Chapter 34. P and NP

Tractability

- Some problems are ***intractable***:
  as they grow large, we are unable to solve them <u>in reasonable time</u>
- What constitutes reasonable time?
  - Standard working definition: ***polynomial time***
  - On an input of size *n* the worst-case running time is $O(n^k)$ for some constant *k*
  - $O(n^2)$, $O(n^3)$, $O(1)$, $O(n\log n)$, $O(2^n)$, $O(n^n)$, $O(n!)$
  - Polynomial time: $O(n^2)$, $O(n^3)$, $O(1)$, $O(n\log n)$
  - Not in polynomial time: $O(2^n)$, $O(n^n)$, $O(n!)$

Polynomial-Time Algorithms

- Are some problems solvable in polynomial time?
  - Of course: many algorithms we’ve studied provide polynomial-time solutions to some problems
- Are all problems solvable in polynomial time?
  - No!: Turing’s “Halting Problem” is not solvable by any computer, no matter how much time is given
- Most problems that do not yield polynomial-time algorithms are either optimization or decision problems.

Optimization/Decision Problems

- Optimization Problems
  - An optimization problem is one which asks, "What is the optimal solution to problem X?"
  - Examples:
    - 0-1 Knapsack
    - Fractional Knapsack
    - Minimum Spanning Tree
- Decision Problems
  - An decision problem is one with yes/no answer
  - Examples:
    - Does a graph G have an MST of weight $\leq$ W?

Optimization/Decision Problems

- Introduce parameter k and ask if the optimal value for the problem is a most or at least k.
  Optimization problem turns into decision problem
- Many problems will have decision and optimization versions
  - Ex: Traveling salesman problem
    - optimization: find Hamiltonian cycle of minimum weight
    - decision: is there a Hamiltonian cycle of weight $\leq k$

The Class *P*

P: the class of decision problems that have polynomial-time deterministic algorithms.

- That is, they are solvable in $O(p(n))$, where $p(n)$ is a polynomial on n
- A deterministic algorithm is (essentially) one that always computes the correct answer

Why polynomial?

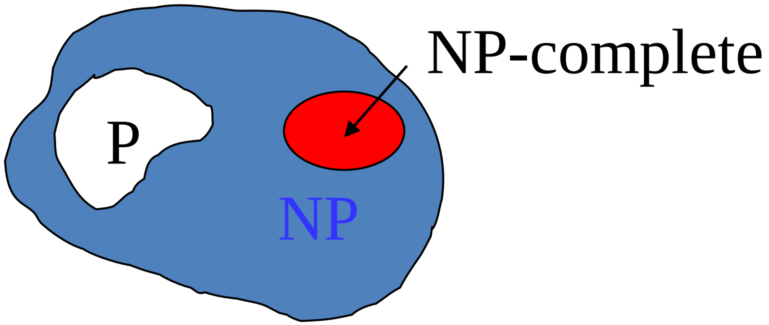
- if not, very inefficient
- nice closure properties
  - *the sum and composition of two polynomials are always polynomials, too.*

The class NP

- NP stands for Nondeterministic Polynomial
- Nondeterministic computation: “guess” or “parallelize”

- A problem can be solved in nondeterministic polynomial time if: given a guess at a solution for some instance of size n, we can check that the guess is correct in polynomial time (i.e. the check runs $O(n^k)$)
- The class of problems where the solution can verified “quickly”
  - In polynomial time in the size of the input

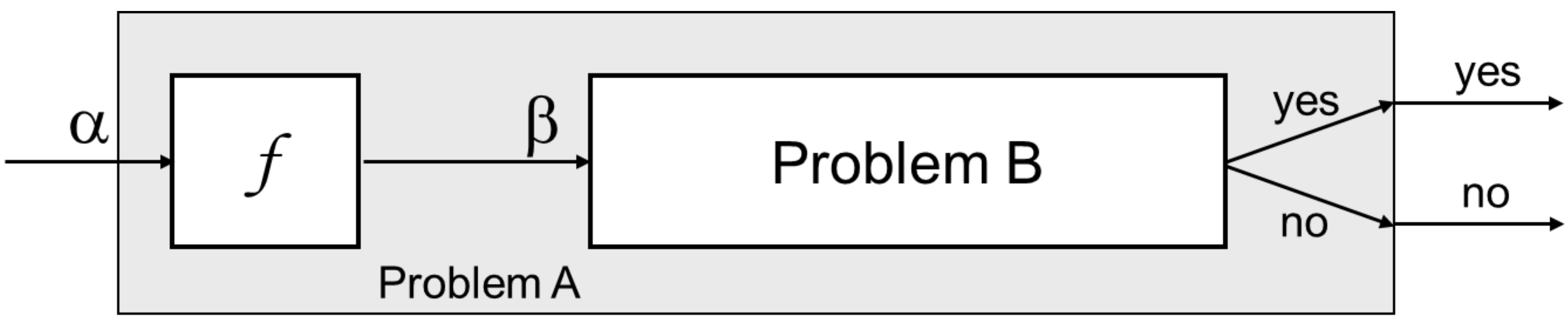
NP-Completeness (informally)



- NP-complete problems are defined as the hardest problems in NP
- Most practical problems turn out to be either P or NP-complete.

Reductions

- Reduction is a way of saying that one problem is **easier** than another.
- We say that problem A is easier than problem B,
(i.e., we write **"A ≤ B"**) if we can solve A using the algorithm that solves B.
- Idea: transform the inputs of A to inputs of B

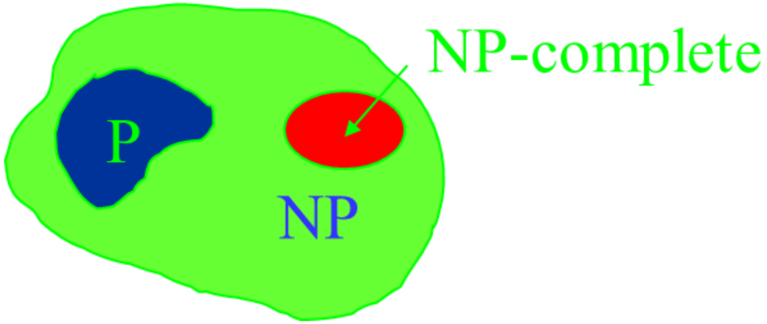


Polynomial Reductions

- Given two problems A, B, we say that A is polynomially <u>reducible</u> to B ($A \leq_p B$) if:

1. There exists a function f that converts the input of A to inputs of B in polynomial time
2. $A(i) = YES \Leftrightarrow B(f(i)) = YES$

NP-Completeness (formally)



- A problem B is <u>NP-complete</u> if:
  - (1) $B \in NP$
  - (2) $A \leq_p B$ for all $A \in NP$
- If B satisfies only property (2) we say that B is <u>NP-hard</u>

- <u>No polynomial time algorithm has been discovered for an NP-Complete problem</u>
- <u>No one has ever proven that no polynomial time algorithm can exist for any NP-Complete problem</u>

P & NP-Complete Problems

- **Shortest simple path**
  - Given a graph G = (V, E) find a **shortest** path from a source to all other vertices
  - Polynomial solution: O(VE)
- **Longest simple path**
  - Given a graph G = (V, E) find a **longest** path from a source to all other vertices
  - NP-complete

P & NP-Complete Problems

- Euler tour
  - G = (V, E) a connected, directed graph find a cycle that traverses <u>each edge</u> of G exactly once (may visit a vertex multiple times)
  - <u>Polynomial solution O(E)</u>

- Hamiltonian cycle
  - G = (V, E) a connected, directed graph find a cycle that visits <u>each vertex</u> of G exactly once
  - <u>NP-complete</u>

The Satisfiability (SAT) Problem

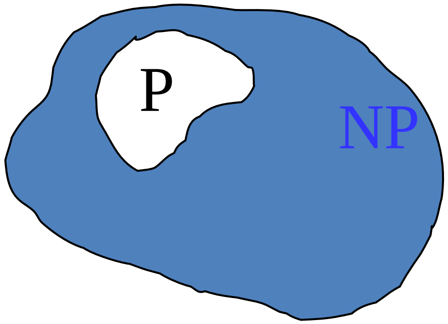
- Satisfiability (SAT):
  - Given a Boolean expression on n variables, can we assign values such that the expression is TRUE?
  - Ex: $((x1 \rightarrow x2) \vee \neg((\neg\, x1 \leftrightarrow x3) \vee x4)) \wedge \neg\, x2$
  - Seems simple enough, but <u>no known deterministic polynomial time algorithm exists</u>
  - But <u>easy to verify in polynomial time!</u>
  - SAT was the first problem shown to be NP-complete!

P vs. NP

- P = problems that can be solved in polynomial time
- NP = problems for which a solution can be verified in polynomial time

- Problems in P: efficient <u>discovery</u> of a solution
- Problems in NP: efficient <u>verification</u> of a solution

Is P = NP?



- Any problem in P is also in NP:

$$P \subseteq NP$$

- The big (and **open question**) is whether $P \subseteq NP$ or $P = NP$
- the Clay Mathematics Institute has offered a $1 million prize for the first proof
- <u>*Most computer scientists believe that this is false but we do not have a proof …*</u>

Classes of problems