CPSC 320: NP-Completeness

Where does this topic fit within the course?

- Up to now:
  - We have been learning “recipes” to design or analyze efficient algorithms.
  - Most of the problems we looked at can be solved in $O(n)$, or $O(n \log n)$, or $O(n^2)$, or maybe $O(n^3)$ time.
  - Every one of these problems was “easy” to solve, that is, has an algorithm that runs in $O(n^k)$ time for some integer $k \geq 0$.

Where does this topic fit within the course?

- Now:
  - We will look at problems that (we think) are hard to solve.
  - It doesn't mean the problems are difficult to describe.
  - Only that we don't know of any efficient algorithms for them.

Where does this topic fit within the course?

- Examples:
  - Find a way to schedule n exams in k time-slots without any exam conflict.
  - Find the cheapest route for a traveling business person to visit n cities and come back to his/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 strength to play tug-of-war.

Decision problems 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: we are given a parameter K, and we need to decide if there is a solution s for which
    - $f(s) \leq K$ [minimization] or
    - $f(s) \geq K$ [maximization].

    So the answer is **True** or **False**.
- They are almost equivalent

Decision problems vs Optimization problems

- Example: Independent Set
  - **Definition**: Given a graph G = (V, E), an **independent set** in G is a subset V' of V such that no two vertices of V' 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?

Decision problems vs Optimization problems

- These two problems are very similar in terms of 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.

The sets 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.

The sets P and NP

Is P = NP ?

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

The sets P and NP

- We have
  - Easy problems.
  - Problems that we think are hard.
- **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 has this property is called NP-complete.

The sets P and NP

- We have already seen many NP-Complete problems:
  - Satisfiability
  - Graph coloring (tutorial 2)
  - Clique, Independent Set (tutorial 3)
  - Vertex Cover, Dominating set (tutorial 3)
  - Hitting Set (assignment 1)
  - Hamiltonian Path (test 1)
  - Monochromatic Triangle (midterm)

Why do we want to know if a problem is NP-complete?

- Suppose your boss asked you to design an algorithm to solve a problem. You would like to avoid this:

  [image deleted to avoid copyright issues]

  text: “I can’t find an efficient algorithm. I guess I’m just too dumb.”

Why do we want to know if a problem is NP-complete?

- Ideally, you would like to do this:

[image deleted to avoid copyright issues]

text: “I can’t find an efficient algorithm, because no such algorithm is possible!”

Why do we want to know if a problem is NP-complete?

- But since you can't (we still don't know if **P = NP**), at least you might be able to do this, and avoid getting fired:

[image deleted to avoid copyright issues]

“I can’t find an efficient algorithm, but neither can all these famous people.”

Proving a new problem NP-Complete

- How do we prove that a problem $\mathcal{P}$ is NP-Complete?
  - First we prove that $\mathcal{P}$ belongs to **NP**. This is (usually) easy:
    - Show what kind of “proof” for Yes answers can be checked in polynomial time.

Proving a new problem NP-Complete

- Example 1:
  - For SAT:
    - The proof is an assignment of true/false values to the variables.
    - We can verify that all clauses are satisfied fairly quickly.

Proving a new problem NP-Complete

- Example 2:
  - For Graph k-colorability:
    - The proof is the colour assigned to each vertex.
    - We can count the different colours and verify there are at most k of them.
    - We can verify that every edge has endpoints of different colours.

Proving a new problem NP-Complete

- How do we prove that a problem $\mathcal{P}$ is NP-Complete (continued)?
  - Then we prove that if we can solve $\mathcal{P}$ in polynomial time, then every other problem in NP can be solved in polynomial time. This is (usually) a bit more involved.
    - The proof of Cook's theorem is ugly (6 pages in Garey & Johnson).
    - Luckily, now that we know that SAT is NP-complete, we can use a much simpler method.

Proving a new problem NP-Complete

- Polynomial-time reduction:
  - Pick a known NP-complete problem $\mathcal{P}_{\text{NPC}}$.
  - Give a polynomial-time algorithm that transforms instances of $\mathcal{P}_{\text{NPC}}$ into instances of $\mathcal{P}$ with the same Yes/No answer.

Instance of $\mathcal{P}_{\text{NPC}}$ $\rightarrow$ Transformation (poly-time) $\rightarrow$ Instance of $\mathcal{P}$

Proving a new problem NP-Complete

- If you could solve $\mathcal{P}$ efficiently, then you could solve an instance of $\mathcal{P}_{\text{NPC}}$ as follows:
  - Transform it into an instance of $\mathcal{P}$.
  - Solve the instance of $\mathcal{P}$ and return the same answer.
- Since $\mathcal{P}_{\text{NPC}}$ is NP-complete, this means you could solve every problem in **NP** in polynomial time!

Coping with NP-completeness

- What do we do if we find the problem we need to solve is NP-Complete?
  - Solving it exactly most likely takes exponential time.
  - Or we can use an approximation algorithm: an algorithm for which we have a bound on how bad the solution is.
  - So we don't get the optimal solution, but we know how far off we are from the optimal.

Coping with NP-completeness

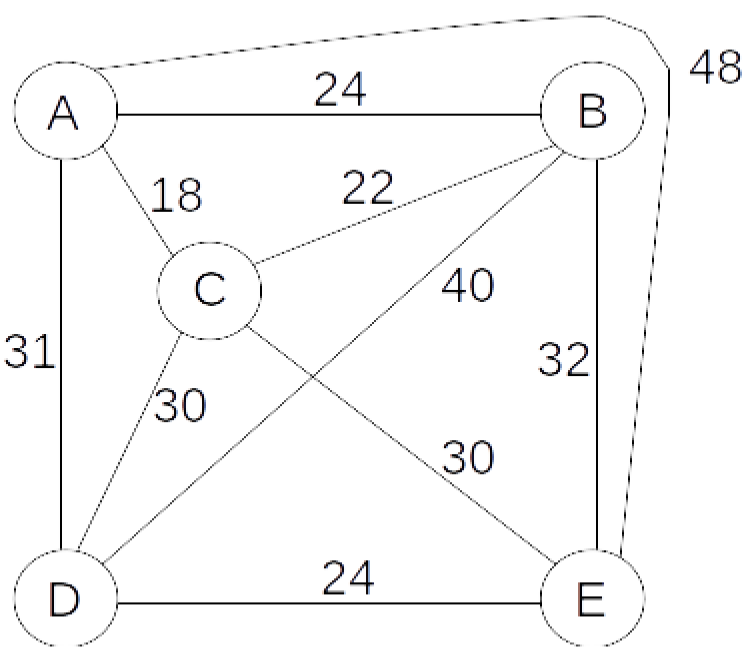
- Example: Traveling salesperson with the triangle inequality.
  - Given:
    - a graph $G = (V, E)$
    - a weight function $w: E \rightarrow R^+$ such that $w(x,z) \leq w(x,y) + w(y,z)$ for all vertices x, y, z.
  - Find a simple cycle that contains every vertex of G, and whose total weight is minimum.

Coping with NP-completeness

- Traveling salesperson with the triangle inequality (continued):
  - Algorithm:
    - Compute a minimum spanning tree of G.
    - Starting from a vertex v, go around the tree.
    - The second (or more) time a vertex w is visited, remove it from the tour.

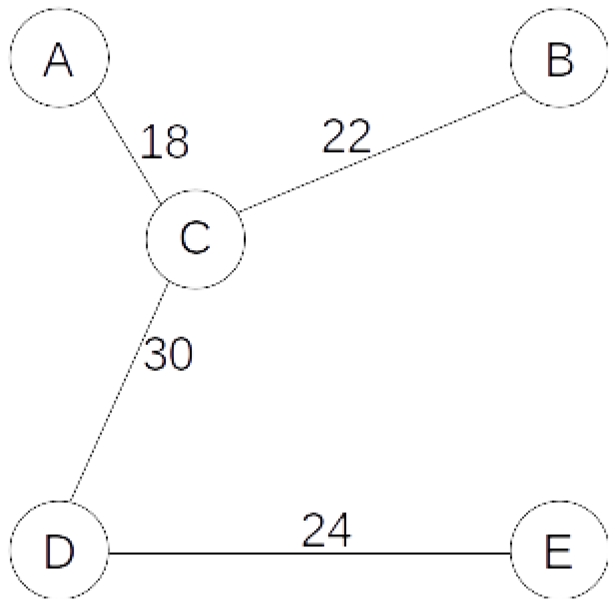
Coping with NP-completeness

- Example: the graph and distances



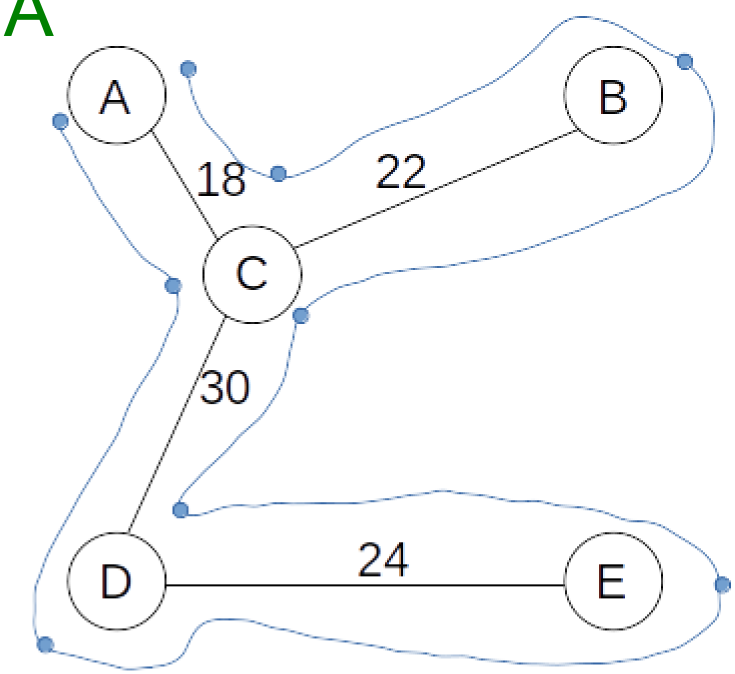
Coping with NP-completeness

- Example: a minimum spanning tree



Coping with NP-completeness

- Example: going around the minimum spanning tree
  - ACDEDCBCA →
    ACDEBA (total weight: 128)
  - Minimum tour: ACBEDA
    (total weight 127)



Coping with NP-completeness

- Traveling salesperson with the triangle inequality (continued):
  - Let
    - W be the weight of the tour produced by this algorithm.
    - MST(G) be the weight of the minimum spanning tree of G.
    - OPT(G) be the weight of the minimum TSP tour.

Coping with NP-completeness

- Traveling salesperson with the triangle inequality (continued):
  - Fact 1: MST(G) ≤ OPT(G)
    - Proof: Take the minimum weight TSP tour. We get a spanning tree T by removing one edge. So MST(G) is ≤ the weight of T, which is ≤ OPT(G).

Coping with NP-completeness

- Traveling salesperson with the triangle inequality (continued):
  - Fact 2: W ≤ 2 MST(G)
    - Proof: the initial tour (the one that goes around the tree) has weight exactly twice that of MST(G)(it follows every edge twice). Because of the triangle inequality the shortcuts can only reduce the weight of the tour.
  - Corollary: W ≤ 2 OPT(G).