Hot Topics in Theory of Computing

Jeff Kinne[*]

March 12, 2009

This article covers material from a presentation to the University of Wisconsin-Madison Computer Science dissertators and gives some highlights of current research in Theory of Computing. I have tried to choose topics/results to highlight that either are of fundamental importance within Theory, are relevant outside of theory, or both. Other researchers would of course choose slightly different results to highlight.

Note: Wikipedia is a good place to look up information on most of these topics. Wikipedia will not go into very much depth and may not include the latest results, but most articles on Theory in Wikipedia are written at a level that can be understood by non-Theorists.

So Theory of Computing considers the efficiency of computation. And we consider this question in a wide variety of computation settings (time, space/memory, communication complexity, parallel computing), and we even consider computation in various "hypothetical/not-real" settings (nondeterminism and alternation, interactive proofs, and maybe quantum computers go under this heading). For these "not real" settings, we are interested in them for their own sake, but they also often end up having "real world" applications as well (see PCPs and hardness of approximation below).

Anyway, we'll just survey a few areas in Theory of Computing...

1 What to do with NP-Complete Problems

Recall the class NP (Nondeterministic Polynomial time) is the class of problems that can be solved efficiently if we are allowed the power to guess. Some examples:

- SAT is the set of satisfiable Boolean formulae, and a nondeterministic algorithm for SAT guesses an assignment and checks that it satisfies the formula.
- 3-SAT is the same as SAT except that the formula is required to be in 3-CNF form (clauses are ORs of 3 variables each and we AND all of those conditions together).
- Vertex Cover are the set of graphs G and integers k so that G has a set of vertices of size at most k that cover all the edges, and a nondeterministic algorithm for VC guesses such a set and checks that it touches all edges.

NP-complete problems are those for which it has been shown that (i) the problem is in NP, and (ii) the problem is at least as hard as every other problem in NP (in the sense that if you could solve this problem efficiently you could solve all of NP efficiently). There are literally thousands

[*]jkinne@cs.wisc.edu

of known NP-complete problems that show up throughout math and science. Note that by "NP hard" we only mean a problem that is at least as hard as every NP problem, while "NP complete" means an NP-hard problem that also is in NP. So if you see "NP hard" in a paper, you do not necessarily know the problem is in NP, but you do know it probably cannot be solved in less than exponential time (assuming P≠NP that is). Some "NP hard" problems of interest include:

- Circuit Minimization: given a boolean function, find the smallest circuit that computes that function. There are also variations of the problem where you can restrict the depth of the circuit or other parameters. WHAT IS COMPLEXITY...
- Other problems of interest to other areas of CS...
- clique because really bad inapproximability result.
- coloring because 2 vs 3 vs 4(planar)
- VLSI hardness results for architecture.

Many of the NP-complete problems are things that we really need to solve, so what can we do?

NP optimization problems are like "functions computable in NP"; for example, for graph G what is the smallest k such that G has a vertex cover of size k? It is typically these optimization problems that we need to solve, and we might be satisfied with an approximate solution.

- For MAX-3-SAT, a randomized approximation algorithm is simply to pick an assignment of the variables at random. For each clause with 3 variables, say $x_1 \vee x_2 \vee x_3$, exactly 7 out of 8 possible assignments to those variables satisfies the clause. In fact, it is not too difficult to see that a random assignment satisfies 7/8 of all clauses in expectation. This can also be derandomized to give a deterministic approximation algorithm that always produces an assignment that satisfies at least 7/8 of the clauses.
- For MIN-VC, consider the following algorithm. Take an edge from the graph, take those two vertices into your vertex cover and remove them from the graph. Then repeat. It is not too difficult to see this produces a vertex cover that is at most twice the size of the minimal vertex cover.

So we see that if we are satisfied with approximate solutions to NP optimization problems, we might be okay. But can we do better than these approximations?

1.1 PCP Theorem and Hardness of Approximation

We know that if P≠NP, we cannot solve NP optimization problems exactly. A beautiful and deep result proved in the 1990's known as The PCP Theorem shows us that for many NP optimization problems *even approximating* the optimal value cannot be done unless P≠NP.

Theorem 1 (PCP Theorem). *NP $\subseteq$ PCP($O(\log n)$, 3).*

By this we mean the following. Any NP problem can be thought of as solved in two phases: (i) you guess a "proof", and then (ii) check that the proof certifies what you want. So for 3-SAT, you guess a satisfying assignment and it is easy to check that this proves the formula is satisfiable. A PCP is a probabilistically checkable proof, and PCP($O(\log n)$, 3) means the following. For phase

(i) you guess/fix a polynomial size proof, and then (ii) using $O(\log n)$ bits of randomness you pick 3 locations in the proof to look at, and then based only on these locations of the proof you are able to check (with high probability) whether the proof certified what you want. In other words, proofs that a 3-SAT formula is satisfiable can be given that only require you to check 3 bits of the proof!

This should seem amazing to you, but what does it have to do with hardness of approximation? An equivalent restatement of the theorem is as follows.

Theorem 2 (PCP Theorem restated)**.** *There exists a polynomial-time function f : 3-CNF $\rightarrow$ 3-CNF, and a constant $0 < \rho < 1$ such that for all $\phi \in$ 3-CNF:*

$$\phi \in SAT \Rightarrow \textit{MAX-SAT}(f(\phi)) = 1; \text{ and}$$
$$\phi \notin SAT \Rightarrow \textit{MAX-SAT}(f(\phi)) \leq \rho,$$

where we use MAX-SAT($f(\phi)$) to indicate the maximum percentage of clauses in $f(\phi)$ which may be simultaneously satisfied.

Notice that this immediately shows that a “hardness of approximation” result for MAX-3-SAT (if we could approximate better than $1 - \rho$ we could run the above theorem and tell between the two cases). In fact, the following holds.

Theorem 3. *Other versions of the PCP Theorem can be used to show that if P≠NP, then MAX-3-SAT cannot be approximated to within $7/8 + \epsilon$ for any constant ϵ.*

This shows that if P≠NP, then the simple approximation algorithm given above for 3-SAT is the best possible!

The PCP Theorem can be used to show hardness of approximation results like this for many other NP optimization problems. So there is much effort going on currently to give tight results like for 3-SAT (give an approximation algorithm, and use the PCP Theorem to show that doing better than that cannot be done unless P=NP).

1.2 Unique Games

A relatively recent development in hardness of approximation is a new method to prove stronger hardness of approximation results but under a weaker assumption than P≠NP. The assumption used in these results is known as “The Unique Games Conjecture”. One particular result is the following.

Theorem 4. *If the unique games conjecture is true, then Min-Vertex-Cover cannot be approximated to within $(2 - \epsilon)$ for any constant $\epsilon > 0$.*

The jury is still out on whether the unique games conjecture holds or not, but it is being widely studied for its wonderful applications to hardness of approximation. An interesting recent result of Prasad Raghavendra “Optimal Algorithms and Inapproximability Results for Every CSP?” gives a general construction that applies to a wide class of problems (although not all problems) and for each gives a tight approximation algorithm for each assuming the unique games conjecture.

1.3 Final Thoughts on Hardness of Approximation

One take away from this is that both approximation algorithms and hardness of approximation results are very active areas of research.

On a philosophical level, the path that led to the PCP Theorem (and thus hardness of approximation results) is very interesting. It comes out of work on interactive proofs from the 1980's and 1990's whose initial motivation had nothing at all to do with NP optimization! So this is a nice example where a line of research ended up having a completely unexpected but very important application. Put another way, "hey don't make fun of me for researching on something that doesn't seem practical – could you have guessed that we would get hardness of approximation results from working on interactive proofs?"

Techniques: I have said what the techniques used for hardness of approximation results are (use some version of the PCP Theorem or unique games conjecture and do "a kind of a reduction" from your problem to those results). For approximation algorithms a very popular technique recently has been linear and semi-definite programming algorithms.

2 Nice Result: undirected connectivity in L

A recent very nice result is by Omer Reingold "Undirected ST-Connectivity in Log-Space". Undirected connectivity has long been a classic example of a problem that can be solved easily with small space using randomness but not so easily without randomness. Given a graph G and vertices s and t, to decide if s is connected to t, the simple randomized algorithm is to take a random walk on the graph (the drunk man's way home, so to speak). Some analysis can be used to show that if s and t are connected, then you will hit t within a polynomial number of steps with high probability.

The result of Reingold builds on a long line of previous work on expander graphs. Expander graphs are graphs that are highly connected but still have small degree. These graphs have found applications in very many areas of Theory, including many other uses for derandomization. The key property for the undirected connectivity problem is that an expander graph (with appropriate parameters) has logarithmic width (any two vertices are at most a logarithmic number of steps from each other). Reingold gives a way to take a graph G and perform some transformations on the graph that: (i) turn the graph into a constant-degree expander, (ii) preserve the connectivity properties of the graph, and (iii) do this all using $O(\log n)$ space. Thus after performing the transformation, we can find if s leads to t by doing a brute force search.

Key take aways here: this settled a long-standing open problem and is a very nice example of a derandomization algorithm, and the proof uses expander graphs – which are very useful in Theory.

3 Famous result: Primes in P

"Primes" is the problem of determining whether a given number is a prime or not. This was for very long a classic example of a problem that can be solved very fast using a randomized algorithm (using e.g the Miller-Rabin primality test) but not with deterministic algorithms. Recently, Manindra Agrawal, Neeraj Kayal, Nitin Saxena in a paper "Primes is in P" gave a deterministic polynomial time algorithm for this problem. This is a very nice result but is not a big surprise to the Theory community: the conjecture is that there are no problems that can be solved more efficiently with randomness than without. Moreover, the fact that Primes is in P does not have a direct effect on the status of the factoring problem (recall that the hardness of problems like factoring is used as the basis of modern cryptography, so it would be very bad for cryptography if a polynomial algorithm were found for these problems).

4 Nice result: IP = PSPACE

Here is another very nice (but older) result. This one is by Adi Shamir in paper titled "IP = PSPACE". IP stands for interactive proofs and means problems that can be efficiently decided by interactive proofs. We won't go into the details of what interactive proofs are. The interesting thing here is how the result is proved. The inclusion IP$\subseteq$PSPACE is trivial (once you have a definition of IP). For the other inclusion, the idea is to take a polynomial space Turing machine and encode the way it computes into a polynomial. The parameters have to be chosen in a right way for things to work out, but you can probably believe that having a polynomial that encodes the computation of a Turing machine could be nice. In particular, it gives a way to give an interactive proof for whether the Turing machine accepts a given input or not. The key property of polynomials that is used is that any two distinct low-degree polynomials agree in few points (so this gives us a method to ask the prover for "the correct" polynomial and be able to check that he/she is not lying).

Key take aways here: this is a nice result, and is an example of a technique that is widely used in Theory – things get much easier when we work with polynomials, so try to encode whatever problem/result you are trying to solve as a polynomial.

5 What to do With Randomness – Extractors

It remains open whether randomized algorithms can do things that deterministic algorithms can't. However there are still algorithms that are much more efficient as randomized algorithms than any known deterministic algorithm. The question is, how could we actually execute a randomized algorithm? The proof that a randomized algorithm works usually assumes perfectly uniform and independent random bits, but no such thing exists in the real world. So if we are going to actually execute randomized algorithms using a flawed source of randomness from the real world (say, the emissions of a radioactive sample or something), we want a general-purpose way to convert flawed random bits into random bits that are very close to being perfect random bits. We call such a general-purpose algorithm an Extractor.

We won't go into the details, but there has been a lot of work over the past 20-30 years on randomness extractors. Within the past 10 years, extractors have been demonstrated with nearly optimal parameters. I don't know if these are used in practice...

One interesting point is that a deterministic single source extractor is not possible. A deterministic single source extractor would be an algorithm E such that if X is a random source with "a certain amount of randomness", then $E(X)$ outputs a string shorter than X but that is very close to being perfect randomness. It turns out this is not possible. However, it is possible to extract randomness from two independent flawed random sources, so then the algorithm looks like $E(X, Y)$.

Key take away: extractors are an active area of research and may have applications in the real world. Also the techniques used to build extractors are varied and include techniques used throughout the rest of Theory – expanders, algebra, linear algebra, ...

6 Connections...

I already mentioned a connection between PCPs and hardness of approximation that may seem surprising. A common theme in Theory has been connections arising between seemingly different areas of research. Here are some recent examples of research areas that have seen such connections: pseudorandom generators and extractors, creating hard functions and error-correcting codes, PCPs and error-correcting codes, communication protocols and circuit lower bounds, hard functions and derandomization, ... Note: a pseudorandom generator is a function that takes as input truly random bits and outputs many more bits that cannot be perfectly random but nonetheless appear random to any computationally bounded adversary.

7 Cryptography

Of course this is an area that has immense practical value and is still a very active research area. One interesting point here is that many aspects of cryptography *require* randomness. Whereas it is conjecture that any decision problem can be solved almost as efficiently without randomness as with, cryptography is an area where we provably need randomness (e.g. picking your private key in a public-key system at random). One interesting area is that of Zero-Knowledge proofs, where for example you would like to prove to your bank that you know your password without actually giving the bank your password (can we really trust anyone these days?).

8 Quantum Computing

This has also become a hot research area, spurred by a few early results (Shor's algorithm to efficiently factor with a quantum computer, and Grover's algorithm to compute OR on n inputs using $\sqrt{n}$ time on a quantum computer). One way this research has gone is something like: "here is something that has been done without quantumness, what happens if we add quantumness". Interesting open questions are whether BQP (efficiently solvable on a quantum computer) is contained in or contains NP.

9 Open Problems

Open problems abound in Theory (and specifically in my research area of Computational Complexity). The grand-daddy of them all (the $1M question) is whether P≠NP, but there are many others:

- Conjecture: P≠NP, and in fact exponential time is required to compute NP-complete languages.
- Conjecture: P=BPP, where BPP stands for problems that can be solved efficiently using randomness.
- Conjecture: Polynomial Hierarchy is infinite.
- Conjecture: L≠P, where L stands for problems that can be solved in $O(\log n)$ space/memory.
- Conjecture: BQP does not contain NP.

- Question: Can the security of cryptography be based on the assumption P$\neq$NP rather than the hardness of non-NP-Complete problems like factoring.
- Question: Is BQP in the Polynomial Hierarchy?
- ...

The sad state of affairs is that for all we know, it could be that NEXP (nondeterministic exponential time) can be computed using constant depth polynomial size circuits that us AND, OR, NOT, and mod[6] gates.

We do know that most techniques that have been used up to this point will not solve most of these problems (see natural proofs, relativization, algebrization). So most of these "big questions" are likely to remain unsolved for the foreseeable future.

10 Miscellaneous

A few other "big results" in Theory over the past 20 years include: Toda's Theorem (PH $\subseteq$ P$^{\#P}$), circuit lower bounds for parity, NL = coNL.

A few other very active research areas include: error correcting codes (focusing on locally encodable/decodable codes).

Techniques used throughout theory include: expanders, algebra/polynomials/arithmetization, Fourier analysis, probability theory, linear algebra, ...

The "big two" conferences in Theory of Computing are STOC (ACM Symposium on Theory Of Computing) and FOCS (IEEE conference on Foundations Of Computer Science) which have been going since the 1960's. ICALP and STACS are the two "general-purpose" Theory conferences in Europe. Other than those, each sub-area of Theory has a conference of its own that is relatively prestigiuos but less-so than STOC and FOCS: RANDOM for randomized algorithms, Complexity for computational complexity, etc. Some people try to get as many FOCS and STOC publications as possible, while others just submit to the conference "that makes sense for their paper" with the next due date. The two most prestigious journals for Theory papers are Journal of the ACM (JACM) and SIAM Journal on Computing (SICOMP).

FYI my research is within Computational Complexity, and in particular on the power of randomenss. For a list of hundreds of complexity classes, for your amusement, see the Complexity Zoo online (this is what happens when you let some theorists go willy nilly defining complexity classes that seem interesting...).