NP-completeness

CPSC 320 2024W1

Up to now…

- We've been learning techniques to design or analyze efficient algorithms
- Most problems we've seen can be solved in **$O(n)$**, or **$O(n \log n)$**, or **$O(n^2)$** or maybe **$O(n^3)$** time
- Every problem has been "easy" to solve – that is, has an algorithm that runs in **$O(n^k)$** time for some constant **k**

Now:

- We will look at problems that (we think) are hard to solve
- By this we mean: we don’t know of any efficient algorithms for them
- Examples:
  - Find a way to schedule **n** exams in **k** time slots without any students having an exam conflict
  - Find the cheapest route for a traveling business person to visit **n** cities and come back to her starting point
  - Find **k** students in a class of size **n** that (already) all know one another
  - Divide a group of **n** people into two teams of equal total weight to play tug-of-war

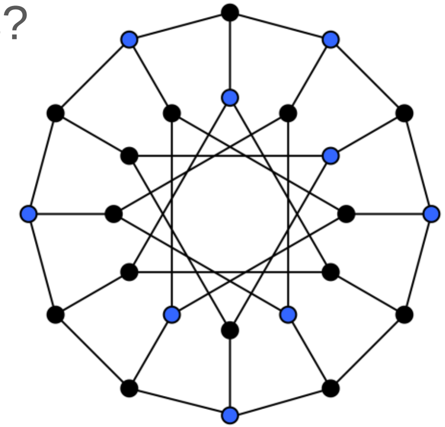
Decision vs. optimization problems

- Two types of problems:
    - Optimization problem: we want to find the solution **s** that maximizes or minimizes a function **f(s)**
    - Decision problem: given a parameter **k**, we want to know if there exists a solution **s** for which **f(s) ≥ k** (maximization) or **f(s) ≤ k** (minimization)
- The two are almost equivalent

Decision vs. optimization problems

Example: Independent Set

- An independent set in a graph G is a subset W of V such that no two vertices in W are joined by an edge in G
- Optimization problem: Given G = (V, E), find the largest independent set in G.
- Decision problem: Given G = (V, E) *and an integer **k**,* does G have an independent set with at least **k** vertices?



Independent set in blue.

Image credit: https://en.wikipedia.org/wiki/Independent_set_(graph_theory)

Decision vs. optimization problems

- The two problems are similar in complexity:
  - If we have a solution to the optimization problem, then it gives us an answer to the decision problem
  - If we can solve the decision problem efficiently, then we can use binary search on **k** to find the answer to the optimization problem
- So we will look at decision problems only

P and NP

- We will consider two sets of problems:
  - **P:** the set of all problems for which we have an efficient solver:
    - Given a problem instance, the solver decides in polynomial time if the answer is **Yes** or **No**
  - **NP:** the set of all problems for which we have an efficient verifier:
    - For every problem instance whose answer is **Yes**, there is a “proof” that a verifier can check in polynomial time

P and NP

Open question: Is P = NP?

- Nobody knows
- Most people think the answer is **No.**

P and NP

- We have:
    - Known easy problems (in P)
    - Problems that we think are hard (in NP, but maybe not P)’
- **Cook’s Theorem:** if SAT can be solved in polynomial time, then **every** problem in NP can be solved in polynomial time
- A problem that belongs to NP and is as hard as SAT (more on what we mean by this later) is called **NP-complete**

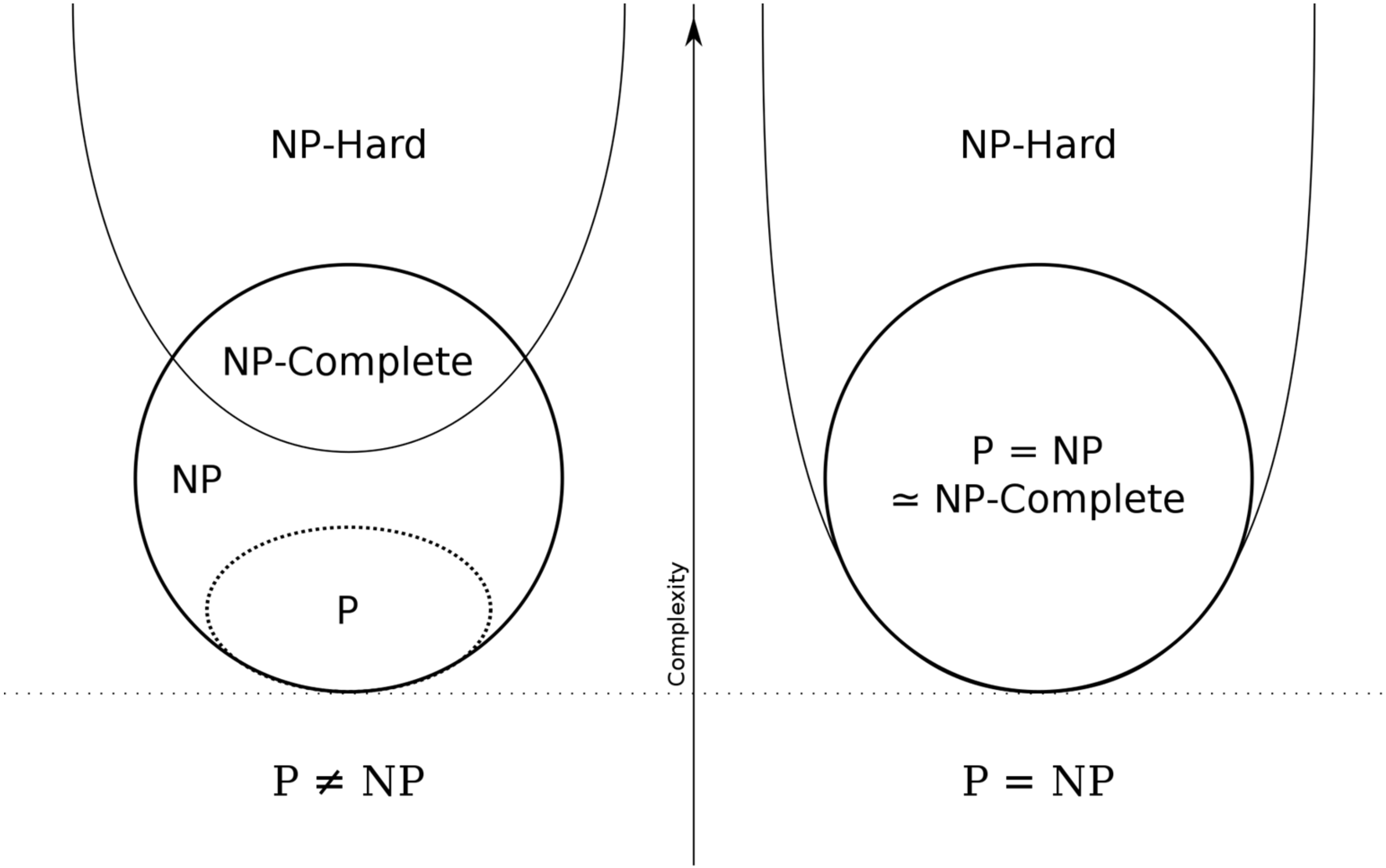


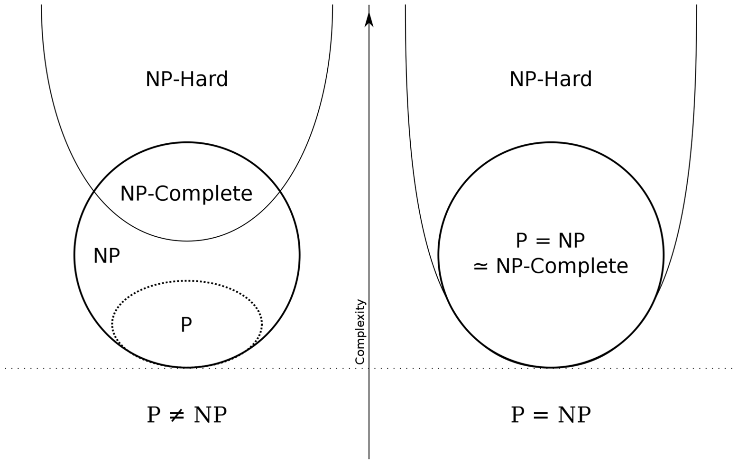
Image retrieved from https://en.wikipedia.org/wiki/NP-completeness, 2024 Mar 14

Familiar NP-complete problems

- We’ve already encountered some NP-complete problems:
  - SAT, 3-SAT (reductions slides), but **not** 2-SAT
  - Tour Grouping problem (A1 – actually graph colouring in disguise)
  - Vertex cover, dominating set (midterm 1)
  - Subset sum (A4)

Proving a new problem is NP-complete

- Two steps for proving that a new problem *P* is NP-complete:
  - Prove that *P* belongs to **NP**
  - Prove that *P* belongs to **NP-hard**



Proving a new problem NP-complete, step 1

- To prove that *P* is in **NP**, we must prove that we can efficiently verify a certificate to the problem
  - **Certificate** is a "proof" of a **Yes** answer; a **verifier** is an algorithm that checks if a certificate is actually a correct proof
  - We can ask an optimization problem as: **Does there exist an <u>X</u> such that <u>Y</u> holds?**
    - Example: Does there exist a <u>subset of nodes in G</u> such that <u>no two of the nodes share an edge and the size of the subset is at least **k**</u>?
  - **Certificate** = whatever goes in **<u>X</u>** (in CPSC 320 vernacular: this is what a "valid solution" looks like)
  - **Verifier** = an algorithm that takes an **<u>X</u>** and checks whether **<u>Y</u>** holds (in CPSC 320 vernacular: this is an algorithm that checks if a "valid solution" is, in fact, a "good solution")

Proving a new problem NP-complete, step 1

- To prove that *P* is in **NP**, we must prove that we can efficiently verify a certificate to the problem
- If a certificate can be verified in **polynomial time**, then *P* is in NP
- Examples:
  - For independent set: the certificate is a subset of the vertices in G. The verifier checks that the subset has size at least **k** and that no two vertices in the subset share an edge (quadratic time)
  - For SAT: the certificate is an assignment of true/false values to variables. The verifier goes through the clauses and checks that each one contains a true literal (linear time)

Clicker Question #1

Recall the Tour Group Problem (FGP) from assignment 1: given a set of students and a matrix K where $K[i][j] = 1$ if and only if student i knows student j, can we find a partitioning of students into three groups such that no two students in the same group know each other?

Which of the following is the **best** certificate for the friend group problem?

A. A list of all the students
B. A subset of the students
C. A subset of the rows of K
D. An assignment of each student to one of the three groups

Clicker Question #1

Recall the Tour Group Problem (FGP) from assignment 1: given a set of students and a matrix K where $K[i][j] = 1$ if and only if student i knows student j, can we find a partitioning of students into three groups such that no two students in the same group know each other?

Which of the following is the **best** certificate for the friend group problem?

A. A list of all the students
B. A subset of the students
C. A subset of the rows of K
D. An assignment of each student to one of the three groups

Proving a new problem NP-complete, part 2

- Next step: prove that P is **at least as hard as any other problem in NP** (i.e., that it's in NP-hard)
- We prove "at least as hard" via polynomial-time reduction:
  - Pick a known NP-complete problem P_{NPC}
  - Give a polynomial-time algorithm that transforms an instance of P_{NPC} into an instance of P with the same Yes/No answer
- If you could solve P efficiently, then you could solve P_{NPC} as follows:
  - Transform it into an instance of P
  - Solve the instance of P and return the same answer
- Since P_{NPC} is NP-complete, this means you could solve every problem in NP in polynomial time!

Proving a new problem NP-complete, part 2

- Next step: prove that P is **at least as hard as any other problem in NP** (i.e., that it's in NP-hard)
- We prove "at least as hard" via polynomial-time reduction:
  - Pick a known NP-complete problem P_{NPC}
  - Give a polynomial-time algorithm that transforms an instance of P_{NPC} into an instance of P with the same Yes/No answer
- **Intuition**: since we can use a solver for P to solve $P_{NPC.}$, this tells us that P is at least as hard as P_{NPC} (its solver is powerful enough to handle $P_{NPC..}$!)

Clicker Question #2

On the midterm exam, we reduced Vertex Cover and Dominating Set to SAT (correctly and in polynomial time). SAT is known to be NP-complete. Does this mean Vertex Cover and Dominating Set are in NP-hard?

A. Yes
B. No

Clicker Question #2

On the midterm exam, we reduced Vertex Cover and Dominating Set to SAT (correctly and in polynomial time). SAT is known to be NP-complete. Does this mean Vertex Cover and Dominating Set are in NP-hard?

A. Yes
B. No

Somewhat trick question: VC/DS *are* NP-hard, but the reduction to SAT doesn't prove it. We've also seen some **non-**NP-hard problems that we've reduced to SAT (e.g., bipartite graph problem).

To prove VC/DS are NP-hard, we would need to reduce **from SAT** (or another NP-complete problem) **to VC/DS**.

How do we solve NP-complete problems?

- Exactly solving an arbitrary instance will take exponential time. Some workarounds:
  - Approximation algorithms: these don’t return an optimal solution, but we have a bound on how bad the solution is (e.g., using a minimum spanning tree to solve TSP gives a solution with no more than twice the optimal cost)
  - Randomization: get a faster running time by allowing the algorithm to fail with some probability
  - Restriction to certain problem types: sometimes certain subtypes of a problem are solvable in polynomial time (e.g., SAT is NP-complete, but we can still solve 2-SAT in poly-time)
  - Heuristics: algorithms (e.g., search-based) that often lack theoretical guarantees but work well enough in practice