Introduction to P and NP
Introduction

Goal

● be familiar with the concept of NP-completeness
● recognize some NP-complete problems
● do some NP-completeness proofs
Introduction

- Design of algorithms
- Analysis of algorithms
- Lower Bounds
  - Can we do better?
  - Do we have the best algorithm
  - How characterize the problems that cannot be solved efficiently
  - Suppose we have an algorithm A for problem P with runtime T(n).
    - Is algorithm A the best?
    - e.g. Branch and Bound for TSP has exponential runtime — is that the best algorithm for TSP?
    - We would need to show that any algorithm for TSP has worst case runtime at least $2^n$ (asymptotically). Such lower bounds are hard to prove.
Classifying problems

Possible classification of computational problems

- Problems that don’t have algorithms. Proved by Alan Turing, 1930’s.
- Problems that have efficient solution (polynomial time)
- Problems that have proven exponential runtime

Major Open Question

- There are many problems, e.g. Travelling Salesman, 0-1 Knapsack, where no one knows a polynomial time algorithm and no one can prove there’s no polynomial time algorithm.
- The best we can do: prove that a large set of problems are equivalent in the sense that a polynomial time algorithm for one yields polynomial time algorithms for all
  - The class of problems with efficient solutions (polynomial time) is the class $\mathbf{P}$
  - The class of equivalent problems are the $\mathbf{NP}$-complete problems.
  - Our main tool is reductions: to prove the problems in this class are equivalent
Polynomial time

- **Definition.** An algorithm runs in polynomial time if its runtime (asymptotic, worst case) is $O(n^k)$ where $n$ is the input size and $k$ is a constant.
  - Examples: Most of the algorithms we've studied have been polynomial time, except for backtracking, branch-and-bound, and the pseudo-polynomial time algorithm for the 0-1 knapsack problem, with runtime $O(nW)$.

  $$O(n^2), O(n^5), O(n \log n), O(2^n), O(n!), O(n^{1,000,000}), O(n^{2^2})$$

  - Polynomial time = “good” = efficient
    - Is it really? Is $O(n^{1000})$ really efficient?
      - Well, no. Curiously, few known algorithms have runtimes like that.
      - On the other hand, some intractable problems can be solved in some restricted form efficiently
    - But generally speaking, has been **extremely successful** in classifying tractable problems
Example of Intractable Problem: TSP

**Input:** A weighted complete graph with non-negative edge weight

**Output:** A tour of minimum weight that visits every vertex only once
Conjecture from 1965 (Jack Edmonds)

- Jack Edmonds: UWaterloo Professor from 1969-1999
- “The classes of problems which are respectively known and not known to have good algorithms are of great theoretical interest. [...] I conjecture that there is no good algorithm for the TSP. My reasons are the same as for any mathematical conjecture: (1) It is a legitimate mathematical possibility, and (2) I do not know.” (1965)

- Jack Edmonds made the following conjecture in 1965

  There is no poly-time algorithm for TSP.

  (equivalent to conjecture: $P \neq NP$)

- To this day, no one has been able to prove/disprove this conjecture.
- How to argue that TSP is an intractable problem?
  - (So far) We have not been able to argue for TSP's intractability in an absolute sense.
  - Instead show “relative” intractability (i.e. show TSP is as hard as bunch of other problems)
Reduction
Reduction

- **Definition.** Problem A reduces to problem B, written $A \leq B$, if an algorithm for B can be used to make an algorithm for A. Think of this as “A is easier than B”.

- $A \leq_p B$
  - if B can be solved in polynomial time, then Y can be solved in polynomial time

**Implications (A \leq_p B)**

- If A cannot be solved in polynomial time, then B cannot be solved in polynomial time
- Even if we don’t have an algorithm for B or a lower bound for A, we can still use reductions to show that problems are equivalently hard (show $A \leq_p B$ and $B \leq_p A$)
Reduction: Example

- A Hamiltonian cycle/path is a cycle/path that visits every vertex of a graph exactly once.

- FACT: No one knows how to find a hamiltonian path/cycle in polynomial time. The best we can do is like trying every possible vertex ordering (exponentially many).
  - A hamiltonian cycle of knight’s moves on a chessboard: has a polynomial time solution
    - https://en.wikipedia.org/wiki/Knight%27s_tour
Reduction: Example

- **Hamiltonian Cycle Problem:**
  - Given a graph, does it have a Hamiltonian cycle?

- **Hamiltonian Path Problem:**
  - Given a graph, does it have a Hamiltonian path?

- **Lemma.** Hamiltonian path problem $\leq_p$ Hamiltonian cycle problem.
  - If there is a polynomial time algorithm for Ham. cycle, then there is one for Ham. path.
  - If there is no poly time algorithm for Ham. path, then there is none for Ham. cycle.
Reduction: Example

- **Lemma.** Hamiltonian path problem \(\leq_p\) Hamiltonian cycle problem.
- **Proof.** Suppose we have a polynomial time algorithm \(A_{\text{cycle}}\) for Hamiltonian cycle. We can call it like a subroutine. We want to make a poly. time algorithm for Hamiltonian path problem:
  - **Input:** graph \(G\).
  - **Output:** Does \(G\) have a Hamiltonian path?
    - Can we just run algorithm \(A_{\text{cycle}}\) on the input \(G\)? NO
    - \(A_{\text{cycle}}\) Returns YES \(\rightarrow\) \(G\) has a hamiltonian cycle \(\rightarrow\) \(G\) has a hamiltonian path
    - \(A_{\text{cycle}}\) Returns NO \(\rightarrow\) We don’t know.
Reduction: Example

- Construct $G'$ such that $G$ has a Hamiltonian path iff $G'$ has a Hamiltonian cycle.
- $A_{\text{path}}$
  - Input: graph $G$
  - Output: Does $G$ have a Hamiltonian path?
- Algorithm $A_{\text{path}}$
  1. construct graph $G'$ by adding one new vertex $v$ adjacent to all vertices of $G$.
  2. run algorithm $A_{\text{cycle}}$ on $G'$
  3. return the YES/NO answer

Runtime: the same runtime as $A_{\text{cycle}}$
Reduction: Example: Proof of correctness

- G has a Hamiltonian path iff G’ has a Hamiltonian cycle.

- Suppose G has hamiltonian path \( u_1, u_2, \ldots, u_n \). Then G’ has hamiltonian cycle \( v, u_1, u_2, \ldots, u_n \).

- Suppose G’ has hamiltonian cycle. Then delete \( v \) to get hamiltonian path in G

This is an example of a **many-one reduction**

- we run algorithm \( A_{\text{cycle}} \) only once and return its answer
Reduction

- **Lemma.** Hamiltonian cycle problem $\leq_p$ Hamiltonian path problem
  - Given $G$, input to the hamiltonian cycle, construct $G'$ input to the hamiltonian path such that $G$ has hamiltonian cycle iff $G'$ has hamiltonian path
  - Given instance of Hamiltonian Cycle $G$, choose an arbitrary node $v$ and split it into two nodes to get graph $G'$

- Now any Hamiltonian Path must start at $v'$ and end at $v''$
Reductions are Transitive

**Claim:** If $A \leq_p B$, and $B \leq_p C$, then $A \leq_p C$

**Proof:** We have to argue if we have a polynomial time algorithm for $C$, we can solve $A$ efficiently. Or $C$ is as hard as $A$.

If we had a poly-time algorithm for $C$

$\Rightarrow$ we could solve $B$ in poly-time (since $B \leq C$)

$\Rightarrow$ we could solve $A$ in poly-time (since $A \leq B$)
Decision vs. Optimization problems
Decision problems vs. Optimization problems

- **Optimization problems**: the answer is an optimal (minimum or maximum) value
- **Decision problem**: answers a given question (problem statement) as **yes** or **no**
Decision problems vs. Optimization problems

- **Knapsack problem**
  - Optimization: What’s the max value we can put into the knapsack?
  - Decision: Can we put value ≥ k into the knapsack?

- **Shortest-path problem:**
  - Optimization: given G, u, v (G a directed graph G) find the shortest path (minimum-weight path) from u to v?
  - Input=(G, u, v, k), where u and v are vertices in G and k is an integer, is there a path of weight k or shorter from u to v?

- **Traveling Salesman Problem:**
  - Optimization: Given a weighted graph, find a Hamiltonian cycle such that the sum of its edge weights is minimum.
    - Finds the best solution
  - Hamiltonian cycle: Given a graph, find a Hamiltonian cycle — a cycle that goes through every vertex exactly once.
  - Finds a solution
Decision problems vs. Optimization problems

optimization and decision are usually equivalent with respect to being solved in polynomial time

- If you can solve the optimization problem (say in time $T$) then you can solve the decision version in time $T$.
- Example: Knapsack-Decision: Can we put value $\geq k$ into the knapsack?

Alg Knapsack-Decision(values, weights, $W$, $k$):

Let $k^* = \text{Knapsack-OPT}(\text{values, weights, } W)$.

if $k < k^*$
    return YES
else
    return NO
Decision problems vs. Optimization problems

optimization and decision are usually equivalent with respect to being solved in polynomial time

- If you can solve the decision problem (say in time $T$) then you can solve the optimization version in time $T \cdot \log(b)$, where $b$ is an appropriate bound on the value that the function we are optimizing can take.
- Example: Knapsack-Decision: Can we put value $\geq k$ into the knapsack?

Alg Knapsack-OPT(values, weights, W):

let $b =$ sum of the values

do binary search (i.e., $k = b$, $b/2$, $b/4$, ..., 1)

Knapsack-Decision(values, weights, W, k)

return maximum $k$ that returns YES
Decision problems vs. Optimization problems

- The theory of NP-completeness focuses on decision problems
  - Usually decision problems are easier or at least no harder than the optimization problems
  - If we can provide evidence that a decision problem is hard —→ we also provide evidence that its related optimization problem is hard

- Equivalence of optimization and decision:
  - there is no general proof but things are usually ok
  - one case where they don’t seem equivalent:
    - testing if a number is prime seems easier than finding its prime factorization (factoring)
    - Decision problem: Given a number, is it prime?
    - Optimization problem: Find prime factorization.
P vs. NP
The class $P$

- $P =$ decision problems that have polynomial time algorithms
  - Is $x$ prime?
    - Algorithm: Agrawal–Kayal–Saxena
  - Is an undirected graph $G$ connected?
  - Is $x$ a multiple of $y$?
The class NP

- **NP** is a large class of decision problems not known to be in P.
- A few problems in NP:
  - Hamiltonian path/cycle
  - Travelling Salesman Problem (decision version)

- Common feature of NP problems
  - if the answer to the decision problem is YES, then there is some succinct information (a certificate) to verify that the answer is YES.
  - A verification algorithm takes input + certificate and checks it
The class NP

- **Definition.** Algorithm A is a **verification algorithm** for the decision problem X if
  - for every input x for problem X, x is a YES for X iff there exists a y (a **certificate**) such that A(x,y) outputs YES
- A is a **polynomial time verification algorithm** if
  - A runs in polynomial time
  - there is a polynomial bound on the size of the certificate y

NP = the class of decision problems that can be verified in polynomial time

NP = Non-deterministic Polynomial time — because the certificate is like a nondeterministic guess
The class NP: Example

Subset Sum $\in$ NP

- Given numbers $w_1, w_2, \ldots w_n$, $W$ is there a subset $S \subseteq \{1, 2, \ldots n\}$ such that

$$\sum_{i \in S} w_i = W$$

- Output: YES
  - Certificate: the set $S$
  - Verification: check that $\sum_{i \in S} w_i = W$

- The above check takes polynomial time
The class NP: Example

TSP (decision version) $\in$ NP

- Given a graph $G$, weights on edges, number $k$, does $G$ have a TSP tour of length $\leq k$

- Output: YES
  - Certificate: a permutation of the vertices ($y$)
  - Verification algorithm:
    - Check $y$ is a permutation
    - Check edges exist to make a cycle
    - Check sum of weights of edges in cycle is $\leq k$
  - The above algorithm takes polynomial time
The class NP: Example

Examples that don’t seem to be in NP

Unique Subset Sum

- Given numbers $w_1, w_2, \ldots, w_n$ and target value $W$, is there a unique subset $S \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i \in S} w_i = W$?

- You can verify that a given $S$ is a solution.
- But how can you verify that $S$ is the only solution?
P and NP: Properties

- **P ⊆ NP**, i.e. if X is in P then X is in NP.
  - **Proof**: The certificate is empty and the verification algorithm is just the polynomial time algorithm for X.

- **Does P = NP?**
  - Is every problem, whose solutions are efficiently verifiable, is also efficiently solvable?
    - If yes, efficient algorithms for TSP, subset sum, …
    - If no,… No efficient algorithms possible for TSP. subset sum , …

We don’t know the answer. But most people believe P ≠ NP
Question

Suppose P ≠ NP. Which of the following are still possible?

A. $O(n^3)$ algorithm for factoring n-bit integers.
B. $O(1.657^n)$ time algorithm for HAMILTONIAN-CYCLE.
C. $O(n^{\log \log \log n})$ algorithm for TSP.
D. All of the above.
Question

Does $P = NP$?

A. Yes.

B. No.

C. None of the above.
Possible outcomes

P ≠ NP

“ In my view, there is no way to even make intelligent guesses about the answer to any of these questions. If I had to bet now, I would bet that P is not equal to NP. I estimate the half-life of this problem at 25–50 more years, but I wouldn’t bet on it being solved before 2100. ”

— Bob Tarjan (2002)
Possible outcomes

P ≠ NP

“We seem to be missing even the most basic understanding of the nature of its difficulty…. All approaches tried so far probably (in some cases, provably) have failed. In this sense P = NP is different from many other major mathematical problems on which a gradual progress was being constantly done (sometimes for centuries) whereupon they yielded, either completely or partially.”

Possible outcomes

P = NP

“I think that in this respect I am on the loony fringe of the mathematical community: I think (not too strongly!) that P=NP and this will be proved within twenty years. Some years ago, Charles Read and I worked on it quite bit, and we even had a celebratory dinner in a good restaurant before we found an absolutely fatal mistake.”

— Béla Bollobás (2002)
Possible outcomes

P = NP

“ In my opinion this shouldn’t really be a hard problem; it’s just that we came late to this theory, and haven’t yet developed any techniques for proving computations to be hard. Eventually, it will just be a footnote in the books. ”

— John Conway
Possible outcomes

$P = NP$, but only $\Omega(n^{100})$ algorithm for 3-SAT.

$P \neq NP$, but with $O(n^{\log^*n})$ algorithm for 3-SAT.

“ It will be solved by either 2048 or 4096. I am currently somewhat pessimistic. The outcome will be the truly worst case scenario: namely that someone will prove $P = NP$ because there are only finitely many obstructions to the opposite hypothesis; hence there exists a polynomial time solution to SAT but we will never know its complexity! ”

— Donald Knuth
P and NP: Properties

- Millennium prize
  - worth $1 million (Millenium Prize)
NP-Complete
The class of NP-complete

Definition. A decision problem \( X \) is **NP-complete** if

- \( X \in \text{NP} \)
- for every \( Y \) in \( \text{NP} \), \( Y \leq_P X \)

Two important implications of \( X \) being NP-complete

- if \( X \) can be solved in polynomial time then so can every problem in \( \text{NP} \)
  - if \( X \in \text{P} \) then \( \text{P} = \text{NP} \)
- if \( X \) cannot be solved in polynomial time then no NP-complete problem can be solved in polynomial time
The class of NP-complete

The first NP-completeness proof is difficult — must show that every problem $Y \in \text{NP}$ reduces to $X$

Subsequent NP-completeness proofs are easier because $\leq_p$ is transitive:

**Claim.** If $Y \leq_p X$ and $X \leq_p Z$ then $Y \leq_p Z$

So to prove $Z$ is NP-complete, we just need to prove $X \leq_p Z$ where $X$ is a known NP-complete problem.
The class of NP-complete

- To prove a decision problem $Z$ is NP-complete
  - 1. prove $Z$ in NP
  - 2. prove $X \leq_p Z$ for some known NP-complete problem $X$

The first NP-complete problem: Circuit Satisfiability

The second NP-complete problem: Satisfiability
History of NP-completeness

- < 1970: lots of unsolved problems
- 1971: Cook-Levin Theorem: SAT is NP-complete (just from the definition of NP)
- 1972: Karp: By showing SAT reduces to 21 other problems, showed the existence of 21 other NP-complete problems. (why?)
- Since 1972: 1000s of problems are NP-complete.
History of NP-completeness
History of NP-completeness

1971: Cook-Levin
History of NP-completeness

1972: Karp
History of NP-completeness
C** is NP-complete!
If we can solve C** efficiently => we solve C* efficiently => we solve all NP problems efficiently
All of these problems (and many, many more) poly-time reduce to SAT.
Implications of Karp

SAT

3-SAT

INDEPENDENT-SET

3-SAT poly-time reduces to INDEPENDENT-SET

DIR-HAM-CYCLE

3-COLOR

SUBSET-SUM

VERTEX-COVER

HAM-CYCLE

KNAPSACK

SAT poly-time reduces to all of these problems (and many, many more)
Implications of Karp + Cook-Levin

All of these problems are NP-complete; they are manifestations of the same really hard problem.