6.006- *Introduction to Algorithms*

Lecture 24

NP-completeness

(The Dismal Computer Science)

Prof. Constantinos Daskalakis

Tractable Problems

- We have seen many problems that can be solved in **polynomial time**.
- e.g. finding the shortest path in a graph
- e.g.2 sorting *n* numbers
- e.g.3 finding the exit in a maze
- e.g.4 finding the square root of an integer
- etc
- These problems are **tractable**.
- Polynomial dependence in the input ≈ mild dependence on the input

true for reasonable input size
[but not for "Internet-size",
or "galaxy-size" inputs]

Intractable Problems

- ~~We have seen many~~ problems that can[not] be solved in **polynomial time**.
- e.g. ?
- Suggestion: Count from 1 to a given *n*
- *?!?!?!???!?!?*
- OK, what if I phrase the problem as follows:

- Observation: to represent *n* I just need to provide O(log *n*) digits.

Unsolvable Problems?

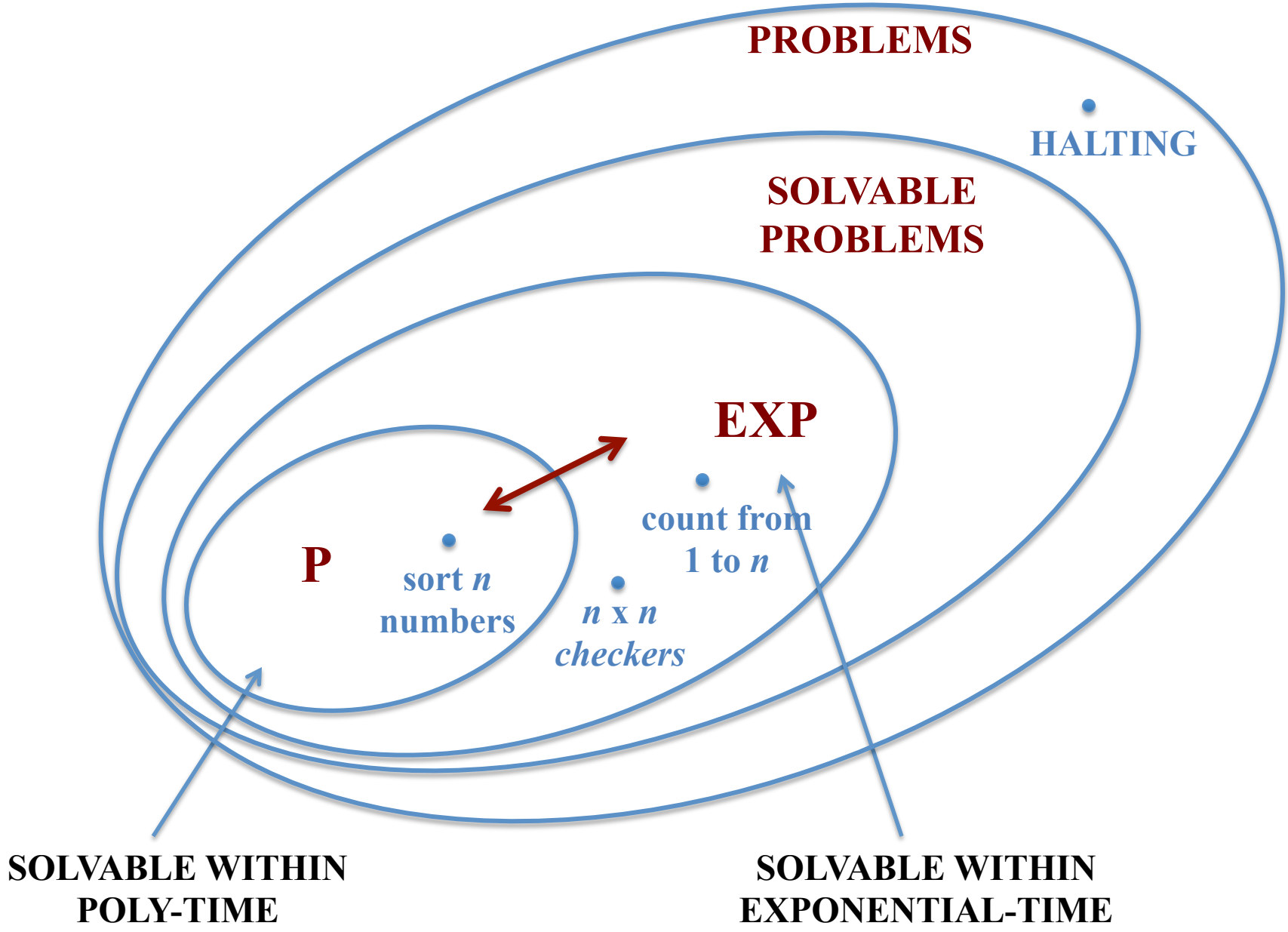
- Are there computational problems that cannot be solved?
- let's try PYTHON: is program P syntactically correct?
- PYTHON can be solved; in particular the python compiler solves it.
- HALTING: Does python program P on input I terminate ?
- e.g.
  - `while True: continue`
    - does not terminate for any input
  - `print "Hello World!"`
    - terminates for any input
- Suppose there exists an Algorithm A solving HALTING, i.e.
- A(P,I)=1 if program P on input I terminates
- A(P,I)=0, if program P on input I runs forever

HALTING PROBLEM

- A(P,I)=1 if program P on input I terminates
- A(P,I)=0, if program P on input I runs forever

- COSTAS
  - Input: Program P
  - if A(P,P)=1, then enter infinite loop
  - else if A(P,P)=0, then stop

- Question: Does COSTAS(COSTAS) terminate?
  - suppose it does, then …
  - suppose it does not, then…
- Contradiction! So there is no algorithm that solves HALTING. More in 6.045


PROBLEMS
HALTING
SOLVABLE
PROBLEMS
EXP
count from
1 to n
P
sort n
numbers
n x n
checkers
SOLVABLE WITHIN
POLY-TIME
SOLVABLE WITHIN
EXPONENTIAL-TIME

Menu

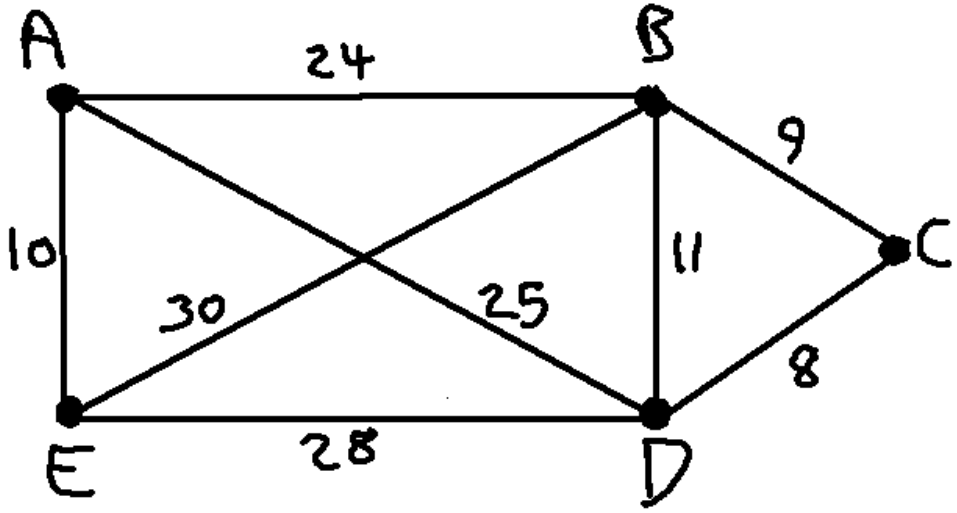
- Classification of problems: P, EXP, unsolvable
- **Problems for which we can't decide yet**
- the class NP
- the P vs NP question

Knapsack Problem

- **Input:**
  - Knapsack of (integer) size S
  - Collection of n items
    - Item i has (integer) size s_i and (integer) value v_i
  - Input size: $\log S + \Sigma_i (\log s_i + \log v_i)$
- **Goal:** Fit maximum value into knapsack
  - i.e., choose subset of items with $\Sigma_i s_i < S$ maximizing $\Sigma_i v_i$
- Gave a DP algorithm running in time $O(n S)$.
- Is there an algorithm that runs in time $\text{poly}(\log S + \Sigma_i (\log s_i + \log v_i))$?

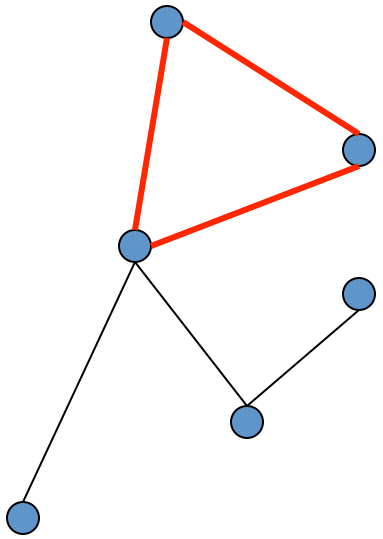
Traveling Salesperson Problem (TSP)

- **Input:** Undirected graph with lengths on edges.
- **Output:** Shortest tour that visits each vertex exactly once.
- Best known algorithm:
  $O(n^2 2^n)$ time.
- Is there a poly-time one?



The CLIQUE problem

- **Input:** Undirected graph G=(V,E)
- **Output:** Largest subset C of V such that every pair of vertices in C has an edge between them.
- Best known algorithm:
  $O(1.1888^n)$ time
- Is there a poly-time one?



Menu

- Classification of problems: P, EXP, unsolvable
- Problems for which we can't decide yet
- **Proving hardness of problems**
- the class NP
- the P vs NP question

What problems are in P?

- We've taught you ways to design polynomial-time algorithms for many problems
- So when faced with a new problem, you'll try applying them in various ways
- What if you don't succeed?
- When can you give up?
- Are there ways for you to know not to waste time trying in the first place?
- Can you prove there's no poly-time algorithm?
- In the problem of counting from 1 through n, the proof is easy as the output itself is exponential in the input.
- But what if the output is polynomial in the input?

Proving a Negative

- How prove there is no polynomial time algorithm for a problem whose solution is polynomial in length?
  - i.e show that there is no algorithm of time O(n), OR $O(n^2)$, OR $O(n^3)$, …

Short Answer: We don't know how to prove such statements.

Don't even have general technique to show that there is no O(n) algorithm

"Proving" a Negative: the Science Way

- How prove no perpetual motion machine?
  - ➢We can prove one exists by building it.
  - ➢We can't prove none exists.
  - ➢Especially if only "evidence" is that we tried and failed.

- Many have tried to build one and failed
  - ➢A preponderance of evidence that is impossible
- But maybe only idiots tried to build PMMs
  - ➢Maybe possible if someone from MIT tries?

A Stronger "Proof"

- Prove that the "laws of physics" preclude its existence.
- Lots of smart people have tested these laws.
  - Gives a real preponderance of evidence the laws are correct.
- If a PMM was possible, it would prove those laws false.
- So unless a very large number of smart people are all wrong, there is no perpetual motion machine.

Menu

- Classification of problems: P, EXP, unsolvable
- Problems for which we can't decide yet
- Proving hardness of problems
  - **hardness via algorithms**
- the class NP
- the P vs NP question

Algorithmic Hardness "Proof"

- Suppose you want evidence that there is no poly-time algorithm for your problem ***Q.***
- Take a problem ***P*** where many scientists have tried and failed to find a poly-time algorithm.
- Prove that if you have a poly-time algorithm for ***Q***, you can use it to build a poly-time algorithm for ***P.***
- Contrapositive: if there is no poly-time algorithm for ***P***, there is no poly-time algorithm for ***Q.***
- All the evidence from those scientists that ***P*** is hard becomes evidence that ***Q*** is hard.

Example: Knapsack

- A "believed hard" problem is **Partition**:
  - Given a set of *n* numbers summing to *S.*
  - Is there a subset of numbers summing to *S*/2?
- We can use this to show **Knapsack** is hard
  - Suppose we have an algorithm A for **Knapsack.**
  - Want to use it to solve **Partition**. How?
  - Given an input $\{s_1,\ldots,s_n\}$ to **Partition.**
  - Consider **Knapsack** problem where item *i* has size s_i and value s_i, and knapsack size is *S*/2.
  - If there is a partition, you can fill the knapsack and get value S/2.
  - Otherwise, best achievable value is < S/2.

Example: Knapsack

- We now have an algorithm for **Partition**:
  - Do a polynomial amount of work to turn the input to **Partition** into an input to **Knapsack**.
  - Call the hypothetical **Knapsack** algorithm.
  - Do a polynomial (actually constant) amount of work to turn the **Knapsack** answer into a **Partition** answer.
    - If Knapsack result is S/2, return YES, else return NO
- If there is a polynomial-time algorithm for **Knapsack**, get one for **Partition**.
- Since we believe no polynomial-time algorithm for **Partition**, conclude none exists for **Knapsack**.

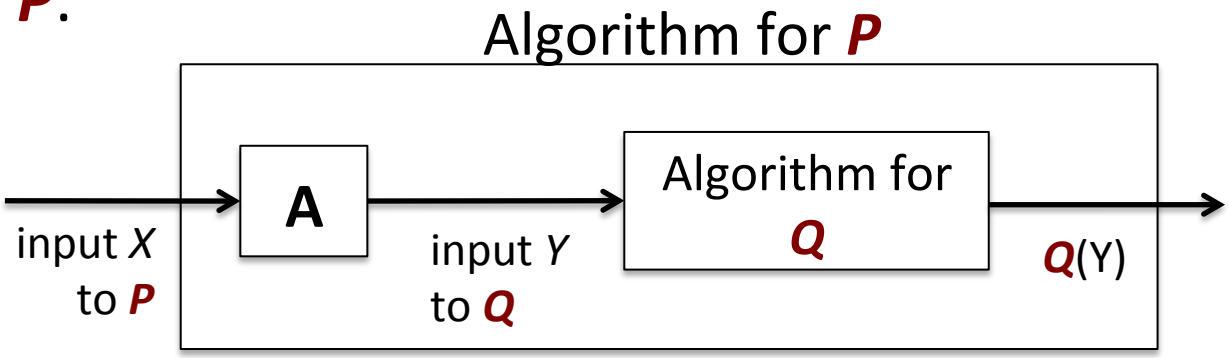
Formalizing our ideas

- We will concentrate on **Decision Problems**
  - These are problems that have a YES or NO answer
- Examples:
  - Given an array, is it sorted in increasing order?
  - Given a list of numbers, are there any duplicates?
  - Given a Knapsack problem, is there a solution (that fits) with total value at least V?
  - Given a graph with positive edge lengths, is there an s-t path of length less than L?
  - Given a graph with edge lengths, is there an s-t path of length greater than L?
  - Given a graph with edge lengths, is there a traveling salesman tour of cost at most C?
  - Given a graph, does it have a clique of size K?

Reductions

- Define a **reduction** from problem ***P*** to problem ***Q***
  - A polynomial-time algorithm A that takes an input X to problem ***P*** and transforms it into an input Y to problem ***Q*** such that:

    P(X)=YES if and only if ***Q***(Y)=YES.
- If there is a poly-time algorithm for ***Q***, the reduction gives one for ***P***:



- Suppose X has size n.
- Then Y=A(X) has size poly(n) (because A is poly-time)
- So overall runtime is poly(|A(X)|) = poly(poly(n)) = poly(n).

Consequence

- If there is a poly-time algorithm for ***Q***, the reduction from ***P*** to ***Q*** gives one for ***P***.
- Contrapositive: If we believe there is no poly-time algorithm for ***P***, we can conclude there is none for ***Q***.
- Reduce ***P*** to ***Q*** → "***Q*** is at least as hard as ***P***"
- Order is important!
  - On the final, at least one person always reduces Q to P and concludes Q is harder than P.

Summary so far

- If problem ***P*** is reduced to problem ***Q***,..
- this shows that ***Q*** is at least as hard as ***P***.
- If people think ***P*** is hard, they'll believe ***Q*** is hard.
- Problem: what is a plausibly hard ***P***?
  - Is there a problem that everyone agrees is hard despite not being able to prove it?
- Solution: Find a whole family of hard problems that can be simultaneously reduced to Q.

Menu

- Classification of problems: P, EXP, unsolvable
- Problems for which we can't decide yet
- Proving hardness of problems
  - hardness via reductions
- **the class NP**
- the P vs NP question

NP

- A decision problem belongs to the class **NP** if:
  - it always has a poly-size solution;
  - whether a proposed poly-size solution is truly a solution can be checked in polynomial-time.
- We say that such problem can be solved in **nondeterministic polynomial time (NP)**.
- In the following sense: We can (non-deterministically) **guess** the solution, then in polynomial-time **check** whether our guess is truly a solution.
- E.g., LONG PATH: Is there an s-t path of length greater than L?
- We can guess a path, then check if its length is larger than L.
- Obstacle: too many possible guesses to simulate deterministically.

The hardest problems in NP

- A problem ***Q*** is **NP-hard** if every problem in NP can be reduced to it
  - i.e., a deterministic polynomial-time algorithm for ***Q*** can be turned into a polynomial-time algorithm for any other NP problem
  - "At least as hard as any NP problem"
- A problem is **NP-complete** if it is in NP and is NP-hard
  - "The hardest problem in NP"
- Cook '73: There is an NP-complete problem!
- Such problem is a good starting point for showing other problems are hard, as it carries with it the hardness of all problems in NP.

Menu

- Classification of problems: P, EXP, unsolvable
- Problems for which we can't decide yet
- Proving hardness of problems
  - hardness via reductions
- the class NP
- **the P vs NP question**

P vs NP

- Many problems have been shown NP-complete
  - Clique, Independent Set, TSP, Graph Coloring, 4-way matching, Vertex Cover, Hamiltonian Path, Longest path, Multiprocessor Scheduling, Max-Cut, Constraint Satisfaction, Quadratic Programming, Integer Linear Programming, Disjoint Paths, Subset Sum…
  - So not just one, but many "hardest problems in NP"
- In 50+ years, scientists haven't found a polynomial-time algorithm for any of them.
- (A poly-time algorithm for one of them, implies a poly-time algorithm for all, as all are reducible to each other)
- The "P vs NP" problem, i.e. answering whether or not there is a poly-time algorithm for any of these problems, is one of the seven millennium prize problems.
- The Clay Mathematics Institute offers $1million for its answer.

Is P=NP?