CS 579: Computational Complexity. Lecture 0

Introduction

Alexandra Kolla

Welcome to CS 579!

Administrative stuff

- Prerequisites: CS 374/473 or equivalent, familiarity with the notion of algorithm, running time, reduction, turing machine, basic notions of discrete math and probability.
- Course Website: https://courses.engr.illinois.edu/ece579/
- Recommended reading: Arora-Barak "Computational Complexity: A Modern Approach".
- Office hours: TBD, or by appointment.
- Homeworks: 3 homework sets(70%), final/final project(30%).

Today

- What is Computational Complexity all about and why do we care?
- Some examples of problems Complexity is interested in.
- Theoretical Applications ☺
- Course plan.

What is Complexity Theory?

- It is to computer science what theoretical physics is to electronics.

- Problems to be solved, algorithms to solve them: how much resources do they need? (time, storage space, randomness)

An (incomplete) overview

- Computational Complexity studies
  - Impossibility results (lower bounds). Eventually would like to prove the major conjectured lower bound P≠ NP, which would imply that thousands of natural combinatorial problems don't admit efficient algorithms.
  - Relations between the power of different computational resources (time, memory, randomness, communication) and the difficulties of different modes of computation (exact vs. approximate, worst case vs. average case…). Eg. would like to prove the conjecture P=BPP.

An (incomplete) overview

- Ultimately, we would like complexity theory to not only answer asymptotic worst-case questions (like P vs. NP) but also address the average and worst-case complexity of finite-sized instances, e.g.
  - The smallest boolean circuit that solves 3SAT on formulas with 300 variables has size more than 2^{70}
  - The smallest boolean circuit that can factor more than half of the 2000-digit integers, has size more than 2^{80}

If all of that happens…

- Develop unconditionally secure cryptosystems
- Understand what makes certain instances harder than others, develop more efficient algorithms
- Provide the mathematical language to talk about not only computations performed by computers, but also the behavior of discrete systems that evolve according to well-defined laws. (working of the cell, the brain, natural evolution, economic systems…)

Till then…

- Complexity theorists have had some success in proving lower bounds for restricted models of computation
- Some of the most interesting results in complexity theory regard connections between seemingly unrelated questions, yielding "unification" of the field
- We will see some of that next

Some examples of lower bounds

- 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 NP problems requiring circuits of exponential size plausible.
- Some success in dealing with restricted models of computation, we will see examples next.

Communication complexity

- Several parties each hold a part of the input to a computational problem and they wish to solve the problem together
- We ignore the complexity of the computations that they perform and focus on how much communication they need to solve the problem
- Most basic set up of this type, there are only two parties holding respectively n-bit stings x and y
- Problem is to collaboratively compute a function

$$f(x, y): \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}$$

Communication complexity

- Trivial protocol requires n+1 bits of communication
- For many interesting functions f(.,.) tight bounds on the communication complexity can be established.

- For example, if the parties wish to determine if x=y then
  - Deterministic communication complexity is linear (n bits)
  - Randomized communication complexity is sub-linear (O(logn) bits)

- If x,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 (deterministic, randomized)

Communication complexity

- Very useful feature of this model: in several setups in algorithm design and data structure design, a good solution implies an efficient protocol for related communication complexity problem.
- Thus, if we have lower bound for communication complexity problem then we can derive lower bound for the related algorithmic or data structure problem
- Example: streaming model (useful in data bases and networks). Allowed to only make one pass over the input, goal is to design algorithms that use limited memory.

Communication complexity

- Streaming algorithm on n-bit input ($x_1,\ldots,x_n$) with space complexity s(n)

- Communication protocol of communication complexity s(n) for two parties, where each party knows the first and last n/2 bit of the input resp.

First party runs streaming algorithm on first n/2 bits and sends the state of algorithm to the second party, who completes computation

Communication complexity

- Another useful communication complexity set-up is "number on the forehead" model.

- 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 knows the value of x_j for all $j \neq i$.
- Clearly, every problem can be solved with communication complexity n/k, but only $n/2^k$ lower bounds are known for explicit functions.
- Proving $n/k^{O(1)}$ lower bound for poly-time computable function is major open problem and its solution would have several applications, including circuit lower bounds.

Proof complexity

- Proof system = formal language of mathematical proofs
- Proof complexity lower bounds inspired by the conjecture NP≠co-NP.

Proof complexity

- If conjecture true, then the co-NP complete problem of testing if a boolean formula is unsat, has families of instances for which the shortest proof of unsatisfiability grows more than polynomially with the size of the formula (fixing any formalism to write down proofs).

- Otherwise, we would have NP algorithm that guesses a poly length proof of unsatisfiability and then verifies its validity.

Proof complexity

- For general proof systems it remains open to construct unsat. boolean formulas whose shortest proofs of unsat. have super-poly length

- "proof complexity" lower bounds are known for specialized proof systems (super-poly, exponential lower bounds)

Proof complexity

- Such "proof complexity" lower bounds have implications for the performance of SAT-solvers.
- If algorithm for SAT is complete (always finds assignment if it exists), then when it outputs no assignment, the sequence of steps of computations is a proof of unsatisfiability of the instance.
- If we can model this proof within a proof system with known lower bounds (e.g. resolution) then those lower bounds apply to the running time of SAT solver.

Proof complexity

- Backtracking based solvers (DPLL)are complete.
- When they fail to output a sat. assignment after t steps, can construct a tree-like resolution proof of unsatisfiability of instance of size ~t.
- Families of unsat formulas are known to have shortest resolution proofs of exponential size, thus DPLL algorithms may take exponential time on such instances.

Integrality gaps

- Common approach to find approximation algorithms is to relax an NP-hard combinatorial optimization problem to a convex optimization problem, where set of feasible solutions is a larger convex set.
- Interality gap of a convex relaxation is worst-case ratio over all instances between the opt of the combinatorial problem and the opt of the convex relaxation.
- If integrality gap large then convex relaxation not useful to derive good approx algorithms.
- Recent results rule out very general classes of relaxations.

Restricted circuits

- Would like to show that 3SAT cannot be solved by circuits of size $2^{o(n)}$ on inputs of size n, but this is wishful thinking.
- Even proving that 3SAT cannot be solved by circuits of poly size would imply P≠NP.
- Some success in proving lower bounds against special types of circuits.
- Unlike the models mentioned earlier (communication complexity, proof complexity) those circuits don't model realistic algorithms but such lower bounds have interesting applications.

Monotone circuits

- A boolean function f(.) is monotone if changing a zero to a one in the input cannot change the output from a one to a zero.
- Easy to see that every monotone boolean function can always be computed by a circuit with AND and OR gates only, and that every function computed but such circuit is monotone (ex).
- Circuits made of AND and OR gates only are called monotone.

Monotone circuits

- Razborov (1980s) showed that the problem $\text{CLIQUE}_{\sqrt{n}}$ of deciding if a given n-vertex graph has a clique of size at least $\sqrt{n}$ cannot be solved by monotone circuits of poly size.
- Note that if we represent graphs as adjacency matrices, $\text{CLIQUE}_{\sqrt{n}}$ is monotone function.

Monotone circuits

- Was conjectured that every poly time computable monotone function can also be computed by monotone circuit of poly size.
- If so, it would follow that $CLIQUE_{\sqrt{n}}$ is not poly time solvable and hence P≠NP.
- It was soon proved that checking if a given graph has perfect matching (poly time computable monotone function) cannot be computed with poly size monotone circuits, so conjecture is false.

AC0

- Class of decision problems solvable by poly size circuits with NOT gates, AND and OR gates of unlimited fan-in and constant depth.
- Captures constant time on a parallel computer with poly number of processors.
- Contains few non trivial boolean functions.
- Known that PARITY (checking if the number or ones in a given input is odd) is not in AC0.

Modular gates

- To see how robust is the AC0 lower bound, hardwire into AC0 circuits the ability to compute PARITY.
- Same proof can be adapted to show that MOD3 (checking if number of ones is multiple of 3) is not in ACC0[2] (constant depth, poly size circuits with NOT gates and AND, OR, PARITY gates of unlimited fan-in).
- Similarly define ACC0[m].
- Possible to show that if p,q distinct primes then MOD[q] is not in ACC0[p].
- Open whether there is a problem in NP not in ACC0[m], where m is composite of two distinct primes.

Connections and Unifications

- Unconditional lower bounds have only been proven against restricted classes of algorithms or problems of very high complexity
- Most work in complexity theory is about connections between questions

Connections and Unifications

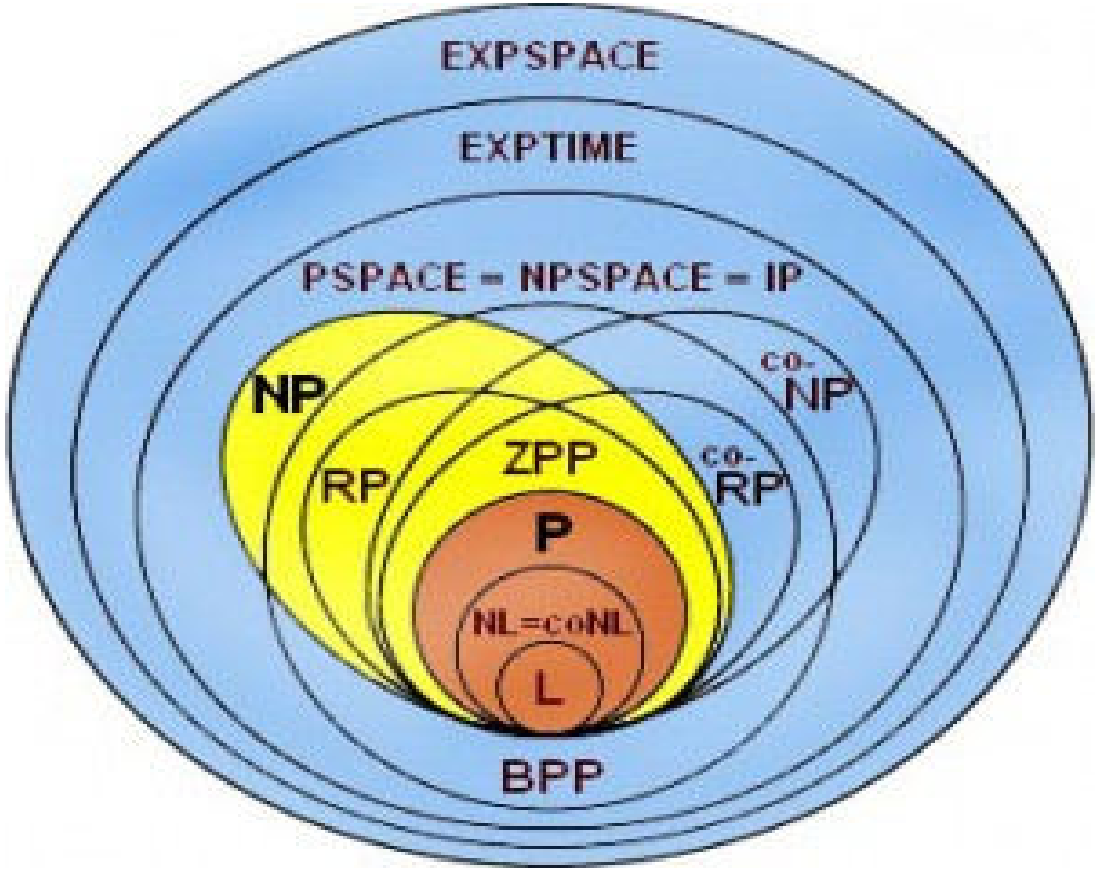
Examples are:

- NP-completeness: We don't know the complexity of NP complete problems, but we know it is the same for all.
- One-way functions: If they exist, then also secure signature schemes, secure authentication schemes, secure encryption schemes exist.

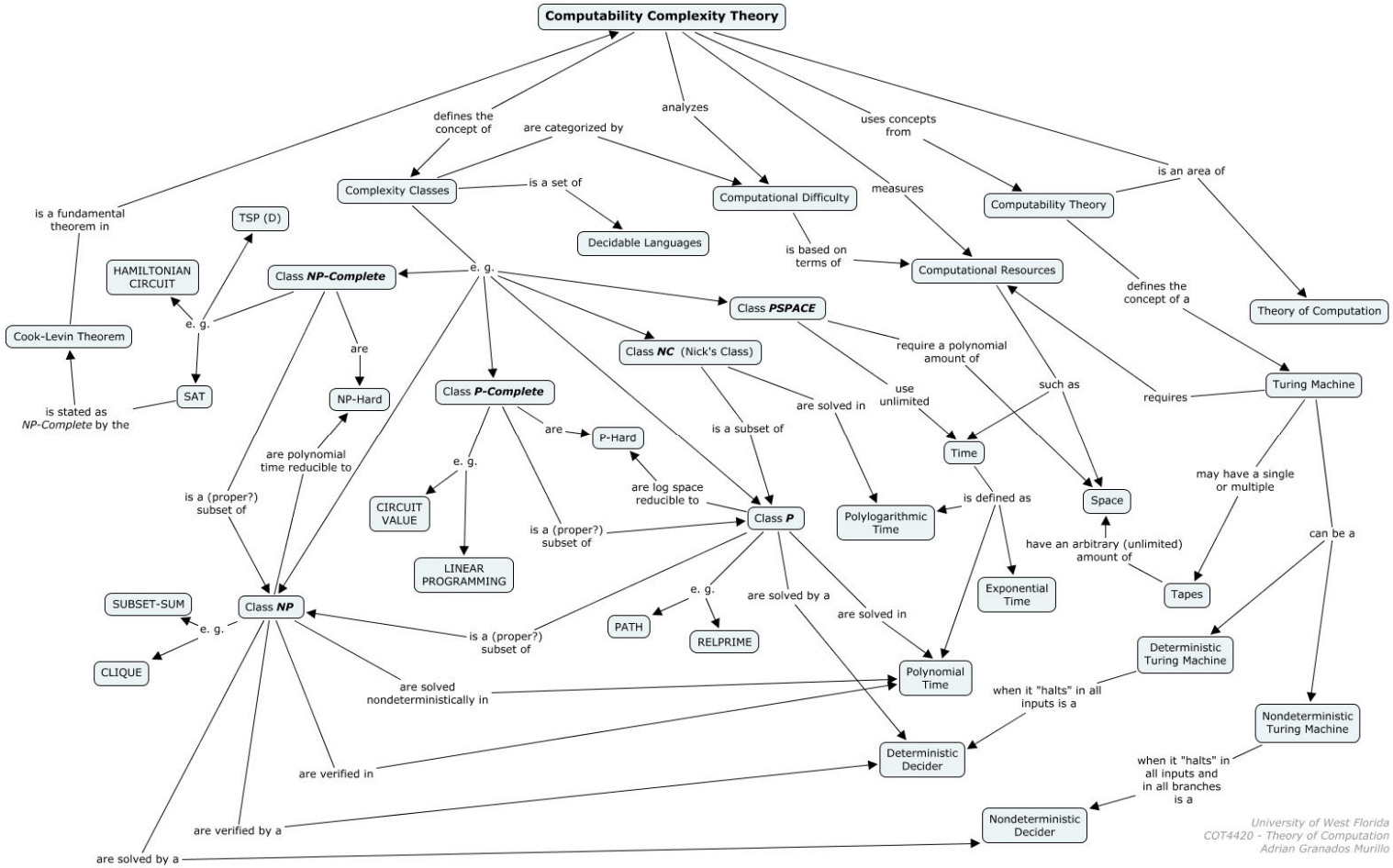
Connections and Unifications

- Probabilistically checkable proofs: characterization of NP that helps prove hardness of approximation.
- Derandomization: ideally would like to show that every randomized algorithm can be simulated deterministically. Can turn hardness assumption into algorithm.
- Worst case vs. average case: for certain problems can turn worst-case hardness into seemingly stronger (but in fact equivalent) average-case hardness.

Complexity Classes



Complexity Zoo!



Course plan

- The basics: look at the models in complexity theory, consider deterministic, non-deterministic, randomized, non-uniform and memory-bounded algorithms and the known relations between them. (about 4 weeks)
- Interactive proofs, IP=PSCPACE. (about 2 weeks)
- Expanders, Reingold's algorithm. (about 2 weeks).
- Derandomization, Pseudorandomness and Average-Case Complexity. (about 2 weeks)
- Unique Games Conjecture, PCP theorem… (tentative).
- Quantum Complexity Theory. (1-2 weeks)