Polynomial-time reductions

We have seen several reductions:

max bipartite matching to max flow

(i.e. we solved max bipartite match. using a max flow algorithm)

etc.
Polynomial-time reductions

Informal explanation of reductions:

We have two problems, X and Y. Suppose we have a black-box solving problem X in polynomial-time. Can we use the black-box to solve Y in polynomial-time?

If yes, we write $Y \leq_p X$ and say that Y is polynomial-time reducible to X.
Polynomial-time reductions

Informal explanation of reductions:

We have two problems, X and Y. Suppose we have a black-box solving problem X in polynomial-time. Can we use the black-box to solve Y in polynomial-time?

If yes, we write $Y \leq_p X$ and say that Y is polynomial-time reducible to X.

More precisely, we take any input of Y and in polynomial number of steps translate it into an input (or a set of inputs) of X. Then we call the black-box for each of these inputs. Finally, using a polynomial number of steps we process the output information from the boxes to output the answer to problem Y.
Polynomial-time reductions

Polynomial-time: what is it?

Class of problems P:

- Consider problems that have only YES/NO output

- Every such problem can be formalized - e.g. encode the input into a sequence of 0/1 and the problem is defined as the union of all input sequences for the YES instances

- Polynomial-time algorithm runs (on a Turing machine) in time polynomial in the length of the input, e.g. for an input of length \( n \) the algo takes (e.g.) \( O(n^4) \) steps to determine if this input is a YES instance

Examples:
- Is a path from s to t?
- Is negative cycle?
Polynomial-time reductions

Example:

Problem 1: **CNF-SAT**

Given is a conjunctive normal form (CNF) expression such as:

\[(x \lor y \lor z) \land ((\neg x) \lor z \lor w) \land \ldots \land ((\neg w) \lor x) = \varnothing\]

Question: Does there exist a satisfiable assignment?

\[\varnothing \text{ is satisfiable bec. } \begin{array}{l} x = \text{TRUE} \\ w = \text{TRUE} \end{array}\]
Polynomial-time reductions

Example:

Problem 2: Clique

Given is a graph $G = (V,E)$ and number $k$.

Question: Does there exist a clique of size $k$, i.e. a subset of vertices $S$ of size $k$ such that for every $u,v$ in $S$, $(u,v)$ is in $E$?

$G$: $k = 4$

$k=5$: NO

YES
Polynomial-time reductions

Example:

Goal: show $\text{CNF-SAT} \leq_p \text{CLIQUE}$.

Let $G$ contain a vertex for every variable in $\Phi$ (if $x_i$ occurs twice, there will be two vertices).

- Add an edge between two vertices if:
  - From different clauses
  - If not a negation of each other

Let $k = \#\text{clauses}$.
Polynomial-time reductions

Example:

Goal: show $\text{CNF-SAT} \leq_p \text{CLIQUE}$.

(Given an instance of CNF-SAT, convert to an instance of CLIQUE so that ... (what ?).)
Polynomial-time reductions

Why reductions?
Polynomial-time reductions

Why reductions?

• to solve our problem with not much work (using some already known algorithm)

• to say that some problems are harder than others
Class NP

Class P
• YES/NO problems with a polynomial-time algorithm

Class NP
• YES/NO problems with a polynomial-time “checking algorithm” - more precisely, given a solution (e.g. a subset of vertices) we can check in a polynomial time if that solution is what we are looking for (e.g. is it a clique of size k?)

Example: Show that CNF-SAT is in NP.
What is the thing we want to check?
How does the “checking algorithm” work in this case?

1) guess a solution (e.g. assignment T/F to the variables)
2) verify the solution (e.g. check that the assignment makes # True)
Class NP

Class P
• YES/NO problems with a polynomial-time algorithm

Class NP
• YES/NO problems with a polynomial-time “checking algorithm” - more precisely, given a solution (e.g. a subset of vertices) we can check in a polynomial time if that solution is what we are looking for (e.g. is it a clique of size k ?)

Example: Show that CNF-SAT is in NP.

Now consider CNF-UNSAT, the problem of unsatisfiable formulas (YES instances are the unsatisfiable formulas, not the satisfiable ones as in CNF-SAT). Is CNF-UNSAT in NP ?
Class NP

Class P

- YES/NO problems with a polynomial-time algorithm

Class NP

- YES/NO problems with a polynomial-time “checking algorithm” - more precisely, given a solution (e.g. a subset of vertices) we can check in a polynomial time if that solution is what we are looking for (e.g. is it a clique of size k?)

In short:

P - find a solution in polynomial-time

NP - check a solution in polynomial-time
Class **NP**

Class **P**
- YES/NO problems with a polynomial-time algorithm

Class **NP**
- YES/NO problems with a polynomial-time “checking algorithm” — more precisely, given a solution (e.g. a subset of vertices) we can check if that solution is what we are looking for (e.g. is it a clique of size k?)

In short:

**P** — **find** a solution in polynomial-time

**NP** — **check** a solution in polynomial-time

**BIG OPEN PROBLEM**

Is **P = NP**?
NP-complete and NP-hard

**NP-hard**

A problem is NP-hard if all other problems in NP can be polynomially reduced to it.

**NP-complete**

A problem is NP-complete if it is (a) in NP, and (b) NP-hard.

In short:
**NP-complete**: the most difficult problems in NP
NP-complete and NP-hard

**NP-hard**
A problem is NP-hard if all other problems in NP can be polynomially reduced to it.

**NP-complete**
A problem is NP-complete if it is (a) in NP, and (b) NP-hard.

In short:
**NP-complete**: the most difficult problems in NP

Why study them? Find a polynomial-time algo for any NP-complete problem, or prove that none exists. (Either way, no worry about job offers till the end of your life.)
NP-complete and NP-hard: how to prove

Given: a problem
Suspect: polynomial-time algorithm unlikely
Want: prove that the problem is NP-hard or NP-complete (thus a polynomial-time algorithm VERY unlikely)

How to prove this?
Given: a problem

Suspect: polynomial-time algorithm unlikely

Want: prove that the problem is NP-hard or NP-complete (thus a polynomial-time algorithm VERY unlikely)

How to prove this?

Thm (Cook-Levin): CNF-SAT is NP-hard.
Given: a problem

Suspect: polynomial-time algorithm unlikely

Want: prove that the problem is NP-hard or NP-complete (thus a polynomial-time algorithm VERY unlikely)

How to prove this?

**Thm (Cook-Levin):** CNF-SAT is NP-hard.

We have already proved that CLIQUE is NP-hard. How come?
The recipe to prove NP-hardness of a problem $X$:
1. Find an already known NP-hard problem $Y$.
2. Show that $Y \leq_p X$.

The recipe to prove NP-completeness of a problem $X$:
1. Show that $Y$ is NP-hard.
2. Show that $Y$ is in NP.
INDEPENDENT SET problem

Input: A graph $G=(V,E)$ and an integer $k$

Output: Does there exist an independent set of size $k$, i.e. a subset of vertices $S$ of size $k$ such that for every $u,v$ in $S$, $(u,v)$ is not in $E$?

Let $G'$ be $G$ with $E'$ where $E' = V \times V - E$, let $k' = k$.

$G$: $k = 4$

$G'$: YES

NP-complete: $\langle G,k \rangle$

NP-hard: $\langle G',k' \rangle$

Solution: a set of $k$ vertices verify: no edge between any pair of these vertices

NP-complete and NP-hard: examples
NP-complete and NP-hard: examples

INDEPENDENT SET problem

Input: A graph $G=(V,E)$ and an integer $k$

Output: Does there exist an independent set of size $k$, i.e. a subset of vertices $S$ of size $k$ such that for every $u,v$ in $S$, $(u,v)$ is not in $E$?

Is INDEPENDENT SET problem NP-complete?
NP-complete and NP-hard: examples

VERTEX COVER problem

Input: A graph $G=(V,E)$ and an integer $k$

Output: Does there exist a subset of vertices $S$ of size $k$ such that every edge has at least one endpoint in $S$

$G$: $k = 5$

in NP:

solution: a set of vertices
verify: every edge has an endpoint in the set
NP-complete and NP-hard: examples

VERTEX COVER problem

Input: A graph $G=(V,E)$ and an integer $k$

Output: Does there exist a subset of vertices $S$ of size $k$ such that every edge has at least one endpoint in $S$

Recall:

CNF-SAT, CLIQUE, INDEPENDENT SET all NP-complete.

We will show that INDEPENDENT SET $\leq_p$ VERTEX COVER.
Lemma: INDEPENDENT SET $\leq_p$ VERTEX COVER.
Other well-known NP-complete problems

**HAMILTONIAN CYCLE**

**Input:** A graph \( G \)

**Output:** Is there a cycle going through every vertex (exactly once)?
Other well-know NP-complete problems

TRAVELING SALESMAN PROBLEM (TSP)

Input: A complete weighted graph $G = (V, V \times V)$ with weights $w$, a threshold number $t$

Output: Is there a cycle going through every vertex (exactly once), with total weight of the cycle $< t$?

$G,w:$
$t = 14$

in $\text{NP}$: solution: a permutation of vertices
verify: overall cycle length $< t$
Other well-know NP-complete problems

TRAVELING SALESMAN PROBLEM (TSP)

Input: A complete weighted graph $G = (V, V \times V)$ with weights $w$, a threshold number $t$

Output: Is there a cycle going through every vertex (exactly once), with total weight of the cycle $< t$?

Is TSP NP-complete?
Other well-know NP-complete problems

3-COLORING

Input: A graph $G$

Output: Is it possible to color vertices of $G$ by three colors so that no edge has its end-points colored by the same color?
Other well-know NP-complete problems

Remarks about coloring problems:
• 2-COLORING is in P (what is the algorithm?)
• 3-COLORING is NP-complete
• how about 4-COLORING?
Other well-known NP-complete problems

**KNAPSACK**

(sometimes also disguised as problem named **SUBSET-SUM**)

- we have $O(nW)$ algorithm for KNAPSACK
- but KNAPSACK is NP-complete
- how come?
Decision vs. construction

Suppose we have a black-box answering YES/NO for the 3-COLORING problem. Can we use it to find a 3-coloring?