Last revised 4/29/2010

This course assumes CS154, or an equivalent course on automata, computability and computational complexity, as a prerequisite. We will assume that the reader is familiar with the notions of algorithm, running time, and of polynomial time reduction, as well as with basic notions of discrete math and probability. We will occasionally refer to Turing machines.

In this lecture we give an (incomplete) overview of the field of Computational Complexity and of the topics covered by this course.

Computational complexity is the part of theoretical computer science that studies

1. *Impossibility results* (lower bounds) for computations. Eventually, one would like the theory to prove its major conjectured lower bounds, and in particular, to prove

   Conjecture 1 $\mathbf{P} \neq \mathbf{NP}$, *implying that thousands of natural combinatorial problems admit no polynomial time algorithm*

2. *Relations* between the power of different computational resources (time, memory randomness, communications) and the difficulties of different modes of computations (exact vs approximate, worst case versus average-case, etc.). A representative open problem of this type is

   Conjecture 2 $\mathbf{P} = \mathbf{BPP}$, *meaning that every polynomial time randomized algorithm admits a polynomial time deterministic simulation.*

Ultimately, one would like complexity theory to not only answer asymptotic worst-case complexity questions such as the **P** versus **NP** problem, but also address the complexity of specific finite-size instances, on average as well as in worst-case. Ideally, the theory would be eventually able to prove results such as

- The smallest boolean circuit that solves 3SAT on formulas with 300 variables has size more than 2^{70}, or
- The smallest boolean circuit that can factor more than half of the 2000-digit integers, has size more than 2^{80}.

At that stage, the theory will help develop unconditionally secure cryptosystems, it will give us an understanding what makes certain instances harder than other, thus helping develop more efficient algorithms, and it will provide the mathematical language to talk not just about computatations performed by computers, but about the behavior of any discrete system that evolves according to well-defined law. It will be the ideal mathematical tool to reason about the working of the cell, of the brain, natural evolution, economic systems, and so on. It will probably involve some of the most interesting mathematics of the 23rd century and beyond.

For the time being, complexity theorists have had some success in proving lower bounds for restricted models of computations, including models that capture the behavior of general algorithmic approaches. Some of the most interesting, and surprising, results in complexity theory regard *connections* between seemingly unrelated questions, yielding considerable "unification" to the field.

1 Some Examples of Lower Bound Results in Complexity Theory

It remains open to rule out the possibility that every problem in **NP** is solvable by $O(n)$ size circuits on inputs of length n, even though the existence of **NP** problems requiring circuits of size $2^{\Omega(n)}$ is considered plausible. There has been some success, however, in dealing with restricted models of computations. Some examples follow.

1.1 Communication Complexity

In this set-up, several parties each hold a part of the input to a computational problem that they wish to solve together. We ignore the complexity of the computations that the various parties perform, and we focus on how much *communication* they need in order to solve the problem.

In the most basic set-up of this type, our computational problem is to compute a function $f(\cdot,\cdot)$. There are only two parties, holding respectively a string x and a string y (say, of equal length), and they wish to collaboratively compute $f(x,y)$. For many interesting functions $f(\cdot,\cdot)$ tight bounds on the communication complexity can be established. For example, if the parties wish to determine whether $x = y$, then the communication complexity is linear if the parties run deterministic algorithms, but sub-linear if they are allowed to use randomness. If x and y are bit-vector representation of two sets, and the parties wish to determine if the two sets have non-empty intersection, then linear communication is required even if the two parties use randomized algorithms.

A very useful feature of this model is that, in several set-ups in algorithm design

and data structure design, a good solutions implies an efficient protocol for a related communication complexity problems. If one is able to establish a lower bound for the communication complexity problem, then one derives a lower bound for the algorithmic or data structure problem.

For example, algorithms in the *streaming model*, which is a useful model for applications in data bases and networks, are allowed to only make one pass over their input, and the goal is to design algorithms that use a limited amount of memory. Now note that, if we have a streaming algorithm using space complexity $s(n)$ to solve a problem on inputs $x = (x_1, \ldots, x_n)$ of length n, then we also have a communication protocol of communication complexity $s(n)$ for two parties who know, respectively, the first $n/2$ bits of the input and the last $n/2$ bits of the input. The first party simply runs the streaming algorithm on the first $n/2$ parts of the input, then sends the state of the algorithm to the second party, who has now enough information to complete the computation.

A very useful communication complexity set-up is the "number on the forehead" model. Here k parties want to jointly compute $f(x_1, \ldots, x_k)$, where each x_i is n/k bits long; the i-th party in the protocol knows the value of x_j for all $j \neq i$. (The name of the model comes from the image of party i having the value of x_i written on his forehead, so that it is the only value that he cannot see.) Every problem can clearly be solved with communication complexity n/k, but only $n/2^k$ lower bounds are known for explicit functions. Proving a $n/k^{O(1)}$ lower bound for a polynomial time computable functions is a major open problem. Its solution would have several applications, including circuit lower bounds.

1.2 Proof complexity

Proof complexity lower bounds are inspired by the conjecture that $\mathbf{NP} \neq co\mathbf{NP}$. If so, then if we consider the $co\mathbf{NP}$-complete problem of testing if a boolean formula is unsatisfiable, and if we fix any formalism to write down mathematical proofs (provided that the validity of a given proof can be checked in polynomial time), there must be families of unsatisfiable formulas such that their shortest proof of unsatisfiability in the formalism grows more than polynomially with the size of the formula – otherwise we would have an **NP** algorithm for unsatisfiability which simply guesses a polynoial-length proof of unsatisfiability for the given formula and then verifies the validity of the proof.

For general formal languages for mathematical proofs (or "proof systems"), it remains open to construct unsatisfiable boolean formulas whose shortest proofs of unsatisfiability have superpolynomial length, but such "proof complexity" lower bounds are known for several specialized proof system there are known super-polynomial, and even exponential, lower bounds.

Such lower bounds have implications for the performance of SAT-solvers. If an algorithm for SAT is complete (meaning that it always finds a satisfying assignment when given a satisfiable formula), then, when the algorithm outputs no satisfying assignment, the sequence of steps of its computation is a *proof of unsatisfiability* of the given instance. If one can model such a proof within a proof system for which there are known lower bounds, then such lower bounds apply to the running time of the SAT solver as well.

Backtracking-based solvers such as DPLL are complete and, when they fail to output a satisfying assignment after t steps, one can construct a *tree-like resolution* proof of unsatisfiability of size about t for the given formula. Families of unsatisfiability formulas are known whose shortest resolution proofs of unsatisfiability are of exponential length, and so DPLL type algorithms must take exponential time on such instances.

1.3 Integrality gaps

A common approach to find approximations to **NP**-hard combinatorial optimization problems is to relax the problem to a convex optimization problem in which the set of feasible solutions, instead of being the set of binary strings satisfying a certain condition, is a larger convex set. Under general conditions, optimizing over a convex set of feasible solutions can be done in polynomial time, and one can often derive an approximate solution to the original combinatorial problem by using the optimal solution of the convex relaxation.

The *integrality gap* of a convex relaxation is the worst-case ratio (over all instances) between the optimum of the combinatorial problem and the optimum of the convex relaxation. For relaxations whose integrality gap is very far from one (very small for maximization problems, or very large for minimization problems), we can conclude that they are not useful to derive approximation algorithms.

Recent results of this type rule out very general classes of relaxations, and they apply to infinite families of relaxations which add sub-exponentially many auxiliary variables and constraints.

1.4 Restricted circuits

Ideally, we would like to show that 3SAT cannot be solved by circuits of size $2^{o(n)}$ on inputs with n variables, but this is completely out of reach for now; even "just" proving that 3SAT cannot be solved by circuits of polynomial size would imply $\mathbf{P} \neq \mathbf{NP}$. Indeed, currently, it is an open question to even prove that 3SAT, or any other problem in **NP** cannot be solved by circuits of size $6n$ on inputs of length n.

There has been some success in proving lower bounds against special types of circuits. Unlike the models mentioned in the previous sections, such circuits do not model

realistic algorithms, but such lower bounds have interesting applications as well.

Monotone circuits. A boolean function $f(\cdot)$ is *monotone* if changing a zero to a one in the input cannot change the output from a one to a zero. It is easy to see that a monotone boolean function can always be computed by a circuit consisting only of AND gates and of OR gates (without NOT gates), and that every function computed by a circuit of this type is monotone. Hence, circuits made only of AND gates and OR gates are called *monotone* circuits. Razborov proved in the 1980s that $CLIQUE_{\sqrt{n}}$, the problem of deciding if a given n-vertex graph has a clique of size at least $\sqrt{n}$, cannot be solved by monotone circuits of polynomial size. Note that, if we represent graphs as adjacency matrices, then, as a boolean function of the input matrix, $CLIQUE_{\sqrt{n}}$ is a monotone function. It was conjectured that every polynomial time computable monotone function can also be computed by a monotone circuit of polynomial size and, if so, it would follow that $CLIQUE_{\sqrt{n}}$ is not solvable in polynomial time and hence $\mathbf{P} \neq \mathbf{NP}$. Unfortunately, it was soon proved that checking if a given graph has a perfect matching (a monotone function computable in polynomial time) cannot be done with polynomial size monotone circuits, and so the conjecture is false.

AC0. $AC0$ is the class of decision problems solvable by polynomial size circuits that have NOT gates, AND and OR gates of unlimited fan-in, and have only constant depth (independent of the input length). This class captures "constant time on a parallel computer with a polynomial number of processors" and contains a few non-trivial boolean functions. It is known that PARITY, the problem of checking if the number of ones in the given input is odd, cannot be computed in AC0.

Modular gates. To see how robust is the AC0 lower bound, it is natural to "hard-wire" into AC0 circuits the ability to compute parity, and see if it is still possible to prove a lower bound. Indeed, one of the proofs that PARITY is not in AC0 can be adapted to show that MOD3 (checking if the number of ones in the given input is a multiple of 3) cannot be computed by a constant depth, polynomial size, family of circuits with NOT gates, and AND, OR, and PARITY gates of unlimited fan-in. The latter class is called $ACC0[2]$, with the two in square brackets standing for the MOD2 gates which are allowed in the model. Similarly, one can define $ACC0[m]$ as the class of problems solvable by polynomial size, constant depth, families of circuits with NOT gates and unlimited-fanin AND, OR and MODm gates. It is possible to show that if p and q are distinct primes then MODq is not computable in $ACC0[p]$. It is open, however, whether there is a problem in **NP** not solvable by linear size $ACC0[6]$ circuits. (Or by linear size $ACC0[m]$ circuits, where m is any composite with two distinct factors.)

2 Some Examples of "Connections" and "Unification" in Complexity Theory

So far, unconditional lower bounds have been proved only against restricted classes of algorithms (or for problems of very high complexity. Most of the work of contemporary complexity theory is on *connections* about questions. For example:

- *NP-completeness* is a prototypical example. We know thousands of **NP**-complete problems, and although we do not know their complexity yet, we know that understanding the complexity of any one of them will imply understanding the complexity of all of them. If we consider, for every **NP**-complete problem L, the question "does L have a polynomial time algorithm?" then we don't know the answer to all those thousands of questions, but we know that *all the answers are the same.*

- *One-way functions.* A function $f()$ is one way if computing $f(x)$ given x is easy, but finding an x' such that $f(x') = y$ is hard given $y = f(x)$. (The difficulty has to hold on average, for a random x.) If one-way functions do not exist, then no cryptographic problem is solvable, except those that have simple information-theoretic solutions. For example one can encrypt one message no longer than a shared secret key using one-time pad, but it is not possible to encrypt messages longer than the shared key. If one-way functions exist, however, then all problems in private-key cryptography, as well as the problem of creating digital signatures, have solutions with extremely high security guarantees. This means that even though we don't know whether there provably exist secure signature schemes, secure authentication schemes, secure encryption schemes and so on, we know that the questions of whether such protocols exist are all the same, and are all equivalent to the question of whether one-way functions exist.

- *Probabilistically Checkable Proofs* (PCPs) provide a characterization of **NP** which is a convenient starting point to prove *hardness of approximation* of combinatorial optimization problems. Via the *PCP theorem* and various reductions, we know that for several optimization problems it is as hard to improve the performance of known polynomial time approximation algorithms as it is to solve the problem optimally in polynomial time.

- *Derandomization* is the task of reducing the amount of randomness used by randomized algorithms. Ideally, one would like to show that every randomized algorithm can be simulated with *no use of randomness* whatsoever, in a purely deterministic way. It is known that if there is a problem in solvable in time $2^{O(n)}$ and having circuit complexity $2^{\Omega(n)}$ then such polynomial time deterministic simulations of randomized algorithms are possible. Note that this

a connection of a different nature than the ones described above. Previously, we discussed results showing that if a certain computational problem (solving a problem in **NP**, inverting a one-way function, optimally solving an optimization problem) is hard, then other computational problems are also hard (respectively, solving any **NP**-complete problem, breaking certain encryption, authentication and signature schemes, approximately solving certain optimization problems). Derandomization results, however, turn a hardness assumption into an *algorithm*.

- *Worst-case versus average-case.* The result on derandomization mentioned above is the combination of two main results: (i) one can simulate deterministically with polynomial slowdown all randomized algorithms, provided that there is a problem solvable in time $2^{O(n)}$ that is hard *on average* for circuits of sub-exponential size; (ii) if there is a problem solvable in time $2^{O(n)}$ that is hard *in the worst case* for circuits of sub-exponential size, then there is also a problem solvable in time $2^{O(n)}$ that is hard *on average* for circuits of sub-exponential size. The second theorem is part of a more general theory showing that for certain problems and complexity classes one can turn worst-case hardness assumptions into seemingly stronger (but in fact equivalent) average-case hardness assumptions.

3 The Plan for this Course

3.1 The Basics

We will start by looking at the basic models in complexity theory, and consider deterministic, non-deterministic, randomized, non-uniform and memory-bounded algorithms, and the known relations between their power.

3.2 Reingold's algorithm

We will then give an overview of Reingold's algorithm, which proves that undirected connectivity is solvable with $O(\log n)$ bits of memory deterministically.

3.3 PCP and Inapproximability

We will continue with Probabilistically Checkable Proofs, the model that allows to prove NP-hardness results for approximability, and sketch Dinur's proof of the PCP Theorem, the main result in this theory.

3.4 Parity not in AC0

We will then see the proof that computing the parity of the number of ones in an n-bit input cannot be computed by polynomial size, constant-depth, circuits made of AND, OR and NOT gates. The proof that we will see has a relatively easy extension to show that there is no polynomial size constant depth circuit made of AND, OR, NOT, and MODp gates that checks whether the number of ones in the given input is a multiple of q, where p, q are distinct primes. It remains open to prove lower bounds for the variation of this model in which MODm gates are allowed, where m is a composite with distinct prime factors. (E.g. $m = 6$.)

3.5 Derandomization, Pseudorandomness, and Average-case Complexity

We define pseudorandom generators, see how one can obtain derandomization results from the existence of strong pseudorandom generators, state the results of Impagliazzo, Nisan and Wigderson on derandomization based on worst-case and on average-case complexity assumptions, and look at the GGM approach to increase the stretch of pseudorandom generators.

3.6 Natural Proofs

Razborov and Rudich have identified a bottleneck that applies to all known techniques to prove circuit lower bounds in restricted model. All known techniques allow us to efficiently distinguish the truth table of a function of low circuit complexity in the model from a random truth table. If strong pseudorandom generators exist, however, it is possible to generate truth-tables of functions that are computable by small circuits but that cannot be efficiently distinguished from random truth-tables.

3.7 Quantum Complexity Theory

We will describe the computational model of quantum computers, and describe one of the two famous quantum algorithms, an algorithm that is able to search over a space of size 2^n in time $2^{n/2}$. Time permitting, we will also show that, with no further assumption on the search space, time $2^{n/2}$ is best possible.