Recent Research Directions

1. Social Coordination Tools and Content Markets

Although I toyed with social networks with my students in a brief set of papers published at WOSN, my stint at Yahoo labs gave me a fresh set of problems in the web milieu that have begun to excite me.

First, with colleagues at Yahoo, I have begun to study what we call social coordination tools. Our premise is that while social networks such as Facebook are well established, they only allow people to be aware of their friends doings. They do not allow friends to do things together --- such as go to a movie, play tennis, watch a Giants game, etc.

On the other hand, there are several ad hoc social coordination tools for niche applications. Evite allows a coordinator to invite a specified set of people to an event such as a birthday at a particular time and place. Meetup allows you to specify a place and a topic (such as French Impressionism) and find any relevant meetings in the area that are open to newcomers. Foursquare allows you to specify a person and find their location, allowing you to drop in spontaneously. While seemingly different, Groupon specifies a deal at a place and coordinates strangers to do group buying at a discount. Each of these web sites has its own UI and its own backend database.

Our insight is that we can generalize all these models as queries across five dimensions: people, places, times, events, and social dynamics. Generalization allows a uniform front end and a uniform back end. More importantly, it allows many new applications. For example, it could be used to “Find a set of 3 other people with Tennis Rating at least 5 to play doubles at Cupertino at 7 pm on Fridays” or to a form of Name-your-own price Groupon that marries Priceline to Groupon and reduces the need for salesmen.

Technically, the hardest of these queries becomes what we call an enmeshed query. We can think of the set of queries forming the nodes of a graph. We place an edge from node $R$ to node $S$ if $R$ is compatible with $S$ (for example, two tennis queries with compatible ratings at the same place and time). The query system attempts to carve this graph into cliques such that in each clique, each node satisfies its capacity constraints (for example, 4 for playing tennis doubles). It differs from publish-subscribe systems in that each new query can cause older queries to be answered; it differs from continuous queries that new queries answer old queries not new data; it differs from Entangled queries in that there is a capacity constraint. While it seems difficult, we believe we can scale an implementation to billions of concurrent queries under some feasible assumptions.

Second, I have also been working on content markets for aggregators. Our premise is that aggregators such as Slashdot are not just isolated phenomena but are a strong alternative for browsing the web for topics for which you do not know in advance what you want. Further, we posit there will be a long tail of aggregators catering to niche content. We wrote a WWW 2012 paper on a game-theoretic model of this phenomenon. The logical next question is “what tools can help support such aggregators”. In particular, we are interested in content markets where aggregators can buy original content --- not outright as in Getty Images or in the feeds one can purchase from Reuters ---- but in a “pay as you go along” fashion. Doing so would democratize content selling as much as Google Ads did for ads with
small players affording to pay. The last part of my Stanford RAIN seminar alludes to this idea. What auction and pricing mechanisms can we use to set prices for such a content market?

2. Genomic Processing and Genomic Databases

With Vineet Bafna, student Christos Kozanitis, and others at UCSD, we have been dreaming of a software processing stack for genomics analogous to the layers for any complex software such as the Internet. What are the equivalents of TCP and IP in such a complex system? This seems like an essential question because genomic hardware is getting cheap and software costs will soon dominate. In particular, we are haunted by the notion of a universal genomic database that contains millions of users and their DNA which: 1) researchers can query to find genomic variations common to say cancer patients (discovery), and 2) which doctors can query to find medical treatments that appear to work for other patients genetically similar to their patient (personalized medicine). Both these queries will need to be cheap and interactive (< msec).

Unfortunately, each genome today is around 250 Gbytes. We investigated compression techniques in a RECOMB paper about a system called SlimGene and found we could get a factor of 10 improvement. Unfortunately, that would not be enough to allow interactive access to a vast genomic database: the only way today is to send disks by UPS. A very appealing alternative is to functionally compress each genome using so called variation callers that simply describe each genome as a set of changed bases, and large scale deletions or insertions compared to a human genome reference. This is much smaller and thus allows the dream of interactivity.

Unfortunately, this dream is flawed for several reasons. First, variation calling is more of an art than a science with several callers each with somewhat different results. Variation calling uses probabilistic inference because the set of DNA Reads has errors and misalignments that must be weighed probabilistically to assess the confidence in a call. Further, most biologists do not directly trust the results but want the underlying raw DNA that supports the call. The underlying DNA is in the form of short random fragments called Reads of the donor DNA that have been mapped to the reference. The biologists then form their own opinion about the validity of the call from a visualization of the Reads. However, the Reads that are used as the basis for a variation call such as a SNP (altered base) or deletion (say 10,000 bases deleted) are a very small fraction of the total number of Reads. If one had a way of asking only for the Reads that are used as the basis for a call, then the biologists could get the small subset of Reads behind a variation call (we call this the Evidence) interactively across the network. Further, even for more complex queries (“tell me all the deletions in a given donor genome”), the set of Reads that attest to this call is a small subset. However, different callers vary in the set of Reads they need to make a call. Thus it seems essential to have a language (which we deliberately call GQL to be reminiscent of SQL) that is as “complete” as possible in being able to precisely select any subset of DNA Reads a caller wants while allowing callers to completely change in the future. As in SQL, GQL uses a Select and Project operator but has a new form of Join based on interval intersection and a new compression operator akin to Interval Union. Christos is building efficient indices for GQL.
This suggests a separation of concerns between an Evidence Layer (whose interface or API is GQL) and a Variation Caller (that does the probabilistic inference). Such a separation of concerns naturally allows providing the evidence used behind any variation call. However, it goes much further than that. Once, one has done this, one can move the Inference Layer to the workstation and only run the Evidence Layer in a server accessible across the cloud. This allows different biologists to use different callers and even allows completely bypassing inference to do visualizations of selected subsets of genomes and locations. This separation and a standard API such as GQL would allow an industry to flourish in building a fast Evidence Layer. History teaches us that standardizing such a layer allows people to go to work to build fast and cheap implementations. Perhaps in 10 years, there would be a Genomic Oracle with a vast market. The techniques used in the Evidence Layer are much the same as in standard databases. All the probabilistic calculations are done in a higher layer.

This layering also allows new inference techniques. For example, in discovering the deletions common to 10,000 cancer patients, it may be unnecessary to do sophisticated inference on each patient. Instead, by asking for variations based on some deterministic criterion perhaps the strength of the signal seen across all 10,000 patients can make individual inference unnecessary. This is somewhat analogous to the way Google uses Big Data to make more effective spell checkers. Our paper on “New Directions for Genomics” provides more details and shows how GQL can make queries across subpopulations; this, in turn, may allow new forms of inference that we call group inference.

3. **New Directions in Network Algorithmics?**

After 20 years of doing Algorithmics, is Network Algorithmics spent? In the early 1990s, Algorithmics was born because the speed of fiber was approaching 10 Gbps but the popular abstractions such as TCP and IP were slow, leading to calls to replace them with new protocols such as XTP and MPLS that were more hardware friendly. A host of techniques developed for servers and routers by our group and others in the 1990s made it clear that principled implementations were all that was required. In the early 2000s, I was inspired by the IBM Autonomic Computing Manifesto to look beyond faster and cheaper to make networks more usable. As a first part of this program, we began to work on measurement instructions for routers (see older directions) such as: finding high-bandwidth conversations, counting conversations, logging, measuring bandwidth etc. Both these directions (*fast routers and router analytics*) are described in the page on “Older Research Directions”. But what’s left?

Two recent directions have begun to fascinate me. First, the OpenFlow movement has suggested making routers more flexible by allowing software to reprogram routers by changing table entries that control the processing of flows. However, a closer look shows that OpenFlow allows a *flexible control plane* but the *data plane is fixed*. A packet comes in, is parsed by a fixed parser to get TCP/IP fields which are looked up in 3 fixed tables and then subjected to a fixed set of actions (drop, QoS etc.) as specified by the table entries. Could we make the parser more flexible, the tables more flexible, and the actions more flexible -- all without going to the extreme of network processors (generally slow). We provided initial answers to the parser question with student Christos Kozanitis in the Kangaroo system. With Glen Gibb, Nick McKeown and Mark Horowitz at Stanford, we are looking more deeply at the remaining questions.
A second direction I like is in algorithmics for data center routers. There has been some seminal work already but I was enticed into following a new line of thought recently by Tom Edsall from Cisco. Tom was seeking a more evolutionary approach towards load balancing in fat trees that he could implement in future chips and yet be better than static hash ECMP. With my student Terry Lam and others at Cisco, we did come up with an algorithm that looks like it will be implemented. Some of our results are very suggestive of future work, especially with respect to the need for changes in data center transport protocols.

A third direction I am becoming interested in is providing a toolkit of algorithms that deal with Big Data. Ideas like Bloom Filters have become mainstream; even commercial systems like Data Domain use Bloom Filters for de-duplication. However, there are a number of interesting sketches beyond Bloom Filters and Count-Min that also seem very useful. In SIGCOMM 2011, we described an efficient sketch for Set Difference of large sets. In my Stanford ISL talk (see web page), I suggested that these questions on reconciling data structures could be extended to structures such as sequences and tables, and across a network graph. With colleagues at Microsoft Research, we found we could, for example, generalize the famous Push-Pull algorithm for rumor spreading to disseminating sets. Mike Mitzenmacher and I found we could generalize set difference to sequence differences, allowing perhaps the faster error correction code we know of for large data sets that allows mutation errors. On data sets of millions of words, our Biff code detected 1000s of errors in under a second, much faster than Reed-Solomon codes. What other techniques can speed the transport and analysis of Big Data?

While all of these research directions seem rather different --- ranging from hardware design to the theory of algorithms and from content markets to load balancing --- they share some common biases that have always affected my choice of topics. First, I love interdisciplinary work because I believe it is in the collisions between the approaches of different fields that the most fruitful discoveries lie, especially those involving the confluence of theory and practice; second, I gravitate to research areas that offer a large canvas and problems whose solutions that can change the world --- because there are so many interesting problems in the world and this is as good a pruning principle as any.