can correct (picture).

There are 3 operations associated with codes.

- Encoding: compute map C.
- Testing/detecting errors: test $u \in C$.
- Correcting: find $c \in C$ nearest to u.

We're interested in sub-linear time algorithms that do these operations. We want them to run in time $\ll n$. They can't even make 1 pass over the codeword! We need several assumptions.

- Random access: they can access each coordinate with some cost.
- Probabilistic (has a small chance of failure).

The first family is locally testable codes (LTC's), which have sublinear time testing algorithms.

Definition 9.1: A **local tester** is given as input $u \in \Sigma^n$ and outputs accept/reject. It satisfies the following:

$$u \in C \implies \mathbb{P}(\text{accept}) = 1$$
$$u \notin C \implies \mathbb{P}(\text{reject}) \geq 0.99 d(u, C).$$

The rejection should depend on the distance, because if it just differs in one location, you probably won't find it.

Locally correctable codes are codes with sublinear time correcting algorithms.

Definition 9.2: A **local corrector** is given as input $u \in \Sigma^n$ such that $d(u, C) < \frac{1}{2}d(C)$ and $i \in [n]$ and is required to output the ith coordinate of codeword closest to u with probability ≥ 0.99.

Definition 9.3: A **local decoder** is given as input $u \in \Sigma^n$ such that $d(u, C) < \frac{1}{2}d(C)$ and $i \in [k]$ and is required to output the ith coordinate of the message m such that $C(m)$ is the codeword closest to u with probability ≥ 0.99.

LDC's seem more natural, but LCC's imply LDC's. If you can decode every coordinate of the codeword, you can decode every coordinate of the message.

Fact 9.4: For linear codes, LCC's imply LDC's.

Our main goal is to minimize the number of queries and the running time.

9.1 Previous work and motivations

Most of the research has focused on a constant number of queries and constant distance. The goal is to maximize the rate. People studied this regime because it would be powerful; sometimes you can do amazing stuff with 2–3 queries (PCP). Studying this regime has applications in complexity theory. They are connected to property testing, hardness of approximation, etc.

In practice, all known constructions of such codes have vanishing rate: as $n \to 0$, rate goes to 0. This is very bad for applications in data transmission and storage.

For LTC's,

- the best constructions have rate $\frac{1}{\text{polylog}\, n}$, due to Ben Sannon-Sudan 2005, Dinur 2007, Meir 2009, and Videman 2013.
- It is not known if we can achieve a constant rate. This is the biggest open problem in this area.

There is an implicit connection to PCPs. Every time someone gets a better LTC, with the same techniques people get a better PCP, and vice versa. However there is no formal connection.

For LCC's,

- we cannot get constant rate (Katz-Trevisan 2000).
- Reed-Muller codes give $\exp(-n^{\varepsilon})$, and there are LDC's with rate $\exp(-\exp(\sqrt{\ln n \ln \ln n}))$ by Yekhanin 2007 and Efremenko 2009.

9.2 High-rate regime

We are interested in constant rate and constant distance, and we want to minimize the number of queries. Ex. $\log n$ queries is fine.

The practical motivation is to encode massive data.

1. Solution 1: break into chunks and encode each part separately.

   Pro: The recovery time is small. We only have to read bits that are close.

   Con: in the presence of errors, the failure probability is the number of pieces times $\exp(-|\text{small}|)$.

2. Encode the message in one big chunk.

   Pro: Failure probability is $\exp(-|\text{big}|)$.

   Con: The recovery time is —big—.

What is known? Up to a few years ago, the only known examples were Reed-Muller. For all $\varepsilon > 0$, you can have a code with number of queries is n^ε and the rate is $\exp(-\frac{1}{\varepsilon})$. This was believed to be best possible.

Actually you can do better. For every $\varepsilon > 0$, you can have a code with queries n^ε and rate $1 - \varepsilon$.

Here are known constructions.

- The first codes were multiplicity codes (LCC's), generalization of Reed-Muller codes. In Reed-Muller codes you evaluate polynomials; here you evaluate polynomials and their derivatives (Kopparty-Saraf-Yekhanin 2011).

  Algebraic flavor.

- Tensor codes (LTC's): BenSasson-Sudan 2006, Videman 2011.

  Combinatorial flavor.

- Lifted Reed-Solomon codes (both LTC's and LCC's), due to Guo-Kopparty-Sudan 2013.

- Expander codes (LCCs), due to Hemenway-Ostrovsky-Wooters 2013.

  Graph-theoretic.

9.3 Our results

We show that we can improve the number of queries to be subpolynomial. We give high-rate LTCs/LCCs with subpolynomial number of queries

Theorem 9.5: For every $\varepsilon > 0$ there is a LCC and a LTC with number of queries equal to $\exp(\sqrt{\ln n \ln \ln n})$ and rate $1 - \varepsilon$.

Theorem 9.6: For all $\varepsilon > 0$ there is a LTC with the number of queries $(\ln n)^{O(\ln \ln n)} = \exp((\ln \ln n)^2)$.

Additionally, we get a code which approaches the singleton bound with distance $\sim \varepsilon$.

9.4 Proof overview

1. Component 1: Get high-rate LTC's/LCC's with subpolynomial queries with subconstant distance.
2. Component 2: Apply distance amplification (due to Alon-Luby 1996).

- For the first part, to get LTCs with $\exp(\sqrt{\ln n})$ queries, we use tensor codes of high dimension.
- For LCCs with $\exp(\sqrt{\ln n})$ queries, we use multiplicity codes of high dimension.
- For LTCs with $(\ln n)^{O(\ln \ln n)}$ queries, use an iterative construction. Alternate tensor product and distance amplification.

This is similar in spirit to techniques/results in complexity theory:

- Zigzag product (Reingold-Vadhan-Wigderson)
- $SL = L$ (Reingold)
- Gap amplification (Dinur)

Rather than just use tensor products, alternate tensor product and amplification.

We use ideas from combinatorial constructions of LTC's due to Meir 2009.

Component 2 is based off of Alon-Luby 1996, based on expanders. Given a LTC/LCC with the following parameters:

- distance $\delta' = o(1)$,
- rate r,
- number of queries q,

for every δ, ε, we obtain a code with the following parameters

- distance $\delta' = \Omega(n)$, $\delta \gg \delta'$,
- rate $r(1 - \delta - \varepsilon)$,
- number of queries $q \operatorname{poly}(\frac{1}{\delta'}, \frac{1}{\varepsilon})$.

Previously they used this to increase the distance. We actually amplify distance of local codes from subconstant to constant.

The application is more to classical data transmission and storage. For appications to complexity, we usually want a constant number of queries.

10 Evolution and dynamical systems/Markov chains

Look at dynamical systems that arise in nature.

We'll find relationships with

1. Markov chains/mixing time
2. algorithms/optimization
3. Fourier analysis on the Boolean cube
4. computational complexity

How can we model and understand molecular evolution? It involves

- replication
- selection
- muations, variability

Think of viruses. This happens at a fast time scale.

What are the origins of life? Apparatus, machines which combines, comes together, to self-replicate.

Outside of TCS and math, people have multiple points of view.

Evolution of language also follows simple principles.

A dynamical system (discrete, continuous) is

$$x(t+1) = f(x(t))$$
$$x'(t) = f(x).$$

A flow that tells you where to go.

Ex. a slime mold's dynamical system gives it the ability to compute. We want to explain how, why, and how fast.

10.1 Equations of life

Consider a situation where you have a population with m different types, $[m]$. Let $x_i(t)$ be the proportion of type i at time t. We have $\sum_{i=1}^m x_i(t) = 1$.

We consider a simple model. We need to introduce more parameters. Let $a_i > 0$ be the fitness of i. A first attempt is to let $x_i(t+1) = a_i x_i(t)$. However, nature puts the constraint $\sum_{i=1}^m x_i(t) = 1$. We renormalize by setting

$$x_i(t+1) = \frac{a_i x_i(t)}{\sum_{j=1}^m a_j x_j(t)}.$$

We've introduced reproduction and selection.

We need to introduce mutations (errors). Let Q_{ij} be the fraction of unit mass that mutates to j.

There are 2 time scales: start with a population and see how it changes, vs. environment changes.

Let $A = \text{diag}(a_1, \ldots, a_m)$. Let

$$x_i(t+1) = \frac{\sum_j a_j x_j(t) Q_{ji}}{\sum a_j x_j(t)},$$

so

$$x(t+1) = \frac{QAx(t)}{\|Ax(t)\|_1}.$$

Why does this have a steady state? If $Q, A > 0$, then it has a dominant right eigenvector by Perron-Frobenius.

So the unnormalized values at time t is $(QA)^t x(0)$.

We look at dynamic systems $f : \Delta_m \to \Delta_m$. This subclass is more special and interesting. No matter where you start from, you will converge to a fixed point. Spectral ratio tells you how many iterations you need.

Do all dynamical systems of this type f have a unique fixed point? Can more complex behavior happen? There can be multiple fixed points. The first hurdle is existence of multiple fixed points.

- Multiple fixed points.
- Periodic orbits/limit cycles.
- Chaos. (Let's not discuss this.)

These do come up in evolution of languages and origins of life.

Stability is important when we're talking about equilibria: nearby trajectories converge.

10.2 Finite populations

We thought of populations as infinite. Here we have actual particles, say N of them. We introduce stochasticity in evolution. We don't get a dynamical system, but a Markov chain. Each particle of type i replicates to a_i particles. Use the stochastic matrix to make the mutations. We have replication and mutation. Now do selection: sample N points with replacement. M and sampling are stochastic.

What is the number of states of the Markov chain? Think of $N = 1000$ as large, $m = 50$. The number of states is m^N (or $\sim N^m$ since order doesn't matter). The state space is large.

These models are used to simulate how viruses evolve. Track certain parameters. How much time does it take to reach a steady state?

Are these methodologies sound?

In theoretical computer science we don't simulate. Do they converge quickly?

Mutagenic drugs try to increase the stochasticity and remove genetic info from the system. How many life spans evolve before it loses genetic information? If it's much more than your life span...

- computationally

- feasibility of
- could life have evolved quickly enough? (rapidly mixing)

Mutation vs. steady state. Let (Q, A, M) be the parameters of f. How do solutions change when we change Q, A, M? Let's focus on Q. Assume $Q_{ii} = 1 - \mu$ and $Q_{ij} = \frac{\mu}{m-1}$: replicate correctly with probability $1 - \mu$ and make a random error else. What happens as μ increases? For this system (proposed by Eigen and Schuster in the 70's, Eigen was a Nobel laureate in chemistry), as μ increases from 0 (not far from 0), there is a point (depending on m) where the vector goes thorugh a phase transition, called the **error threshold**. μ decreases with m. This observation is the basis of mutagenic strategies.

Their motivation was not to model viruses. They were trying to understand the information capacity of a system. How much genetic information can a system maintain? Assume $\mu \sim \frac{1}{m}$. If $\mu > \frac{1}{m}$ then the output is junk/random/no genetic information. If $\mu < \frac{1}{m}$ then it preserves genetic information. We want $\mu < \frac{1}{m}$; a drug designer wants $\mu > \frac{1}{m}$.

Go back 40 million years ago. μ is a thermodynamic parameter. It turns out, based on simplistic models, that the largest bit content is 50. How long is DNA? Billions, 10^9. You will hit a bottleneck less than a billion.

$(1 - \mu)^L \sim \frac{1}{L}$.

Theorem!

So many creatures, why only 1 way to store information? If everything started from scratch, organisms should have a different way to store information! The more you think the more you stop believing (simplistic accounts of?) evolution.

10.3 Hypercycle

Suppose there are N types. The differential equations are

$$\dot{x}_i = k_i x_i x_{i+1} - x_i \sum_j k_j x_j x_{j+1},$$

$\sum \dot{x}_i = 0$. There are mechanisms in stars which have this form (Bethe-Weierstrass cycles). Autocatalytic cycles. How can we bypass the error threshold? Why don't we see alternate forms of information-organizing systems?

Think of each system as a collection of proteins, RNA's which have nice properties. Another collection of things that have nice properties. Can use A to do B, B to do C. Functional linkages. For this to survive, you must justify there is a solution to this dynamical system which lives inside the simplex. If one type goes to 0 it kills the cycle. Conjectured there is a limit cycle inside the interior, unique limit cycle. Proved in late 1980's by mathematicians. There are still open questions.

Mathematically, for this version, we understood how much time to get to the steady state.

Not obvious. Important: things have to stabilize in some sense. This was studied by Papadimitriou and Visheet. Use techniques from computational complexity. There is something special about these dynamical systems. If can choose coefficients, then PSPACE-complete. (Given the parameters of the dynamical system, give a point close to the orbit. cf. Nash

equilibirum.) For specific dynamical systems, what can we do? We are in complete lack of mathematical techniques. Part of the interest is not explaining biology and science, but also: are there mathematical techniques that we still lack.

In a (recently terminating) series of results DSV12, V15, PSV15, we proved a large class of evolutionary dynamical systems mix rapidly. (Finite sampling.)

10.4 Language

Here's a problem which bothers us. When a baby is born, how does it learn how to speak? We know how to teach a baby a language.

Chomsky proposed universal grammar: a baby is born with a set of grammars. It tries to learn from examples. Uniform distribution of grammars that slowly converges.

Question: How big a grammar? This depends on how good the learning is. Let's model it.

Novak et. al. proposed a model in 2001. Suppose there are 10 grammars $Q_1, \ldots, Q_m$. What is the relation between the ith and jth grammar. What is the probability that a sentence/word from i makes sense in j? Keep track of populations $x_1, \ldots, x_m$: confidence in grammars. The model is: define $b_{ij} = \frac{F_{ij}+F_{ji}}{2}$, pairwise compatibility of 2 grammars. Fitness.

Probability of confusion is (mutation) $Q_{ij} =$, probability that trying to learn i, picking up something from j.

We can define the fitness of a grammar

$$a_i = \sum_j b_{ij} Q_{ji} x_j :$$

knowing i, how well you can communicate with rest of population? a_i has to do with the learning. $Q_{ii} = 1 - \mu_i$, $Q_{ij} = \frac{\mu}{m-1}$.

Quadratic f, $x(t+1) = f(x(t))$. Study the steady state vector as a function of μ. As you increase the mutation rate, all points close to corners are stable. They slowly disappear and only the point in the center is stable. The Tower of Babel. They call it the coherence threshold.

Batch learning, etc. Heuristic arguments: how big the grammar has to be as a function of the learning environment. cf. we got some bound on M given μ. Recently we're looking at how the mixing time behaves. Because of multiple equilibria, can prove phase transition. Not a good idea to have multiple fixed points in dynamical systems where we're studying finite versions. Parameters fine-tuned slowly. Prefers dynamical systems with nice equilibria?

Different models of learning, do we need the notion of universal grammar. Here's a learning mechanism for which the size of the grammar is small. Less information have to be born with, ability to teach how to speak.

Techniques for quadratic dynamical systems. Fixed point. Move in parameter space. Determined by Jacobian, eigenvalues. Smooth functions. All eigenvalues real part negative. Converge from every direction. Change parameters, then these properties violated. Hopf fibration (?). Can understand convergence from close-by points.

Algorithmic: finite sampling.

Now can do linear programming via slime. What optimization? Riemannian manifold.

11 Dynamical systems and space-bounded computation

A dynamical system is function $f : X \to X$. We care about compact $X \subseteq \mathbb{R}^n$. This is the most general model for physical systems. X is the state space, f is the transition. Very simple functions f can give rise to chaotic behavior.

Mathematicians study chaos: 2 starting states that start off almost the same diverge quickly. For example the logistic map $f(x) = 4x(1-x)$ is chaotic. The general goal is to understand iterates.

An **invariant measure** is a measure fixed under f: For all measurable $A \subseteq X$,

$$\mu(A) = \mu(f^{-1}(A)).$$

Can we compute invariant measures?

No. You can embed a universal Turing machine inside a smooth dynamical system.

Given a nice A, e.g., a box $A = [a_1, b_1] \times \cdots \times [a_n, b_n]$, what is $\mu(A)$ to within precision δ? If the Turing machine halts, all the mass of the Turing machine is on one set; if not, it's on the complement.

We expect reasonable systems to be computable. What's present in real life but not here is noise.

Consider a random perturbation of a dynamical system, $x \mapsto f(x) + (\varepsilon\text{-noise})$. Think of the noise as uniform on an ε-ball or Gaussian.

For the results on invariant measures to be "physical", they must be robust. Physical computers are robust. These mathematical objects are not physical. Noncomputable phenomenon do not survive under noise.

Encode a Turing machine as 2 decimal numbers representing what's to the left/right on the tape. If I shake the system, erase all but $\log \varepsilon$ window around the head. The Turing machine dissipates under noise; all the good simulators are high-dimensional.

BGR show that invariant measures of random perturbations are computable. Originally you have infinite memory, it gets erased. By invariant we mean

$$\mathbb{P}_\mu(X_{t+1} \in A) = \mathbb{P}_\mu(X_t \in A).$$

Think of f as acting on measures. The invariant measures are those that are fixed under the operation.

How might you compute it? Divide space into A regions. Assume it's roughly constant on each. The problem transforms to computing the stationary distribution of a A-state Markov chain. This can be done in polynomial time in A. If you want to compute to precision δ, each should have size depending δ; take $A = \text{poly}\left(\frac{1}{\delta}\right)$.

Should depend primarily on ε and not so much on the precision: $O_\varepsilon(\text{poly}\left(\frac{1}{\varepsilon}\right))$.

This makes sense when $\varepsilon \ll \delta$: the precision is much smaller than ε.

With noise the measure is almost constant on each piece: you jump and scatter. The probability that you are close by is the same. There is smoothing. Without noise there is no guarantee. (You can get weird invariant measures, supported on fractals, etc.) The noise smoothens all the bumps.

We'll do something similar but more clever.

Divide space into A regions $a_1, \ldots, a_n$. In each region choose center points $x_1, \ldots, x_n$ arbitrarily. We'll write the density function as a Taylor series around the x_i.

Decompose $\mu_t(x) = \sum_{i=1}^{A} \mathbb{1}_{x \in a_i} \sum_{k=1}^{\infty} p_{i,k}(t)(x - x_i)^k$. (We assume f and noise are nice, i.e., smooth. Gaussian noise smooths things out.) We can express any smooth measure in terms of Taylor coefficients. The nice thing is that the dynamical system acts linearly on these coefficients. Start with arbitrary μ_0, evolve the system to get $\mu_1, \mu_2, \ldots, \mu_t, \ldots$.

There is an explicit linear map $Pp^{(t)} = p^{(t+1)}$. P is infinite-dimensional but we can fix this by truncating at level N. We stopped at level 0 in the picture before.

Let $P_N = P$ truncated to degree $\leq N$.

This is an $AN \times AN$ matrix. We lose something. To get $\left\| P_N^T \mu - \pi \right\| \leq \delta$, set

$$T = O(\ln\left(\frac{1}{\delta}\right) \exp(\varepsilon^{-1})), \quad T = O(\ln\left(\frac{1}{\delta}\right) \operatorname{poly}(\varepsilon^{-1})).$$

We can compute the map in polynomial time. The total time is $O(\ln\left(\frac{1}{\delta}\right) \operatorname{poly}\left(\frac{1}{\varepsilon}\right))$. This gives an efficient natural time bound.

11.1 Space complexity

The argument of why these should be computable is from space complexity: you can't store so much in one step because of noise.

We use a similar technique.

Theorem 11.1: For nice f and Gaussian noise, we can compute the invariant measure in space complexity

$$O(\operatorname{poly}(\ln\left(\frac{1}{\varepsilon}\right) + \ln\ln\left(\frac{1}{\delta}\right))).$$

It's polylog in the size of the input and output.

The model is the typical: There is a read-only input tape, a work tape of the above size, and a write-only output tape. This is tight: any Turing machine with this space complexity, can be embedded in a dynamical system. ε-noise is like distinguishing $\frac{1}{\varepsilon}$ space. You should behave like a memory $\lg \frac{1}{\varepsilon}$ machine. This is a characterization of dynamical systems as a computational device.

The algorithm is essentially no different. We still want to

1. compute P_N,
2. raise P_N^T.

Computing P_N boils down to computing a bunch of integrals. It's relatively easy to do. The second part is trickier.

Now ignore all these dynamical system stuff. It's just a problem of raising matrices to large powers in polylog space. Given an $n \times n$ matrix all of whose entries are integer multiples of $2^{-\operatorname{poly}(n)}$ (i.e., numbers specified to that precision), compute in $O(\operatorname{polylog}(n))$

space M^{2^n}. We want an exponential power because T can be exponential iin $\exp(\varepsilon^{-1})$ or $\exp(\varepsilon^{-2})$. Another way of thinking of this is that I give you a Markov chain

Each entry has poly(n) bits. You cannot even store 1 entry/number in the graph. (Cf. reachability: you can't store the graph. Recall what you can do in log space.

1. In $O(\lg n)$ space you can add n-bit numbers (add bit by bit and remember the carry),
2. multiply n-bit numbers (add n n-bit numbers at once.)
3. divide n-bit numbers (harder, resolved in 90s). Division can be done in TC^0.
4. work with real numbers that are multiples of $2^{-\operatorname{poly}(n)}$,
5. Multiply 2 matrices (follows from multiplying and adding)

When you compose, space adds. (You can use recursion, pretend like you wrote the intermediate result; call the first program instead, you need stack space for both)

In polylog space you can

1. Compute $M^{\operatorname{poly}(n)}$. By iterated squaring you can do this with $\lg(n)$ operations. This takes $O((\lg n)^2)$ space. This is essentially Savitch's Theorem: replace the matrix with the adjacency matrix.
2. Compute $\det(M)$ and the characteristic polynomial of M in time $O((\lg n)^2)$.
3. Find roots of a real/complex polynomial to within precision 2^{-n} in time $O((\lg n)^5)$.

But we need to compute to exponential-sized exponents. Repeated squaring would take time greater than n.

First consider raising a number to an exponential power x^{2^n}. Repeated squaring takes too much space. Rewrite as

$$x^{2^n} = \exp(2^n \ln x)$$

which you can approximate in logspace using Taylor series, to precision $2^{-\operatorname{poly}(n)}$.

If we can diagonalize M then we are happy. If you can diagonalize $M = U^{-1}DU$, then $M^{2^n} = U^{-1}D^{2^n}U$, and you can do this in polylog time. But no one knows how to diagonalize matrices in polylog time.

What you can do is compute eigenvalues, because they are roots of the characteristic polynomial. Compute $\lambda_1, \ldots, \lambda_n$ or M. Then compute $(\lambda_i^{2^n})$. By Lagrange interpolation construct a degree $n-1$ polynomial $p(\lambda_i) = \lambda_i^{2^n}$. In the denominator you get $\lambda_i - \lambda_j$. We need a lower bound on the distance between 2 roots, which we have by 2^{-n} precision. Now compute $p(M) = M^{2^n}$. If there are repeated eigenvalues, perturb M. One of $n+1$ perturbations will succeed. A small amount of perturbation to a univariate polynomial will have distinct roots. $(1-t)M + E$: $\prod(\lambda_i - \lambda_j) = P(M_{ij})$.

12 Formulas resilient to short-circuit errors

(Kalai, Lewleo, Rao, FOCS12)

Theorem 12.1: Every formula F can be efficiently converted to E where E is resilient to $(\frac{1}{10} - \varepsilon)$-fraction of short-circuit noise at any input-to-output path, with $|E| = O_\varepsilon(|F|)$.

What is short-circuit noise?

Consider the von Neumann noise model: There is a BSC channel at the output: the right answer g with probability $1 - \varepsilon$ and $\hat{g}$ with probability $1 - \varepsilon$.

Short circuit noise connects one of the inputs to the output. It can be adaptive: it can look at the input and choose which to output. What's clear is that $g(0, 0) = 0$ and $g(1, 1) = 1$. Short-circuit noise is slightly weaker. In the short-circuit csae we can deal with worst-case error. In the von Neumann model, if the output gate is wrong, the output is wrong.

Consider what happens when there are multiple outputs. You can't beat $(1 - \varepsilon)^m$, while in the short-circuit model you can. Short-circuit errors are actually realistic.

You can have a whole layer of gates that is short-circuited, disconnecting the input and output. We assume the negations are pushed to the bottom.

We prove the theorem in 3 steps.

1. Convert F into an interactive protocol P (Karchmer-Wigderson)
2. Convert P to P' resilient to noise.
3. Conver from P' into formula W (KW).

12.1 Interactive protocol

A formula is a circuit with fan-out 1. Assume we just use AND and OR gates. (Assume negation at the bottom is noiseless.) I.e., just consider monotone formulas.

A protocol: Alice gets input X and Bob gets Y. Alice sends a bit; people react based on what they see. This creates a tree, until you get to the output.

Karchmer-Wigderson says these are equivalent. (They both involve trees!) Alice gets $x \in f^{-1}(0)$ and Bob gets $y \in f^{-1}(1)$. Their task is to find i such that $x_i \neq y_i$.

Look at the formula: every time there is an AND gate, Alice speaks and says which input evaluates to 0. Bob speaks when there is OR, and says which input evaluates to 1.

What happens when there is a short circuit? An AND is short-circuited. In the protocol, no matter what Alice sends, Bob receives 0. Bob believes that they are in the other branch of the tree. Alice is aware that Bob got 0. This is exactly errors with feedback. The conversion goes both ways. Formula with short circuit is equivalent to interactive protocols with feedback. Because the protocol continuation is given by Bob's branch; by definition of the tree representation of the protocol, Alice knows.

If P with feedback is resilient to e errors then $F_{(P)}$ is resilient to e-short circuits.

12.2 Making interactive protocol resilient to noise

EGH15 in ICTS: $\frac{1}{3} - \varepsilon$ noise over 3-ary alphabet, $\frac{1}{5} - \varepsilon$ noise over binary. Communicate $\{0, 1, \leftarrow\}$ (go back). Every time we lose 3 rounds; this leads to $\frac{1}{3}$ noise. If we have a protocol with n rounds, with $n + 3e$ many we can resist e errors:

$$\frac{e}{n+3e} = \frac{\frac{n}{\varepsilon}}{n + \frac{3n}{\varepsilon}} > \frac{1}{3} - \varepsilon.$$

Take $e = \frac{n}{\varepsilon}$.

For binary, $\leftarrow$ needs 2 rounds to communicate. We need $n + 5e$. This is not tight.

We can get $\frac{1}{3} - \varepsilon$ over binary (2015).

For every noise, pay 3 or 5 times. Size is linear in $\frac{1}{\varepsilon}$: $e \sim \frac{n}{\varepsilon}$, $n + \frac{n}{\varepsilon} = O(\left(\frac{1}{\varepsilon}\right) n) = O_\varepsilon(n)$.

Theorem 12.2 (KLR12)**:** For every $\varepsilon > 0$ there is an efficient compiler that takes fan-in 2 F of depth d and generates E_2 or E_3 of depth $4d + 5e$ or $2d + 3e$ respectively resilient to e short-circuit errors, fan-in 3. The 2 is to convert F into and/or's.

What they get is $\frac{1}{10}$, 10 and 6 respectively. The fan-in is the alphabet size. $\frac{1}{10} - \varepsilon \to \frac{1}{5} - \varepsilon$. $\frac{1}{6} - \varepsilon \to \frac{1}{3} - \varepsilon$.

Use the new result (sophisticated) and put it in, should get $\frac{1}{3}$ in fan-in 2.

What happens with circuits, fan-out?

Make the circuit become a formula. Into formula of exponential size. Want noise-resistant of poly size.

What if there is no feedback? Protocol can diverge; how to convert it to a circuit? The motivation is that we have results that are conceptually harder than the feedback for noise-resilient protocols.