Intractability and NP Completeness

Lecture 16
Prof. Ronitt Rubinfeld
Efficiently solvable problems

• So far in 6046: *problems which can be solved efficiently*
  – Matrix Multiplication
  – Is N a prime or composite?
  – Minimal Spanning Tree
  – Perfect Matching in a graph
  – Feasible Linear Program
Warning: entering the depressing part of the lecture
Can all computational problems be solved?

No!!!

e.g., Halting Problem: Given a program P written in Java and an input x, does P(x) terminate?

Undecidable (Take 6.045 or 6.840 for more)
Universe of computational problems

All problems

solvable

unsolvable

All problems
What do we do now?

Of those that are solvable, can we solve them efficiently?

No! Some require exponential time
Some require doubly exponential time
Some require triply exponential time

... and some are “tractable” (polynomial time)
Universe of problems (not to scale)

Needs more than doubly exponential time

Doubly-exponential time solvable

Exponential time solvable

Poly-time solvable

unsolvable

All problems
Tractable problems

• Polynomial time algorithm:
  • Runtime on inputs of size $n$ is $O(n^k)$

• Polynomial time problem:
  • There is a poly time algorithm $A$ which solves the problem on all inputs

• Comments:
  • $k$ is fixed constant --- can be 1,2,3,..., 101,... but $\textit{not}$ log $n$
  • also called “tractable”, “feasible”, “efficiently solvable”,...
Classify problems

- **Desiderata.** Classify problems according to how fast you can solve
  - Cannot solve
    - Undecidable: e.g., Halting
  - Can solve, but provably requires exponential-time.
    - Given a Turing machine, does it halt in at most $k$ steps?
    - Given a board position in an $n$-by-$n$ generalization of chess, can black guarantee a win?
  - Can solve efficiently

- **Frustrating news.** Huge number of fundamental problems have defied classification for decades.
Easy or hard?

- **Next two lectures:** Show a large class of fundamental problems are "computationally equivalent"; if prove that one is intractable, then all are intractable.
But first a detour…
three types of questions
Decision problems

• Does an object of a certain quality exist?
  • YES/NO type question
  • E.g., Given graph $G=(V,E)$, weight function $w$ on edges, and bound $k$, is there a spanning tree of $G$ of total weight less than $K$?
Search problems

• Find an object of a certain quality.
  • E.g., : Given a weighted graph $G=(V,E)$, a weight function $w$, and a bound $K$, find a spanning tree of $G$ of total weight less than $K$. 
Optimization problems

- Find an object of the best quality.
  - E.g., Given a weighted graph $G=(V,E)$, and weight function $w$, find the minimum weight spanning tree of $G$. 

Decision, Search, Optimization Problems

- **Spanning Tree?** Given a weighted graph $G=(V,E)$, a weight function $w$, and a bound $k$, is there a spanning tree of $G$ of total weight less than $K$?... a decision problem
- **Find Spanning Tree:** Given a weighted graph $G=(V,E)$, a weight function $w$, and a bound $K$, find a spanning tree of $G$ of total weight less than $K$... a search problem
- **Min Spanning Tree:** Given a weighted graph $G=(V,E)$, and weight function $w$, find the minimum weight spanning tree of $G$... an optimization problem

Clearly ?: Optimization harder than Search harder than Decision
Why decision problems?

- Have nice properties for complexity theory
  - If show decision version is hard, then automatically get hardness of search & optimization
  - Not really weaker than search/optimization (e.g., see homework)
  - Convenient to work with

- Formally: a decision problem is a function $D$ from the set of all binary strings (all possible inputs represented in binary) to the set of $\{1,0\}$.
  
  We call an input $x$ to $D$, a **yes-input** if $D(x)=1$
  and a **no-input** if $D(x)=0$

**Notation:** we write $x \in D$ if and only if $D(x)=1$. 
Polynomial Time

- \( P = \{ \text{all decision problems } D \text{ such that there exists a polynomial time algorithm } A \text{ such that } A(x) = D(x) \} \)

- We say that \( A \) decides \( D \)

- Examples:
  - MST?
  - Feasible Linear Program?
What isn’t in P?
Not known to be in \( P \):
Travelling Salesman Problem (TSP)

- **TSP**: Given a weighted graph \( G \) & integer \( K \), is there a closed path that visits all nodes once whose total weight <\( K \)?
- **Search version**: find the closed path
- **Optimization version**: Find the closed path of min weight.
- **Best known algorithm**: finds path in time \( O(2^V) \) for a graph with \( V \) vertices.
  - Given a path, it is easy to verify that its weight is <\( k \)
  - Given a path, can’t verify its weight optimality in polynomial time
- Contrast with MST and shortest path
SAT: Given Boolean formula $\Phi$

- $n$ boolean variables $x_1, \ldots, x_n$,
- ANDs, ORs, NOTs
- $m$ clauses
- Is there a setting of the variables for which $\Phi$ evaluates to 1 (TRUE)?

- **Search version:** find the setting
- **Best known algorithm:** try all assignments
Not known to be in P: cSAT

**Circuit-Satisfiability (cSAT):** Given a Boolean circuit C made of gates and wires (an acyclic digraph whose vertices can be AND, OR or NOT gates) with n-inputs and 1 output, is there a way to assign 0-1 values to the inputs so that the output value is 1 (true)?

**Search Version:**
Find a 0-1 assignment to the inputs that makes the circuit evaluate to 1

**Best Known algorithm:** Try out all $2^n$ assignments

**Notation:** given an assignment $x_1,...,x_n$

$C(x_1,...,x_n)$ is the value of the output gate
What do we do when we need to solve them?

• Spend more time working on it?
  • People have tried for decades with no luck

• Complain?
  • Looks bad …

• Prove there is no efficient algorithm?
  • Difficult
  • No superlinear lower bounds on unrestricted computational models
What we do:

- Show these problems are “equivalent”
  - If can solve one then can solve all the others
  - So, I may not know how to solve it but no one else does either
  - Approach used for thousands of problems
Left to do…

- Identify the class of problems of interest
- Define a notion of equivalence
- Prove the equivalences
What is common to these problems:

• All “Yes”-instances have a short and efficient-to-check proof of correctness (certificate)
  • E.g., solution to search problem
  • May not be easy to find
  • Example: TSP -- Given a weighted graph G & integer K, is there a closed path that visits all nodes once whose total weight < K?
    • If yes, then to prove it just need to give the closed path as a proof
      • Since each edge on the path is present, verifying that it is
NP: Decision problems D for which there exist polynomial time algorithm V & \( c > 0 \)

- if \( D(x) = 1 \) (x is a yes-input of D), then \( \exists \ y, |y| < |x|^c \), \( V(x, y) = True \)
- if \( D(x) = 0 \) (x is a no-input of D), then \( \forall y, V(x, y) = False \)

\( V \): verification algorithm

\( y \): certificate of \( x \in D \)

\( y \) is a certificate

There is no certificate
Remarks about Certificates

Let D be a decision problem in NP

• For every yes-input x, there exists at least one certificate but possibly many

• Common Terminology
  • Witness, proof, and certificate are used interchangeably
    • Sometimes “the guess” is used as in “the guess of the proof”
  • Succinct (or poly-size): The size of the certificate is bounded by a fixed polynomial in the length of the input
  • Easy to Verify: There is a polynomial time algorithm which takes as inputs: a input to D and an alleged certificate, and verifies that $x \in D$
Remarks on NP

• **P** means decision problems which can be solved in polynomial time, but

• **NP does not mean** decision problems which are `not in polynomial time` rather `decision problems with polynomial time verification algorithms`
  • In particular … **P ⊆ NP** (any problem in P is also in NP)

• Not all decision problems are in **NP**
  • For example, what about:
    • given circuit C decide if there are **no** assignment to the input variables x that makes C(x)=1

We don’t know!
The big question...
Is P=NP?

Exponential time solvable

Is this area empty?
P vs. NP

• Is P=NP?
  • Is it easy to solve as to verify?
  • Can creativity be automated?

• Question open since 1970’s

• How can we make progress on this problem?
  • Find the “hardest” problems in NP and just work on them:
    the NP-complete problems
NP-Complete Problems:

The “hardest” problems in NP

How would you define NP-Complete?
The importance of NP-completeness

- If you recognize your problem is NP-complete, don’t waste trying to solve it in polynomial time
  - Use slow algorithm
  - Settle for less than best
  - Change your problem formulation so it's in P rather than being NP-complete. Sometimes special cases are surprisingly easy.
The impact of NP-completeness

- More than 6000 papers each year have “NP-complete” in title, abstract or keywords
  - Disciplines span statistics, bio, physics, chemistry, math, economics, …
  - More hits than “compiler”, “database”, “neural network”...
  - “Computer Science’s greatest intellectual export” (Papadimitriou 2007)
Two benefits of hard problems

• Source of randomness (Randomness and Complexity course 6.842)

• Cryptography!
  • A man that enjoys our sufferings