Contents

1 Industry talks

1.1 Auctions and market algorithms at Google (Gagan Aggarwal, Google Research)

Key areas of research:

1. Mechanism design: ad selection and pricing, reserve pricing. We care about incentive properties, welfare/revenue, and computational complexity

2. Understanding user and advertiser behavior. We want to model ad-user interactions and predict advertiser response.

3. Ad allocation under constraints. We want to make decisions online with small memory footprint. We do proxy-bidding, ex. compute per-auction bids knowing an advertiser's general values/preferences.

A lot of ad systems uses the GSP auction: rank ads by bid, and the price of an ad is the next lower bid.

VCG is the standard auction in the literature. It ranks ads by bid, and price is "loss in value to others." VCG induces truthful bidding.

GSP and VCG have different bids at equilibrium. Suppose we want to know how advertisers change bids when we change to VCG.

The typical approach is to experiment: try on a subset. Possible designs are:

- Run VCG on random fraction of queries. (This gives diluted results.)
- Offer truthful bidding to a random subset of advertisers. (This is better.)

The hybrid auction problem: how to run an auction where some people think they are participating in GSP, some think they are participating in VCG. Given a set of bidders with different utility functions, design an auction truthful for all bidders simultaneously. Buyer specifies value $v(i, j)$ and maximum price; seller specifies buyer-specific reserve price.

We extend the assignment game for our class of utility functions, show structural results, and design efficient algorithms. For the special case of ads, we design a simpler hybrid auction.

1.1.1 Beyond greedy auctions

GSP is greedy: it looks from top to bottom, filling the next position with the highest bid. This works when ads and positions are homogeneous. This is not true: ads are heterogeneous. Ranking them by bid is not optimal/efficient anymore.

Instead, run the combinatorial VCG auction. But computation of optimal allocation is NP-hard; there are strong inapproximability results. Approximations do not give truthfulness. If you try to get truthfulness, you have to sacrifice approximation ratio.

Questeion: Is there a low-complexity mechanism with good welfare and GSP-like incentive properties, perhaps for a practical special case?

1.1.2 Online ad allocation using LPS

Problem: Allocate inventory to ads under constraints.

Solution: Model problem as LP. Use the dual solution (much fewer variables); use dual optimal to compute primal optimal. If you solve the LP under sample of impressions and use dual variables from that sample, that is a good solution to the primal as well.

1.2 (Yuval Peres, Microsoft Research)

1.2.1 Bandit optimization

We see a sequence of loss functions. For each rount $t = 1 : T$, choose x_t and receive loss $l(x_t)$. We want small regret

$$R_T = \sum_{t=1}^{T} l_t(x_t) - \min_x \sum_{t=1}^{T} l_t(x).$$

An open problem since 2004 was: does there exist a strategy with $R_T \precsim \text{poly}(n)\sqrt{T}$ when l is from a set of convex functions on a convex body in $\mathbb{R}^n$? Bubeck, Eldan, and Lee 2016 showed $R_T \precsim n^{9.5}\sqrt{T}$.

The nonconstructive $\sqrt{T}$ bound of Bubeck, Dekel, Koren, Peres: mimiax theorem reduces to Bayesian case, use Thompson sampling.

1.2.2 Max-cut in weighted graph

Instead of finding max-cut (NP-hard), focus on something easier: local max-cut. A partition defines a cut. A local max-cut means we can't improve the cut by moving a single vertex. This is related to Nash equilibriums (potential games), Hopfield networks.

The FLIP algorithm: start from an initial partition; flip a vertex one by one to increase cut size.

This is interesting because of the contrast between theory and experiment. This algorithm is usually fast, linear-time in practice. In theory, worst-case takes exponential number of steps—vertices may move back and forth. Reaching local optimum is not a problem in practice but is a problem in theory!

We mimic the smoothed analysis of Spielman and Teng. Given deterministic edge weights and initial configuration, add random noise Z independently on each edge.

Question: by adding small noise with bounded density to each edge weight, will FLIP take poly time?

Etscheid, Rőglin 2014 showed quasipolynomial. We improved to be polynomial (Angel, Bubeck, Peres, Wei), $n^{15.1}$. Open: get down to linear.

Look at rounds of length $2n$. Show that show that it's impossible that every move in a round makes very small improvement $< n^{-12.1}$.

1.3 5G NR (Joseph Soriaga, Qualcomm)

I'll talk about standardization for wireless for the next generation, slated for release 2020. NR is "new radio" to emphasize it's not backwards-compatible. What are the requirements, the spectrum we target? Computing is really enabling this new tech.

There are 3 categories we're optimizing for.

1. Enhanced mobile broadband: get data rate much higher. Decrease latency.

2. Mission-critical control. Relax spectral efficiency to increase reliability.

3. Massive Internet of Things: have much more devices on the network. They don't move, so we can't take advantage of wireless improving at a different location.

Spectrum:

1. Low bands below 1GHz: mobile broadband, IoT
2. Mid 1GHz to 6GHz for eMBB, mission critical
3. High, above 24 GHz (mmWave): extreme bandwidth

We want to make sure tech scalable.

Ex. make subcarrier spacing wider for higher bandwidths.

Frame structure: ensure services are addressable in common frame structure. MIMO. Collapse so NAK is 16 times sooner. High data rate now comes with low latency. Spatial multiplexing. Bounce through walls, go through windows. We have to look at channel design, code: multi-edge LDPC codes are more efficient than LTE Turbo codes at higher data rates. They have high efficiency, low complexity, and low latency.

We have to build these things, to be commercially realizable.

1.4 AI platform for business (Michael Karasick, IBM)

I'll talk about AI strategy.

Games provide a laboratory for reasoning. Checkers (Art Samuels 1957), TDGammon, DeepBlue, AlphaGo. The Kolmogorov complexity of the games is small; description of problem is straightforward although state space is enormous.

New: University of Edmonton: 10^{160} state space. Hold'em Poker. It's about search with lots of hidden information.

IBM Watson: Description of problem is in some sense small and large. You're doing probabilistic search on a large natural language corpus. Think of it as probabilistic theorem proving (?).

After Jeopardy, we got involved in health care (cancer treatment). Talk to subject matter experts and make tools they can use.

How sophisticated do these systems need to be in problem level? AI is mostly syntactic, understanding vocab in particular domain. This is far from understanding the semantics.

In 1970's: LINPACK to solve linear algebra problems at scale. Required was deep understanding of math behind complex matrix algorithms.

Now: full stack deep learning to train deep learning networks at scale. Required is deep understanding of math behind training algorithms and performance of hybrid (CPU/GPU) architectures.

Ex. understanding obligations from pdf document. Transform into logical forms that you can prove theorems about. The reason for doing this is so companies don't go to jail. Given analyses of obligations and company policies (internal control documents), check that companies are following the obligations.

Figuring out semantics of document is NLU, decompilation problem. Build a model of documents so we can decompose into chunks. Taking a picture of a line plot and reasoning about the data generating the plot is difficult.

Example: Take a horror movie (Morgan) and create a trailer for it. Decompose into scenes, do PCA for every feature. Every horror movie has important scenes either tender, scary, or suspenseful. We turned something that takes a month to something that takes an hour.

Story: Studio came to us, "we want you to help us design the trailer. We're launching it during Super Bowl, please monitor social media." We thought the trailer was disastrous. Fans made their own trailers. We analyzed social media reaction to those. At the end, they were listening to us, but not before they lost $100+ million dollars, head of studio was fired.

2 Lightning talks

2.1 (Aviad Rubinstein)

In progress work with Schramm and Weinberg.

Secret graph has 5 vertices A, B, C, D, E. Objective is to find minimum cut. You can guess subset of vertices (partition) and I'll tell you how many edges cross the cut.

- For ED/ABC, 3 edges.
- A: 2.
- B: 1.
- CD: 2.

The graph is ACD, EB. You want to minimize number of queries. You can do this in $\binom{n}{2}$ steps: beyond that there is linear dependency.

Conjecture: You need $\widetilde{\Omega}(n^2)$ queries.

Theorem 2.1. *For s-t MinCut we can achieve $\widetilde{O}(n^{\frac{3}{2}})$; for approximate MinCut, $\widetilde{O}(n)$.*

2.2 The high-dimensional geometry of binary neural networks (Alex Anderson)

Joint work with Cory Berg. Supervised by Bruno Olshausen.

`http://alexanderganderson.github.io`

A typical feedforward neural network: apply matrix, nonlinearity,

Binary: binary values at nodes. Binarized function: sign. This executes faster and requires less memory access.

Bourbariaux, Hubara 2016: train with binary weights and activations, augment with continuous latent variables, use SGD.

These networks work because of high-dimensional geometry of binary neural networks. Angle between vector and binarize vector is small. Binarizing approximately preserves direction in high dimension. We validate in BH network. Dot products for original and binarized are highly correlated.

Our theory is a foundation for understanding BNN and low-precision NN. This is foundation for understanding other architectures.

2.3 Sparsification for graphs (Anup Rao)

One important tool for graph algorithms is the ability to solve Laplacian linear equations in almost linear time.

$$L_G = D_G - A_G.$$

We can solve linear equations of the form $L_G x = b$ in time $\widetilde{O}(m)$, m the number of edges. Most algorithms rely on graph sparsification. Spectral sparsifications: given graph G, approximate by sparse H, in certain sense, $L_G \approx_\varepsilon L_H$. Use sparse graph as preconditioner to solve equations.

Every graph has a linear $\widetilde{O}\left(\frac{n}{\varepsilon^2}\right)$ size approximator, constructed in linear $\widetilde{O}\left(\frac{m}{\varepsilon^2}\right)$ time.

This has led to improvements for min cut, max flow, matching algorithms. So far the algorithms are for undirected graphs. The main bottleneck for directed graphs is the right definition of sparsification. Cut sparsifiers don't exist. We defined a notion of sparsification for directed graphs. Diagonal is out-degree matrix, $L_G = D - A$. We recover desired properties of spectral sparsifiers. We can compute PageRank vector, etc. in nearly linear time.

2.4 Proofs of work (Manuel Sabin, UC Berkeley)

Eve sends an email to Bob. Dear Sir, I've recently come to inheritance to Nigerian prince and want to share. Spam!

- for each email, need to solve a puzzle that takes some work.
- DoS
- Recently important in cryptocurrency, Bitcoin.

Have a proof of work that secures the blockchain. Majority vote: to convince people of your timeline, need majority of computing power. Find preimage to hash.

But such work is useless. Bitcoin is a big industry: 5 times computing power of Google. This is an environmental disaster.

We propose proofs of useful work.

Ex. for graph, find shortest path. This is useful but easy.

We want average-case hardness. For random graph, this is hard but useless.

We marry hardness and usefulness.

We use all-pairs shortest path, and use fine-grained complexity.

2.5 ETH hardness for densest k-graphs (Pasin Manurangsi, UC Berkeley)

We want to show hardness of approximation. Typically reduce from NP-hard problem to optimization problem. If 3SAT is satisfiable, optimum is large $OPT \geq c$. If it's not satisfiable, we want $OPT \leq cr$. If reduction is polynomial time, then the 2nd problem is hard to approximate to ratio r. Many problems are known to be hard to approximate, ex. 3SAT. Others are fundamental but we don't know how to prove.

Find densest k-subgraph. There is approximation algorithm $\frac{1}{n^{\frac{1}{4}}}$. We don't even know NP-hardness to approximate to within 0.9.

We can use a stronger assumption that $P \neq NP$. The assumption is ETH: no subexponential time algorithm for 3SAT. This allows us to make the reduction size huge. Before we had to have poly-size reduction. Large reduction but slightly less than 2^n, ex. $2^{n^{0.99}}$ is fine.

We show under ETH, DkS is hard to approximate to within $n^{\frac{1}{\text{poly}\log\log n}}$.

2.6 Non-malleable commitment (Akshayaram Srinivasan, UC Berkeley)

A commitment scheme is protocol between committer and receiver.

1. Commit phase: At the end, receiver doesn't learn any information about message m.
2. Opening phase: Opener cannot open to any meesage beyond m.

Ingredient in ZKP, multi-party proofs.

But if channel is not authenticated, some attacks are not captured by this model.

Consider man-in-model: inject new messages and tamper with communication.

Goal of MiM is to get R to commit to message that is related to original message. To product against these attacks, notion of non-malleable commitment introduced [DDN91].

Only way to get certain multi-party protocols.

Want to understand communication and round complexity.

[GPR16] got 3-round NM $|m|^7$. We show 3 rounds, quasi-optimal $|m| \ln k$, k security parameter. We connect to 2-source non-malleable extractors.

2.7 Learning dynamics of neural networks (Maithra Raghu, Cornell University & Google)

Deep learning is everywhere. We're just at the beginning of principled understanding. Random neural networks, generalization, probability distributions.

One fundamental question: how network evolves to final state. At the end it learns useful representations. Look at network representation to understand how learning happens. For each neuron, consider as vector $[z_1, \ldots, z_n]$, encode input $[x_1, \ldots, x_n]$. We can do this for every layer. Because we can linearly combine, we have a subspace for each layer. We hope to use this to compare representations.

Use CCA. Patterns

1. Network converge bottom up: layers at bottom converge first. Perfectly correlated along diagonal.

2. Blocks of batch normalization layers.

Should we treat layers similarly? Freezing lower layers before end of training doesn't hurt performance. Vary learning rates in layerwise fashion.

2.8 Algorithm design for massive data (Xiaorui Sun)

Massively parallel computing: hadoop, spark are used everywhere. Implementations are different, but they share the same model, mapreduce. Have large set of data, too large to fit in single machine. Each machine processes data on local memory. After local computations, results are aggregated and sent back to cloud. We cannot do much in a single round, need to layer and to in multiple rounds. We want small number of rounds of communication.

What kind of problems can be efficiently solved/not in this framework?

- Divide and conquer: naturally parallelizable.
- Greedy: [Kumar-Moseley-VV13], [Barbosa-Ene-NW16]
- Dynamic programming: wide open. Results depend on results on previous stages.

We propose new principled methods to approximately solve dynamic programming in $O(1)$ rounds. This gives efficient algorithms for longest increasing subsequence, weighted interval scheduling, optimal binary search tree... Key ideas are decomposability and monotonicity.

2.9 (Dylan Foster, Cornell University)

Online learning. Protocol: at time t, get X_t, predict label $\widehat{Y}_t$, and get loss $l(\widehat{Y}_t, Y_t)$. This is in adversarial setting. Measure performance by regret, loss compared to benchmark

$$\sum_{t=1}^{n} l(\widehat{y}_t, y_t) - \inf_{f\in F} \sum_{t=1}^{n} l(f(x_t), y_t).$$

F a hypothesis class. ZigZag is instance-optimal—match on any setting up to constant factors. Ex. iid, adversarial. More generally, interpolate between regimes.

We characterize when this is possible.

Theorem (for linear predictors): If it is possible to achieve instance optimality, then F is a decoupling space: normed vector space, weakly dependent processes (martingales), depend like random processes.

We leverage Burkholder's Theorem. There is a special function, Burkholder/Bellman function with zigzag convexity. We can drop it into our algorithm.

2.10 Logical clustering (Marcell Vazquez-Chanlatte, UC Berkeley)

Motivating example: suppose you have a highway, car, want to change lane. Guarantee that it changes lanes. End up with traces, some follow reference well, some going crazy. Get gigabytes at time, group by how well achieving criteria in unsupervised fashion. Try dynamic time warping, etc., but because traces satisfy logical criterion that doesn't do any good.

Use rich language of temporal logic. For example,

$$((x - x_{ret}) < \alpha \cup \text{step}) \wedge G(\text{stop} \to \diamond_{[0,\infty)}(|x - x_N| < \alpha))$$

Split up, divide and conquer. Approximate Hausdorff distance. Group and project to single point. Find traces. Grow template library.

2.11 Statistical and computation tradeoffs (Ahmed Alaoui, UC Berkeley)

Try to detect signal with much less data than what is possible in computational efficient way.

Often in compressed sensing gap is constant. We're inspired by data collection in genetics. Large pool, analyze pool. Get frequency spectrum of characteristics.

Get histogram of types. We characterize information-theoretic thresholds to reconstruct signal. Let m be number of measurements.

$$m = \frac{\ln d}{d-1}\frac{n}{\ln n}$$

where d is number of types, n is number of people. Computationally efficient threshold:

$$m = \frac{\ln d}{d-1} n \cdot BP$$

There is phase transition. Formula related to spanning tree polynomial.

2.12 Regularized clustering is noise-robust (Shrinu Kushagra, University of Waterloo)

Clustering: group similar points together and separate distant points.

Define cost; find the minimum-cost partition. One way is k-means. Find clustering so square of distances to centers is minimized,

$$\min_{c_1,\ldots,c_k} \sum_{i=1}^{k} \sum_{x} \|x - c_i\|^2 .$$

If data is separated, then minimizing this finds the clusters.

Noise: add 1 point (outlier). It has to be in a cluster of its own; otherwise its cost is too large. All other points become 1 cluster. One point can dramatically change structure.

Make objective noise-robust. Add a garbage cluster

$$+\lambda|c_{k+1}|.$$

Now we can dump the outlier in the garbage cluster. This objective is also NP-hard. We have heuristic based on convex relaxation. If there is nice structure, the heuristic gets the optimal clustering and dumps the garbage points.

2.13 Sample complexity of neural nets (Abbas Mehrabian, University of British Columbia)

Join work with Nick Harvey and Chris Law.

Each node in graph takes a linear combination and applies nonlinear function, send to next node.

Fundamental theorem of machine learning: number of samples for PAC learning within error ε with VC-dimension v is $\Theta\left(\frac{v}{\varepsilon}\right)$.

If activation function is piecewise poly

$$cel \le v(e,l) \le C(el^2 + el\ln e), Ce^2.$$

We improve bound if activation function is piecewise linear

$$ce\ln\left(\frac{e}{l}\right) \le v(e,l) \le Cel\ln e.$$

2.14 Genome assembly problem (Ilan Shomorony, UC Berkeley)

Figure out genome of some organism. Output of sequencer is N reads (subsequences). Assembler puts them together $\widehat{S}$.

Standard method: combinatorial optimization. Define read-overlap graph. Put directed edge with overlap amount. Visit every node once, find path that yields shortest possible sequence. This is NP-hard; people use heuristics.

Alternative approach (Bresler, Bresler, Tse): put model in sequencing. Assume substrings uniformly sampled. Focus on real genome. Divide set of instances (defined by N number of reads, L length) into possible/impossible instances—do the reads have enough information to allow perfect reconstruction?

Even though the original problem is Hamiltonian path, if you start with feasible instance, there is a pruning graph making the graph sparse, solution into a Euler path.

We evaluate HINGE, does better than other algorithms, helps sequence other species.

3 Views from the programs

3.1 Pseudorandomness (Luca Trevisan)

Under standard assumptions (you cannot speed up algorithms by making them longer), there are polytime PRGs with seed length $O(\ln n)$.

Bounded memory: random generator, seed length $O((\ln n)^2)$, universal for space-bounded algorithms. Gets good hash functions for streaming algorithms. Reasonable bounds.

Pseudorandomness for cuts: up to error, number of edges depends up to small error on $|A|$ and $|B|$. Find sparse things with properties of dense things. A generalization of expanders is cut sparsifiers: H is sparsifier of G if for every cut, the number of edges G-edges from A to B is about the same as the number of H-edges from A to B. (Expander is sparsifier of clique.)

Another property of random graphs: Every graph has large clique or independent set. Erdos invented probabilistic method to show there exist large graphs without large cliques or independent sets. There are breakthroughs in explicit constructions, Cohen- Chattopadhyay-Zuckerman 2016 through extractors.

Models for primes: primes have or are conjectured to have properties of random set. Pick N with proability $\frac{1}{\ln N}$. Refinement, get sequence of probabilistic models that are better: $\frac{2}{\ln N}$ if not even, $\frac{3}{\ln N}$ if not even or divisible by 3.

Van der Corput, 1939: infniitely many triples of primes in arithmetic progression. Proof: Fourier transofrm of smoothed version of indicator function of primes.

Green-Tao: infinitely many k-tuple. Proof of Szemeredi's theorem does not distinguish primes from a dense set of integers. Reason about Gowers norm of almost-primes.

2 starting points:

1. Szermeredi's Theorem: if S is set of integers of constant density, A contains arbitrarily long arithmetic progressions.

2. Almost primes pseudorandom in strong sense measured by Gowers norms. Primes have constant density in almost primes.

Dense model: all integers indistinguisheable from almost primes.

Model set has infinitely many k-progressions by Szemeredi's Theorem. Primes have infinitely many k-progressions by indistinguishability.

This is similar to how we reason about pseudorandomness in complexity theory, like a reduction.

3.2 Machine learning (Sanjoy Dasgupta)

Theoretical machine learning is a flourishing field and made high-impact contributions (boosting, online learning and bandits, generalization gives a way to think about sample complexity).

What's not so wonderful: community is working on a small subset of machine learning, where the math framework is well-defined and there are clear problems to make progress.

We want to grow the fields of inquiry within ML.

Three area/workshops: Interactive, representation, computational challenges. We also include talks from non-theoreticians because we're interested in making models when there isn't so much theory.

3.2.1 Interactive learning

Much of success in ML has been in supervised learning. Dataset is obtained, someone labels, human goes away, and a machine is started up and told to find a classifier. We're interested in situations where human and machine collaborate closely in hopes of better outcome. Ex. label points when necessary. Hope this leads to better outcomes and reduced human involvement.

Most work is ad-hoc in one-off systems. The general model is not clear.

Some areas:

- explanations and interpretations. Ex. explain to Alexa why you like the movie. The additional information can be much more informative than just the label.

  Conversely, how to get classifier to explain prediction to us, ex. why reject a loan application?

  Ex. In each step see a picture, make a prediction. Human corrects and points to some part of picture that explains the difference. How to use this to build classifiers more efficiently, that explain their predictions?

- teaching. Human teaching machine. Like supervised learning, but human takes more active role, choosing informative examples. Usual sample complexity results don't apply—how much better can we do?

  Intelligent tutoring, human teaching human, machine teaching machine. We want a formal theory of teaching.

  Theory has basic failings—teacher knows internals of learner, ex. what type of classifier. Model could be complicated/inscrutable, ex. neural nets. Develop more general models that don't require detailed knowledge of what learner looks like. This would be useful in all these different contexts.

- Improving unsupervised learning by interactions. There has been huge progress in unsupervied learning: clustering, embedding, topic modeling. For a long time these were solved by local search algorithms without guarantees; there are counterexamples. Progress: sophisticated linear algebra, using geometry...

  Even if you obtain optimal solution, it may not be the clustering you want, simply because in high-dimensional data there are many different salient structures, and it's not clear which one you actually want; it may not coincide with the one the the cost function picks out. This can't be done with improving the algorithm; you need interactive feedback loop.

  Ex. learning similarity or distance function.

3.2.2 Representation learning

This has been a preoccupation in AI because having good representations facilitates learning. A decade ago, people focused on manifolds. Data lies close to low-dimensional manifold. Find it and project data to manifold. This would make learning magically easier.

Now a lot of representation learning focuses on neural nets. 2 types of questions:

1. how to find this structure in data?
2. How much data is needed?

Ex. generalization behavior of neural nets. Why does learning from training set do well on distribution? If class of models is not too rich, generalization theory explains this. But neural nets work when the number of parameters is more than the number of data points. Generalization is more complex than number of data points! Train using modified SGD where final product is nonvacuous generalization bound. Another model: framework for building generators.

Model to the point where you can generate samples that look like they came from the distribution. Generative adversarial nets is recently popular. Distinguishing tries to tell apart real and generated data. Both are updated at the same time.

Do this is a way that has rigorous guarantees. This has been a point of synergy between the two programs: dense model theorems are helpful in this context. Pseudorandomness has been thinking about how to fool various tests.

3.2.3 Computational challenges

Deployment of optimization algorithms.

Bayesian algorithms has a host of unanswered questions: complexity of sampling problems, mixing rates of MCMC methods, other forms of approximate inference; relation between ML, MAP, and sampling.