Lecture 11: Polynomial-time Reductions, NP-hardness

Yury Makarychev
TTIC and the University of Chicago

Hard Problems

In this class, we discussed many techniques for solving combinatorial optimization problems.

- Greedy algorithms
- Dynamic Programming
- Reducing a problem to Max Flow / Min Cut
- Linear Programming

Using them, we designed polynomial-time algorithm for many problems.

Now, we will look at problems that cannot be solved in polynomial time.

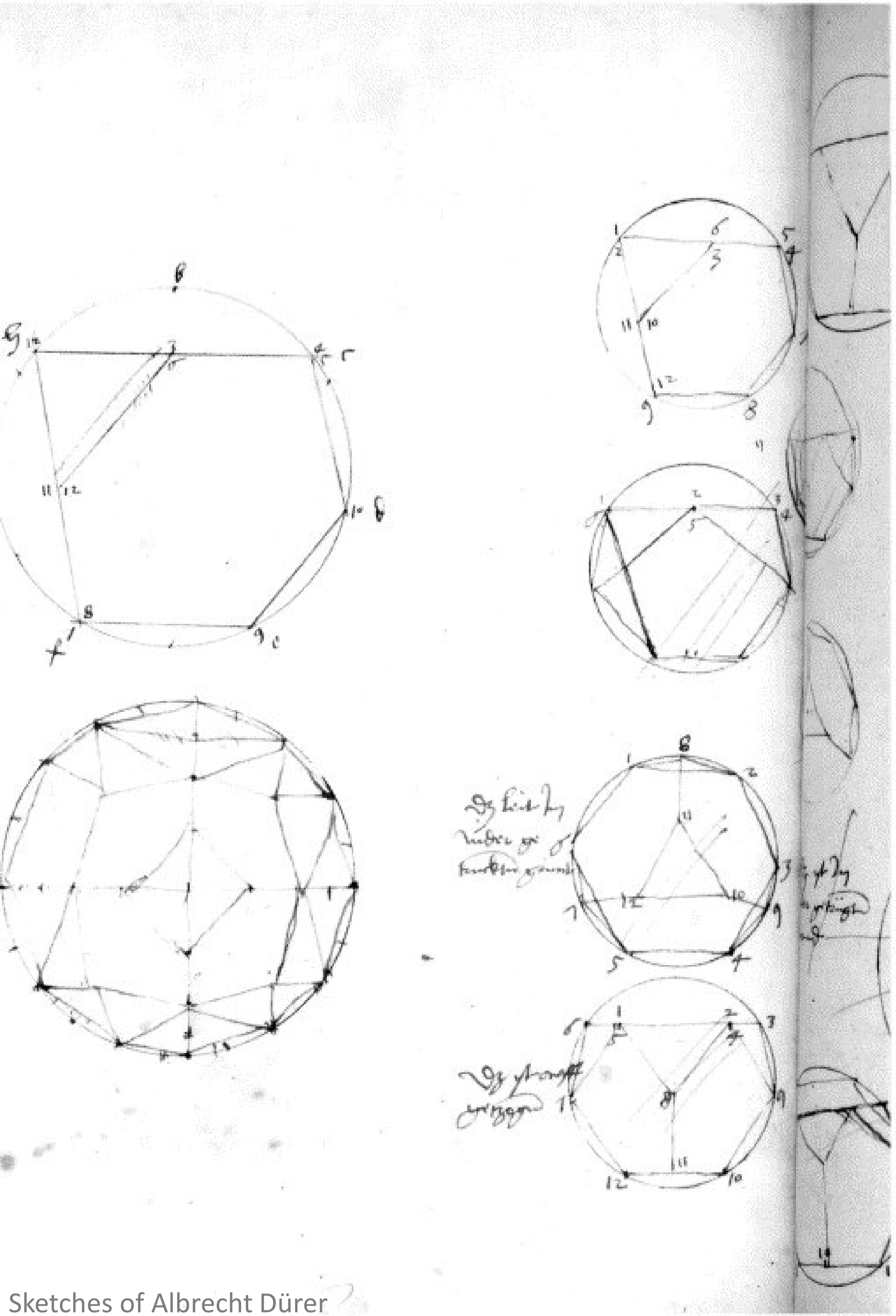
Undecidable Problems

There are very hard problems that cannot be solved by any algorithm.

Halting Problem: Given a Turing Machine (an algorithm), decide whether it will ever stop.

Busy Beaver Problem: Given n, find the largest number that a program of size at most n can output?

Automatic Theorem Proving: Given a theorem statement, decide whether it can be proved (e.g., in the standard axiom system ZFC).



Sketches of Albrecht Dürer

Problems can be solved in exponential or super-exponential times

Bounded Halting Problem: Given a Turing Machine M and parameter T (written in binary), decide if M stops after executing at most T steps.

Euclidean Geometry: Given a geometric statement, either prove or disprove it.

Problems with huge output: Given n, print 2^n zeroes.

What about optimization problems?

Problems we just saw are quite different from those we studied in this course.

We are interested in problems more similar the ones we studied.

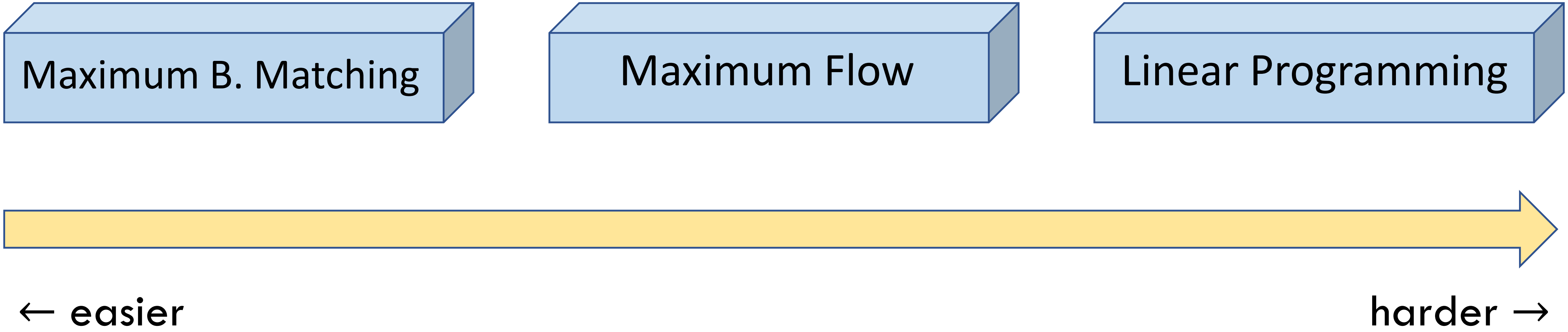
Max Cut: Given a graph G, find a maximum capacity cut. cf. Min Cut

Bisection in Graphs: Find a minimum bisection in G. cf. Bisection in Trees

Integer LP cf. LP

$$\max c^T x$$
$$Ax \leq b$$
$$x_i \in \{0,1\}$$

Reductions


Maximum B. Matching
Maximum Flow
Linear Programming
← easier
harder →

Polynomial-time reduction

Consider two problems X and Y. A polynomial-time reduction from X to Y is an algorithm for solving X that

- makes at most $p(n)$ standard computational operations
- makes at most $q(n)$ black-box calls to an oracle that solves problem Y

where $p(n)$ and $q(n)$ are polynomials, and n is the size of the input.

The program must write the input for the oracle and read the output.

Examples

Here is a reduction from Weighted Bipartite Matching to Linear Programming:

- Input: an instance of Weighted Bipartite Matching
- Write an LP for the instance
- (Oracle Call) Find an optimal vertex solution for the LP x^*
- Let $M = \{e : x_e^* = 1\}$

Reducibility

We write $X \leq_p Y$ if there is a polynomial-reduction from X to Y.

Claim: If $X \leq Y$ and $Y \leq Z$, then $X \leq Z$.

Proof: Let A be a reduction from X to Y, and B be that from Y to Z.

We construct a reduction form X to Z.

- Run A
- Whenever A makes an oracle call to solve an instance of Y, use reduction B.

$$p_C(n) \leq p_A(n) + q_A(n) \cdot p_B\big(p_A(n)\big)$$
$$q_C(n) \leq q_A(n) \cdot q_B\big(p_A(n)\big)$$

Reductions

Maximum B. Matching $\leq_p$ Maximum Flow $\leq_p$ Linear Programming

← easier harder →

Is it also true that $LP \leq_p \text{Maximum Matching}$?

Decision vs Search

It will be more convenient to work with decision problems, in which the answer is either yes or no.

Search problem

Max Flow: find a maximum flow in the flow network.

Decision problem

Decision version of Max Flow: Given G and M, decide if is there a flow of value at least M?

Decision Problems: IS and VC

A subset of vertices A is an independent set for a graph $G = (V, E)$, if no edge connects two vertices in A.

Independent Set Problem: Given a graph G and a parameter M, decide if there is an independent set of size at least M.

A subset of vertices A is a vertex cover for a graph $G = (V, E)$, if for every edge (u, v), at least one of the vertices u and v lies in A

Vertex Cover Problem: Given a graph G and a parameter M, decide if there is a vertex cover of size at most M.

Decision Problems: IS and VC

Observe that A is a vertex cover if and only if $B = V \setminus A$ is an independent set.

A is a VC $\Leftrightarrow$ for every $(u, v) \in E$, $u \in A$ or $v \in A \Leftrightarrow$

for every $(u, v) \in E$, $u \notin B$ or $v \notin B \Leftrightarrow B$ is a IS.

Claim VC $\leq_p$ IS

Proof: Make an oracle call to determine if there is an IS of size $\geq n - M$.

yes: Then, there is a VC of size $\leq M$.

no: Then, there is no VC of size $\leq M$.

Decision Version of Integer LP

Determine if the following program is feasible

$Ax \leq b$

$x_i \in \{0,1\}$ for every i

Decision Version of Integer LP

Claim: IS $\leq_p$ ILP

Proof:

We need to determine if there is an independent set in G of size at least M.

Test if the following ILP is feasible.

- $\sum_{u \in V} x_u \geq M$
- $x_u + x_v \leq 1$ for every edge $(u, v) \in E$
- $x_u \in \{0,1\}$ for all i

Discuss.

We conclude VC $\leq_p$ IS $\leq_p$ ILP.

3-SAT Problem

- We are given a set of Boolean variables $x_1, \ldots, x_n$.
- A literal ℓ_j is either some x_i or its negation $\bar{x}_i$.
- A clause is a disjunction of literals: $\ell_1 \vee \ell_2 \vee \cdots \vee \ell_k$.
- A SAT formula is a conjunction of clauses $c_1 \wedge c_2 \wedge \cdots \wedge c_m$.

- A truth assignment is an assignment of values true and false to x_i-s.
- Literal x_i is satisfied if $x_i = \mathrm{true}$
- Literal $\bar{x}_i$ is satisfied if $x_i = \mathrm{false}$

3-SAT Problem

- A clause is a disjunction of literals: $\ell_1 \vee \ell_2 \vee \cdots \vee \ell_k$.
- A SAT formula is a conjunction of clauses $c_1 \wedge c_2 \wedge \cdots \wedge c_m$.

- Clause $\ell_1 \vee \cdots \vee \ell_k$ is satisfied by an assignment if at least one of the literals $\ell_1, \ldots, \ell_k$ is satisfied.
- Formula $c_1 \wedge \cdots \wedge c_m$ is satisfied by an assignment if all clauses $c_1, \ldots, c_k$ are satisfied.

A SAT formula $\varphi = c_1 \wedge \cdots \wedge c_m$ is satisfiable if there exists an assignment that satisfies it.

3-SAT Problem

SAT Problem: Given a SAT formula φ, decide if it is satisfiable.

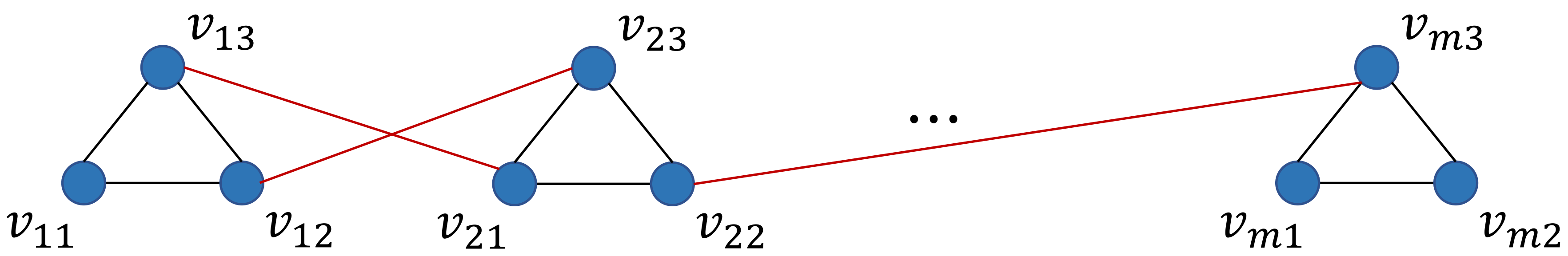
3-SAT Problem: Given a SAT formula φ in which every clause contains exactly 3 literals, decide if it is satisfiable.

Gadget Reductions

Theorem 3-SAT $\leq_p$ IS

Proof: Let $\varphi = c_1 \wedge \cdots \wedge c_m$ and $c_i = \ell_{i1} \vee \ell_{i2} \vee \ell_{i3}$ for every i.

Consider a graph G on vertices $\{v_{ij}: 1 \leq i \leq m \text{ and } 1 \leq j \leq 3\}$.



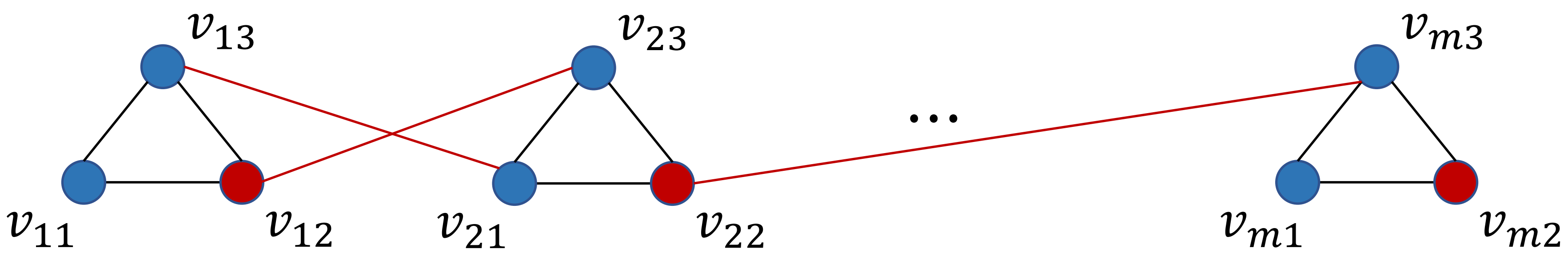
Connect vertices:

in each "cloud" $\{v_{i1}, v_{i2}, v_{i3}\}$

v_{ab} and v_{cd} if ℓ_{ab} is the negation of ℓ_{cd} (one is x_k and the other is $\bar{x}_k$)

Graph G: Connect v_{i1}, v_{i2}, v_{i3}. Connect v_{ab} and v_{cd} if ℓ_{ab} is the negation of ℓ_{cd}.

Claim: φ is satisfiable iff G has an independent set of size at least m.



Assume that φ is satisfiable. Consider a satisfying assignment.

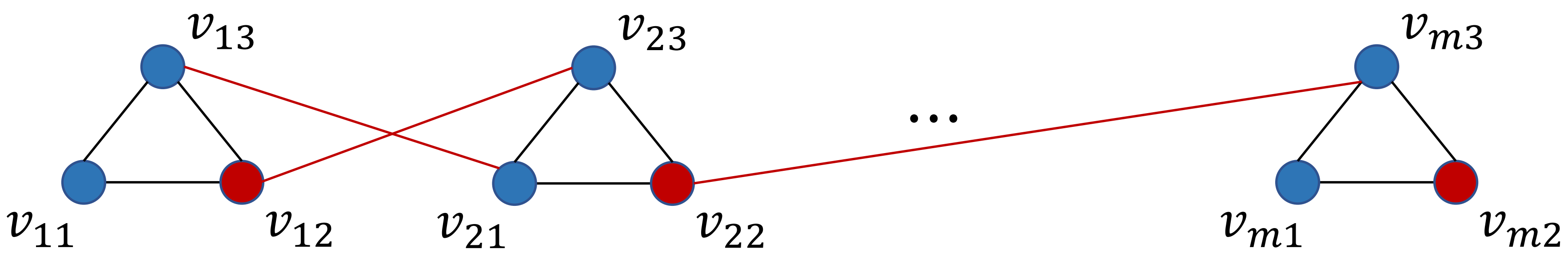
Choose a satisfied literal ℓ_{it_i} in every clause c_i.

Observe that $I = \{v_{it_i}: 1 \leq i \leq m\}$ is an independent set.

- only 1 vertex in each cloud
- Can it happen that $v_{ab}, v_{cd} \in I$ and $(v_{ab}, v_{cd}) \in E$?

Graph G: Connect v_{i1}, v_{i2}, v_{i3}. Connect v_{ab} and v_{cd} if ℓ_{ab} is the negation of ℓ_{cd}.

Claim: φ is satisfiable iff G has an independent set of size at least m.



Q: Can it happen that $v_{ab}, v_{cd} \in I$ and $(v_{ab}, v_{cd}) \in E$?

No! If $v_{ab}, v_{cd} \in I$, then

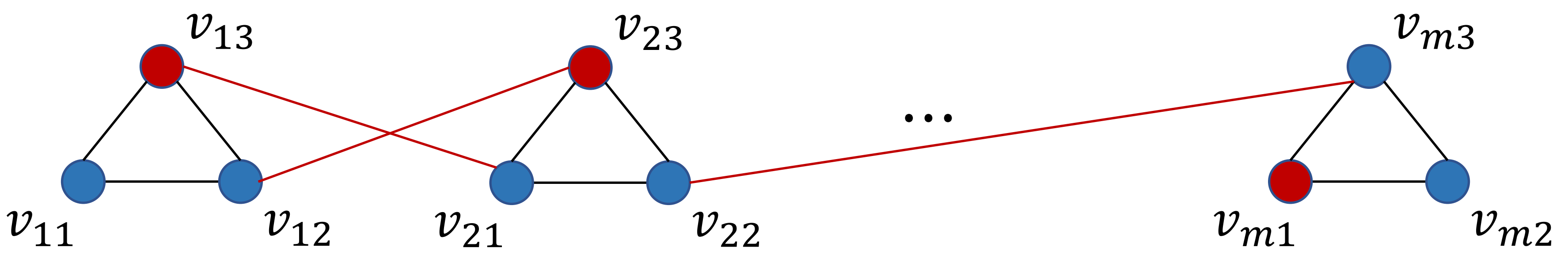
ℓ_{ab} and ℓ_{cd} are satisfied by the assignment, and

thus ℓ_{ab} and ℓ_{cd} cannot be negations of each other.

The size of I is m, as required.

Graph G: Connect v_{i1}, v_{i2}, v_{i3}. Connect v_{ab} and v_{cd} if ℓ_{ab} is the negation of ℓ_{cd}.

Gadget Reductions



Assume that there is an independent set I of size at least m.

Then for every cloud i, there is exactly one vertex v_{it_i} from the cloud in I. **Why?**

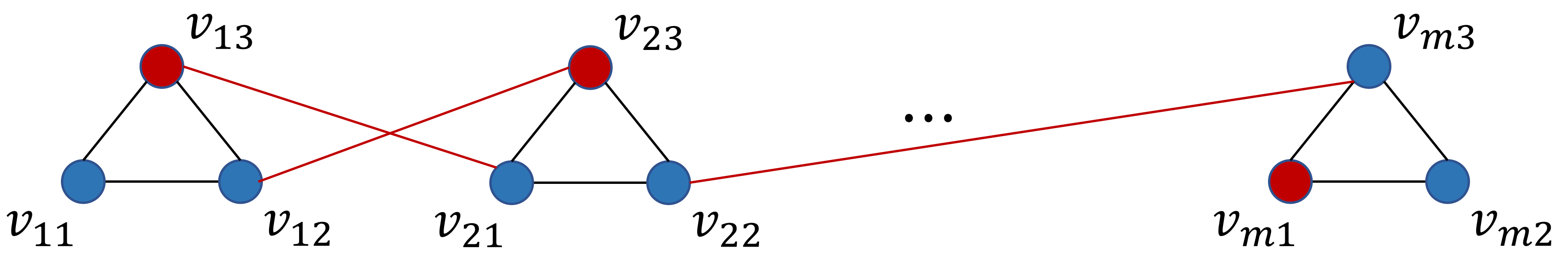
For every i do the following.

If $\ell_{it_i} = x_j$, then $x_j = \text{true}$
If $\ell_{it_i} = \bar{x}_j$, then $x_j = \text{false}$

Assign arbitrary values to unassigned variables.

Graph G: Connect v_{i1}, v_{i2}, v_{i3}. Connect v_{ab} and v_{cd} if ℓ_{ab} is the negation of ℓ_{cd}.

Gadget Reductions



If $\ell_{at_a} = x_j$, then $x_j = \text{true}$
If $\ell_{ct_c} = \bar{x}_j$, then $x_j = \text{false}$

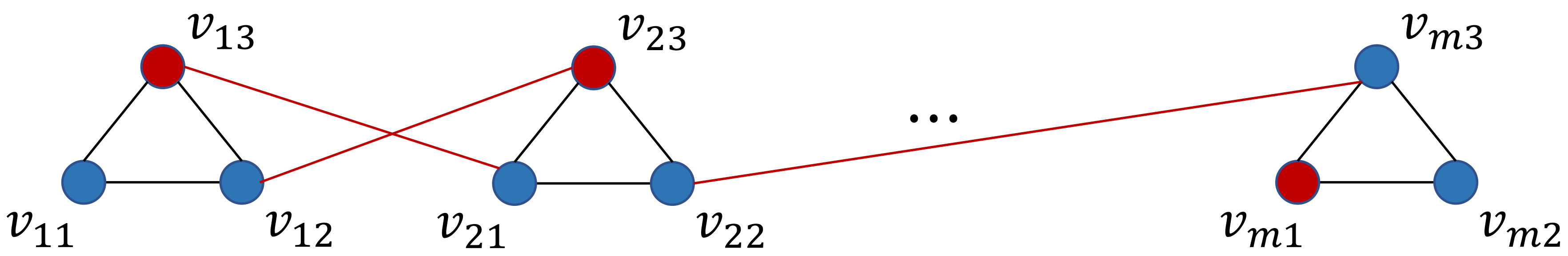
Q: Can we assign both true and false to x_j?

A: No, because ℓ_{ct_c} is the negation of ℓ_{at_a}. Both literals cannot be in I.

Q: Why does the obtained assignment satisfy φ?

A:

Gadget Reductions



Reduction from 3-SAT to IS:

- Construct the graph G
- Make an oracle call that return if there is an IS of size at least m in G

yes: the formula is satisfiable

no: the formula is not satisfiable

Classes P and NP

Class P: A problem M is in P if there is a polynomial algorithm for M.

Decision versions of problems we studied earlier in this course are in P.

Class NP: A problem M is in *NP* if there is a polynomial-time algorithm V that gets as input an instance x of M and a witness/certificate w such that

(completeness) If x is a yes-instance, then there exists $w \in \{0,1\}^{\mathrm{poly}(n)}$ s.t. $V(x, w) = yes$

(soundness) If x is a no-instance, then $V(x, w) = no$ for every $w \in \{0,1\}^{poly(n)}$.

NP = non-deterministic polynomial-time

Classes P and NP

(completeness) If x is a yes-instance, then there exists $w \in \{0,1\}^{\text{poly}(n)}$ s.t. $V(x,w) = yes$

There is a witness for yes-instances.

(soundness) If x is a no-instance, then $V(x,w) = no$ for every $w \in \{0,1\}^{poly(n)}$.

There is no witness for no-instances.

equivalently, if there is a witness for an instance,

then it is a yes-instance.

NP = non-deterministic polynomial-time

Example: ILP is in NP

Given an instance

$$Ax \leq b$$
$$x \in \{0,1\}^n$$

and a witness w, algorithm V verifies whether $x = w$ is a feasible solution.

If it is, V outputs yes; otherwise, no.

Soundness: if the algorithm return yes, then $x = w$ is a feasible solution for the ILP. Thus, the ILP is feasible.

Completeness: if the ILP is feasible, it has a feasible solution x^*.
Then V accepts witness $w = x^*$.

Questions

Q: Is it true that $P \subseteq NP$?

Q: Is it true that

VC, IS, 3-SAT, SAT $\in$ NP ?

Open Problem: Is it true that $P \neq NP$?

- It's widely believed that $P \neq NP$.
- This is a major open problem in CS since 1970-s.

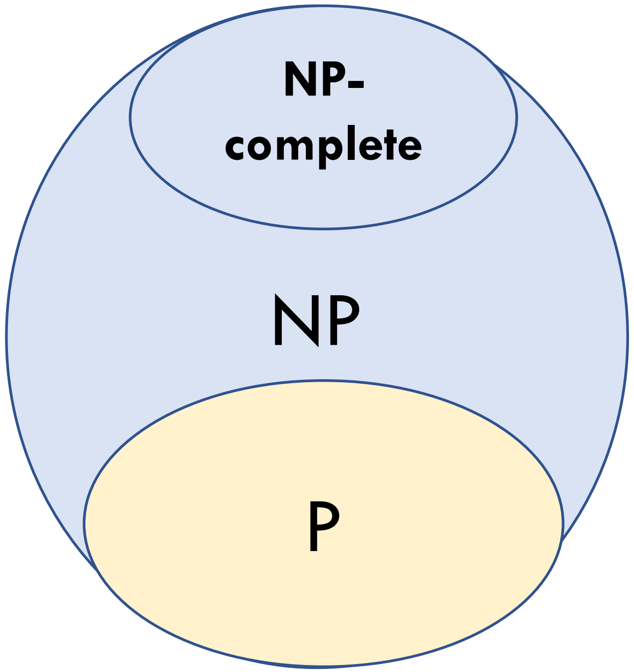
NP-completeness

NP-hardness: A problem X is NP-hard if

$X \geq_p Y$ for every problem $Y \in NP$

NP-completeness: A problem X is NP-complete if

- $X \in NP$
- $X \geq_p Y$ for every problem $Y \in NP$.



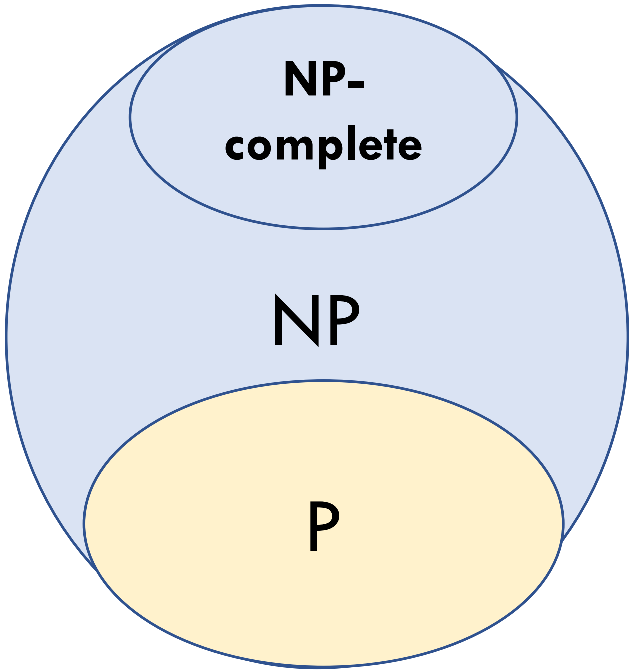
NP-completeness

NP-completeness: A problem X is NP-complete if

- $X \in NP$
- $X \geq_p Y$ for every problem $Y \in NP$.

If X is NP-complete and $Z \geq_p X$ then Z is also NP-complete.

Proof: $Z \geq X \geq Y$ for $Y \in NP$.



NP-completeness

Claim: Assume that $P \neq NP$.

If X is NP-hard or NP-complete, then X cannot be solved in polynomial time.

Proof: Assume to the contrary that there is a polynomial algorithm A for X.

Since X is NP-hard, for every problem $Y \in NP$, there is a poly-time reduction from Y to X.

Recall that the reduction makes oracle queries to solve problem X. Use A to answer these queries. We get a poly-time algorithm for Y.

Thus, $Y \in P$. Therefore, $NP \subseteq P$. We get a contradiction.

NP-completeness

We assume from now on that $P \neq NP$.

Good news:

- To prove that there is no polynomial-time algorithm for X, it is sufficient to prove that $X \geq_p Y$ for some NP-complete problem Y.
- Constructing reductions is not that difficult (we did it today!).

Bad news:

- We need to have at least one NP-complete problem.

NP-completeness

Cook-Levin Theorem: 3-SAT is NP-complete.

Proof: Please read the textbook if you are interested!

Corollary:

- Independent Set, Integer Linear Programming, Vertex Cover are NP-hard.
- Thus, there are no polynomial-time algorithms for them, assuming $P \neq NP$.

Thousands on problems are known to be NP-complete, including Minimum Graph Bisection, Knapsack, Maximum Cut, Graph Coloring, Sparsest Cut, Min Balanced Cut, Multiway Cut, Hamiltonian Path.

Polynomial-time Reductions

$\leq_p$ is a Cook reduction or polynomial-time Turing reduction

Usually, Karp or polynomial-time many-to-one reduction $\leq_m$ is used.

$A \leq_m B$ if there is a polynomial-time computable function $g(x)$ such that $x \in A$ if and only if $g(x) \in B$.

Reduction:

- compute $g(x)$
- ask the oracle if $g(x) \in B$ (*single call to the oracle!*)
- return the answer

Karp Reductions

All reductions we considered today were Karp reductions.

Usually NP-complete and NP-hard problems are defined with respect to Karp reductions.