Polynomial-time reductions

We have seen several reductions:

- max flow problem to linear programming
  (i.e. used lin. prog. to solve max flow)

- bipartite max. matching to max flow
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:

1) Is there a negative-weight cycle?
2) Is there a path from $s$ to $t$?
3) Is there a flow of value $\geq k$?

(makes it YES/NO)
Polynomial-time reductions

Example:

Problem 1: CNF-SAT

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

\[(x \text{ or } y \text{ or } z) \text{ and } ((\text{not } x) \text{ or } z \text{ or } w) \text{ and } x \text{ and } ((\text{not } w) \text{ or } x)\]

Question: Does there exist a satisfiable assignment?

\[\begin{align*}
  z &= \text{true} \\
  x &= \text{true} \\
\end{align*}\]

Satisfiable
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:**

$k = 4$

$\text{YES}$

$k = 5$ ?

$\text{NO}$
Polynomial-time reductions

Example:

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

\[
\phi = (x_1 \lor x_2) \land (x_1 \lor x_3) \land (x_3 \lor x_4, \lor x_2)
\]
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

\[ Y \leq_{p} X \]

if \( Y \) is "really hard", then \( X \) must be "really hard"
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 (here, a T/F assignment to the variables)
2) verify the solution (here, plug the T/F assignment in to see if \( \Phi \) is 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 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

**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

If solve any NP-complete problem in P-time, then every problem in NP has a P-time algorithm.
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 \( \mathcal{Y} \)

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?

- know NP-hard problem \( \mathcal{X} \)
- you show that \( \mathcal{X} \leq_p \mathcal{Y} \)
**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?

**Thm (Cook-Levin):** $\text{CNF-SAT}$ is NP-hard.
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?

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 $E$?

$G$: $k = 4$

Let $G'=(V,E')$ where $E' = V \times V - E$, $k' = k$.
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 $k$ vertices
Verify: are all edges covered

Yes
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.
NP-complete and NP-hard: examples

Lemma: **INDEPENDENT SET** $\leq_p$ **VERTEX COVER**.
Other well-know 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 < \( t \)?

\[ G, w: \]
\[ t = 14 \]
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?
Remarks about coloring problems:

- 2-COLORING is in P (what is the algorithm?)
- 3-COLORING is NP-complete
- how about 4-COLORING?
Other well-know 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?