Polynomial-time reductions

We have seen several reductions:

- sorting to convex hull
- longest incr. subseq. to longest common subseq.
- network flows to lin. prog.
- multi-source multi-sink max flow to single-source single-sink max flow
- max bipartite 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
Polynomial-time reductions

Example:

Problem 1: **CNF-SAT**

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

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

Question: Does there exist a satisfiable assignment?

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

e.g. \[
\begin{align*}
x &= \text{anything} \\
y &= \text{anything} \\
z &= T \\
w &= F
\end{align*}
\]
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 $E$?

$G$: $k = 4$

Brute force:
- try all subsets $O(2^n \cdot k^2)$
- or try only size $k$: $O((\binom{n}{k}) \cdot k^2)$

$k = 3$ **YES**
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 ?).)
Polynomial-time reductions

Why reductions?

A ≤ₚ B

if B is poly-time, then A is poly-time
B ∈ P → A ∈ P

if A is "hard"
could it be that B ∈ P?

No bec.
B must be "hard"
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?
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
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 A

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?

Take a known NP-hard (NP-complete) problem B and show: $B \leq_p A$
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.
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 $E$?

$G$: $k = 4$

$\text{YES}$

in NP: set of $k$ vertices
verify: no edges
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 $S$?  

$G$:  

$k = 5$

YES

In NP:
Solution: a set of vertices of size $k$
Verify: every edge covered
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**.

Let's look at the complement of the IS:
- Is it a vertex cover?

We need to show that every edge has an endpoint that is in the VC, i.e., that has an endpoint that is not in the IS (not circled).

**YES**, bec. if both endpoints in IS, then that would not be an IS!

And vice versa, complement of VC is an IS.
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 < \( t \) ?

\[
G, w: \\
t = 14
\]

\[
\text{YES} \quad \text{in NP} \quad \text{similar to HC}
\]
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?

Take HC $G$ : create a complete graph, $eeE(G)$ of weight $1$

$e \in E(G)$

$t = n$
**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-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?