CS 341 – Algorithms

Lecture 17 – Polynomial Time Reductions

21 July 2021

Today’s Plan

1. Polynomial Time Reductions
2. Simple Reductions
3. More Simple Reductions
4. A Non-Trivial Reduction

Polynomial Time Reductions

Once we have learned more and more algorithms, they become our building blocks and we may not need to design algorithms for new problems from scratch.

So it becomes more and more important to be able to use existing algorithms to solve new problems.

We have already seen a few reductions.

For examples, we have reduced subset-sums to knapsack, longest increasing subsequence to longest common subsequences, and basketball league winner to maximum bipartite matching, etc.

In general, if there is an efficient reduction from problem A to problem B and there is an efficient algorithm to solve problem B, then we have an efficient algorithm to solve problem A.

Decision Problems

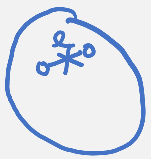
To formalize the notion of a reduction, it is more convenient to restrict our attention to <u>decision problems</u>, for which the output is either YES or NO, so that every problem has the same output format.

For example, instead of finding a maximum matching, we consider the decision version of the problem "Does the input graph G have a matching of size at least k?".

As we will discuss later, for all the problems that we will consider, if we know how to solve the decision version of our problem in polynomial time, then we can use the decision algorithm as a blackbox/subroutine to solve the search version of our problem in polynomial time.

e.g. max bipartite matching.

- binary search on k, to find optimal value k
- G

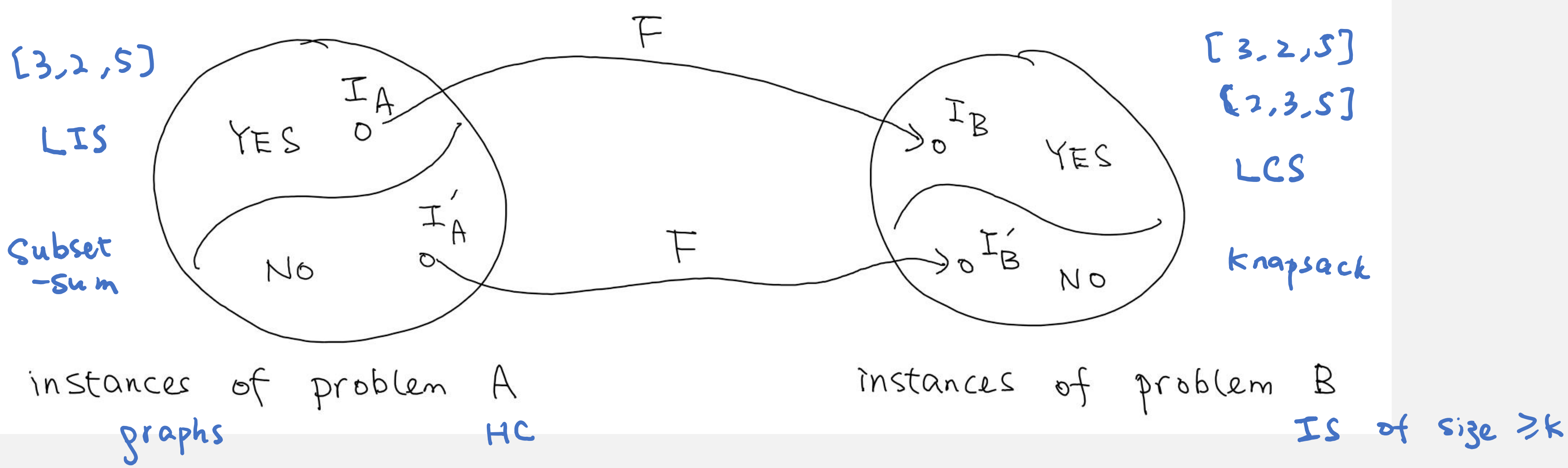


Does $G-e$ have a matching of size at least k? → YES, delete e. → NO, keep e.

⇒ reduce to m decision problems.

Definition (Polynomial Time Reductions)

Definition. We say a decision problem A is polynomial time reducible to a decision problem B if there is a polynomial time algorithm F that maps/transforms any instance I_A of A into an instance I_B of B (that is, $F(I_A) = I_B$) such that I_A is a YES-instance of problem A if and only if I_B is a YES-instance of problem B.



We use the notation $A \leq_p B$ to denote that such a polynomial time reduction exists, intuitively saying that problem A is not more difficult than problem B in terms of polynomial time solvability.

Algorithm (Solving Problem A by Reduction)

Input: an instance I_A of problem A

Output: whether I_A is a YES-instance

1. Use the reduction algorithm F to map/transform I_A into $I_B = F(I_A)$ of problem B.

2. Return $ALG_B(I_B)$.

Correctness:

1. I_A YES $\Leftrightarrow$ $I_B = F(I_A)$ YES (reduction)
2. $ALG_B(I_B)$ is correct

time complexity: F maps I_A into I_B in $p(|I_A|)$

ALG_B solves I_B in $q(|I_B|)$, then algo solves A in $q(p(|I_A|))$

Proving Hardness Using Reductions

We showed $A \leq_p B$ and use an efficient algorithm for problem B to solve problem A.

This is the usual direction, but now we explore the other implication of the inequality $A \leq_p B$.

Suppose problem A is known to be impossible to be solved in polynomial time.

Then $A \leq_p B$ implies that problem B cannot be solved in polynomial time either.

Therefore, if A is computationally hard and $A \leq_p B$, then B is also computationally hard.

By our current knowledge, however, we know almost nothing about proving a problem cannot be solved in polynomial time, so we could not draw such a strong conclusion from $A \leq_p B$.

$A \leq_p B$

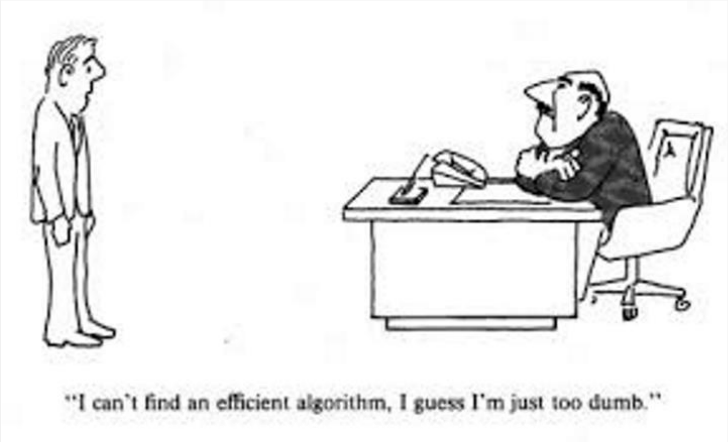
easy $\longleftarrow$ easy

$A \leq_p B$

hard $\longrightarrow$ hard

Cartoon (from book by Garey and Johnson)

Suppose our boss gives us a problem C, but we don't know how to solve it in polynomial time.



It would be much more convincing if you could prove e.g. TSP $\leq_p C$.

Dream

Today’s Plan

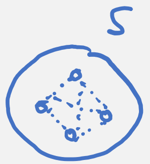
1. Polynomial Time Reductions
2. Simple Reductions
3. More Simple Reductions
4. A Non-Trivial Reduction

Three Problems

Maximum Clique (Clique): A subset $S \subseteq V$ is a clique if $uv \in E$ for all $u, v \in S$.

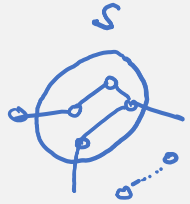
Input: A graph $G = (V, E)$, an integer k.

Output: Is there a clique with at least k vertices?



Maximum Independent Set (IS): A subset $S \subseteq V$ is an independent set if $uv \notin E$ for all $u, v \in S$.

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



Output: Is there an independent set with at least k vertices?

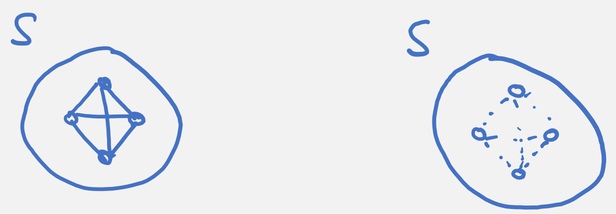
Minimum Vertex Cover (VC): A subset $S \subseteq V$ is a vertex cover if $\{u, v\} \cap S \neq \emptyset$ for all $uv \in E$.

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

Output: Is there a vertex cover with at most k vertices?

Cliques and Independent Sets

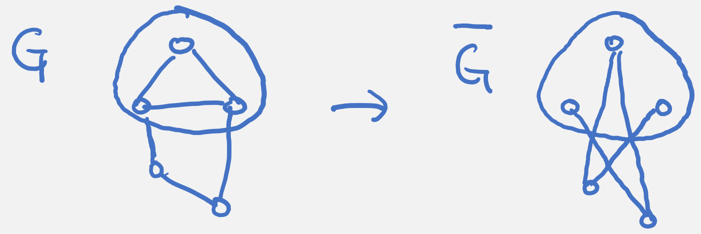
Proposition. Clique $\leq_p$ IS and IS $\leq_p$ Clique.



Proof Clique $\leq_p$ IS.

Reduction Algorithm F: (G, k) Clique

$F((G,k)) \longrightarrow (\bar{G}, k)$



$G = (V, E)$ $\quad \bar{G} = (V, \bar{E})$ where $\bar{E} = \{uv \mid uv \notin E\}$

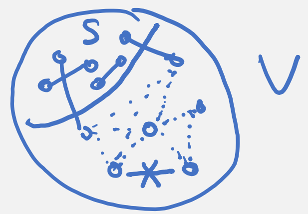
Claim: G has a clique of size at least k $\Leftrightarrow$
$\bar{G}$ has an independent set of size at least k.

Clique $\Leftrightarrow_p$ IS

$\square$

Independent Sets and Vertex Cover

Observation. In $G=(V,E)$, a subset $S\subseteq V$ is a vertex cover if and only if $V-S$ is an independent set.



Proposition. VC $\leq_p$ IS and IS $\leq_p$VC.

proof $\quad$ VC $\leq_p$ IS

reduction algorithm F: $\quad (G,k)$ for VC

$$F((G,k)) \to (G, n-k) \qquad n = |V(G)|$$

Claim $\quad$ G has a vertex cover of size $\leq k$

$\Leftrightarrow$ G has an independent set of size $\geq n-k$. $\quad \square$

Polynomial Time Reductions are Transitive

Proposition. If $A \leq_p B$ and $B \leq_p C$, then $A \leq_p C$.

proof

$A \leq_p B$: $\exists F$ $\quad I_A$ YES $\Leftrightarrow$ $I_B = F(I_A)$ YES

F can be done in $p()$ time

$B \leq_p C$: $\exists H$ $\quad I_B$ YES $\Leftrightarrow$ $I_C = H(I_B)$ YES

H can be done in $q()$ time

$A \leq_p C$: $H(F(I_A)) = I_C$ $\quad I_A$ YES $\Leftrightarrow$ I_C YES

$H(F())$ can be done $q(p())$. $\square$

Therefore, Clique, IS, and VC are all equivalent in terms of polynomial time solvability.

Today's Plan

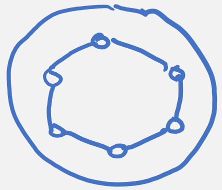
1. Polynomial Time Reductions
2. Simple Reductions
3. More Simple Reductions
4. A Non-Trivial Reduction

Hamiltonian Problems

Hamiltonian Cycle (HC): A cycle is a Hamiltonian cycle if it touches every vertex exactly once.

Input: A graph $G = (V, E)$.

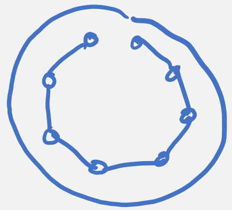
Output: Does G have a Hamiltonian cycle?



Hamiltonian Path (HP): A path is a Hamiltonian path if it touches every vertex exactly once.

Input: A graph $G = (V, E)$.

Output: Does G have a Hamiltonian path?



Traveling Salesman Problem (TSP):

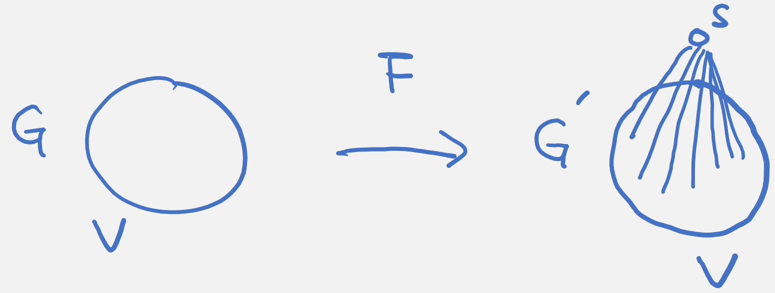
Input: A graph $G = (V, E)$, with an edge length l_e for each edge $e \in E$, and an integer L.

Output: Is there a Hamiltonian cycle with total length at most L?

HP $\le_p$ HC

Proposition. HP $\le_p$ HC.

proof



reduction F : $G = (V, E)$ for HP

map into $G' = (V + s,\ E + \{sv \mid v \in V\})$

Claim : G has a HP $\Longleftrightarrow$ G' has a HC

$\Rightarrow$

$\Rightarrow$

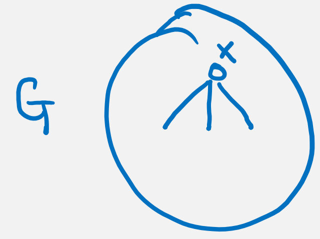
□

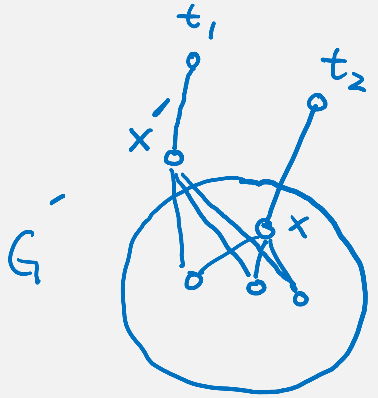
□

HC $\leq_p$ HP

Proposition. HC $\leq_p$ HP.

proof idea: "open up" the cycle

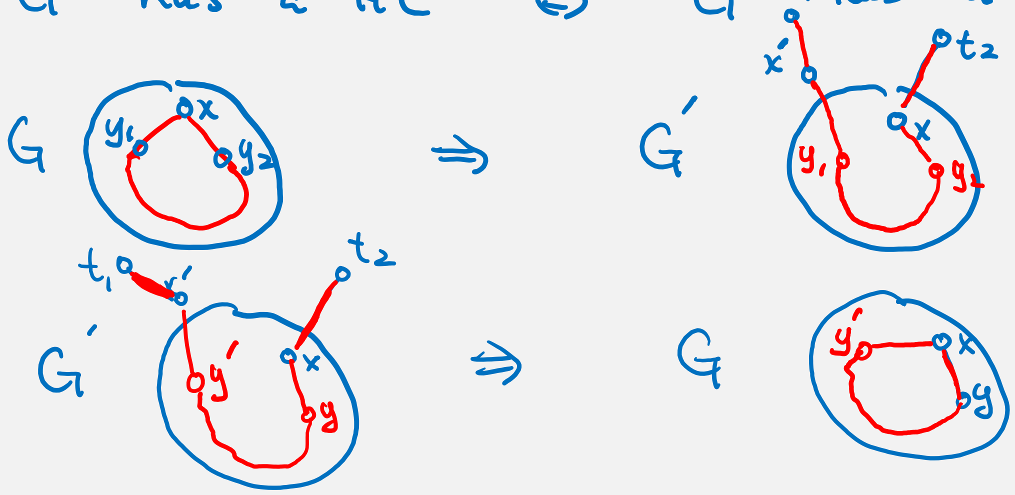




reduction: $G = (V, E)$ for HC

maps into $G' = (V + x' + t_1 + t_2,\ E + \{x'v \mid v \in N(x)\} + t_1 x' + t_2 x)$

Claim: G has a HC $\Leftrightarrow$ G' has a HP



Traveling Salesman Problem

A common technique in doing reduction is to show that one problem is a special case of another problem.

We call this technique <u>specialization</u>.

<u>Proposition</u>. HC $\leq_p$ TSP.

<u>proof</u>

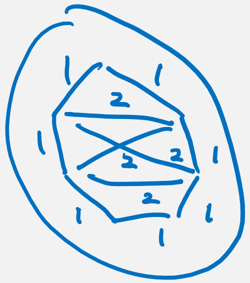
$G = (V, E)$ for HC $\xrightarrow{F}$ $G' = (V, \cdot)$ for TSP

complete graph

$uv \in E \to l_{uv} = 1$

$uv \notin E \to l_{uv} = 2$

$\longrightarrow$



<u>Claim</u> G has a HC $\iff$ G' has a HC of total cost/length $\leq n$.

$\square$

$\square$

Today’s Plan

1. Polynomial Time Reductions
2. Simple Reductions
3. More Simple Reductions
4. A Non-Trivial Reduction

3-Satisfiability (3SAT)

This is an important problem in the theory of NP-completeness.

We are given n Boolean variables $x_1, x_2, \dots, x_n$, each can be set to be either True or False.

We are also given a formula in conjunctive normal form (CNF), where it is an AND of the clauses,

where each clause is an OR of the literals, where a literal is either x_i or $\overline{x_i}$.

literal

$\underbrace{(x_1 \vee x_2 \vee \overline{x_3})}_{\text{clause}} \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee x_4) \wedge (x_3 \vee \overline{x_4})$

e.g. $x_1 = T \quad x_2 = F \quad x_3 = T \quad x_4 = T/F$

$(x_1 \vee x_2) \wedge (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2})$ $\Leftarrow$ not satisfiable

3-Satisfiability (3SAT)

Input: A CNF-formula in which each clause has at most three literals.

Output: Is there a truth assignment to the variables that satisfies all the clauses?

3SAT $\leq_p$ IS

Theorem. 3SAT $\leq_p$ IS.

Proof. Given a 3SAT formula, we'd like to construct a graph G so that the formula is satisfiable

if and only if the graph has an independent set of size m where m is the number of clauses.

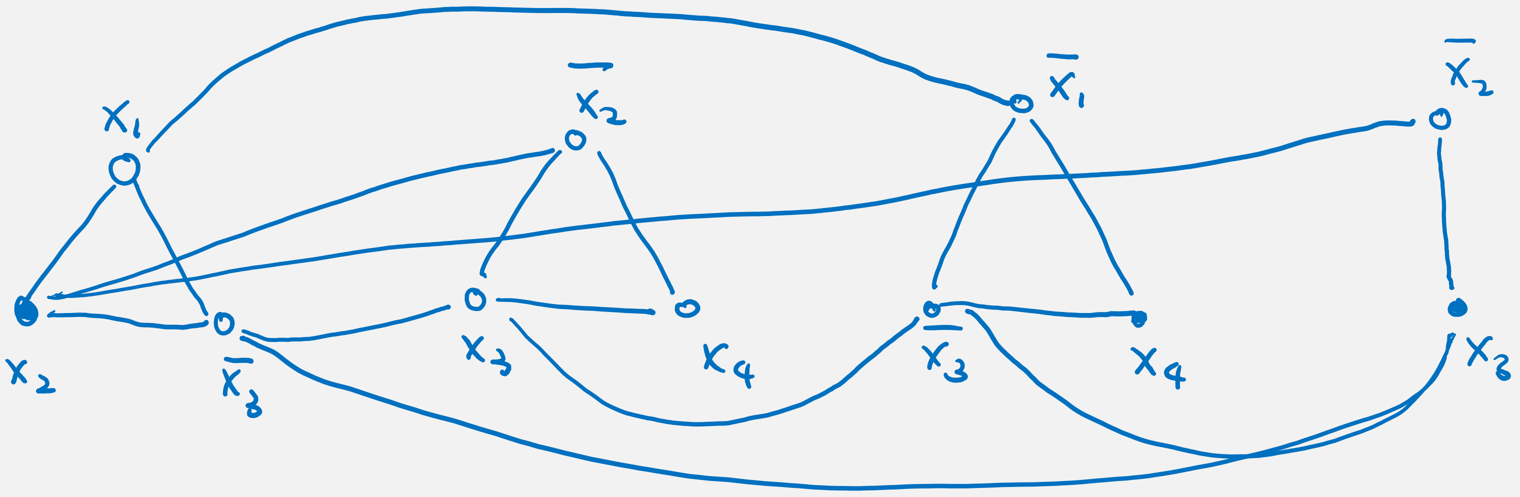
Idea: We would like the independent set to tell us how to satisfy the formula.

Reduction: $(x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3)$

Claim

formula is satisfiable

$\Leftrightarrow$ graph has an IS of size $\geq m$.



3SAT $\leq_p$ IS

Theorem. 3SAT $\leq_p$ IS.

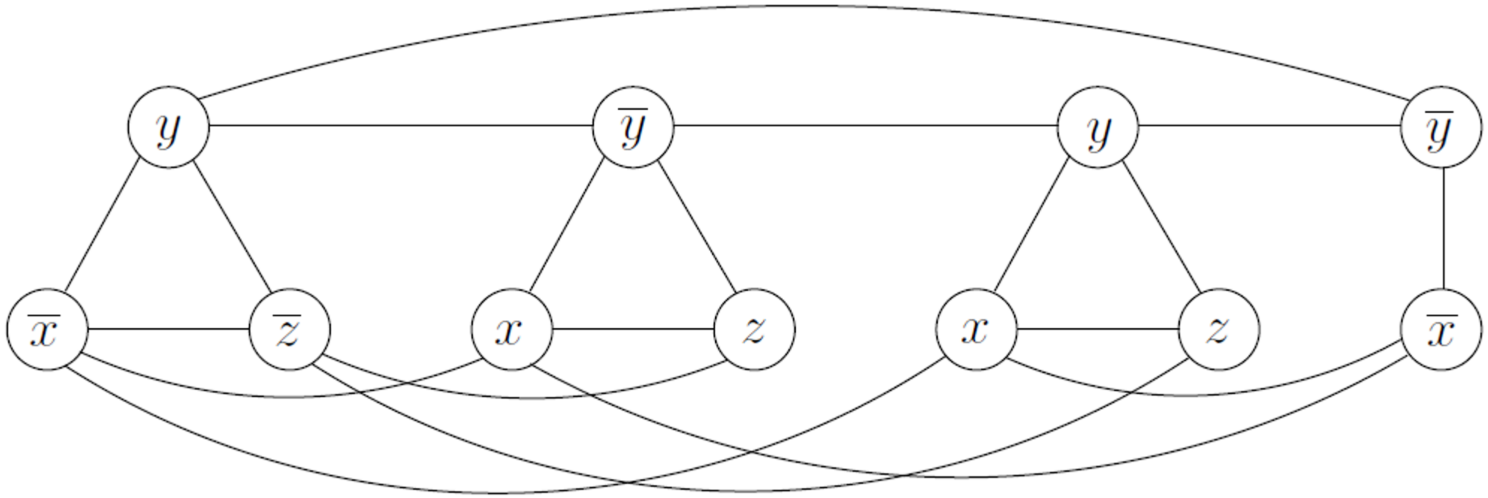
Proof. Given a 3SAT formula, we'd like to construct a graph G so that the formula is satisfiable if and only if the graph has an independent set of size m where m is the number of clauses.

Idea: We would like the independent set to tell us how to satisfy the formula.

Figure 8.8 The graph corresponding to $(\overline{x} \vee y \vee \overline{z})\ (x \vee \overline{y} \vee z)\ (x \vee y \vee z)\ (\overline{x} \vee \overline{y})$.

first type of edges within clauses

second type of types $x_i \leftarrow \bar{x}_i$

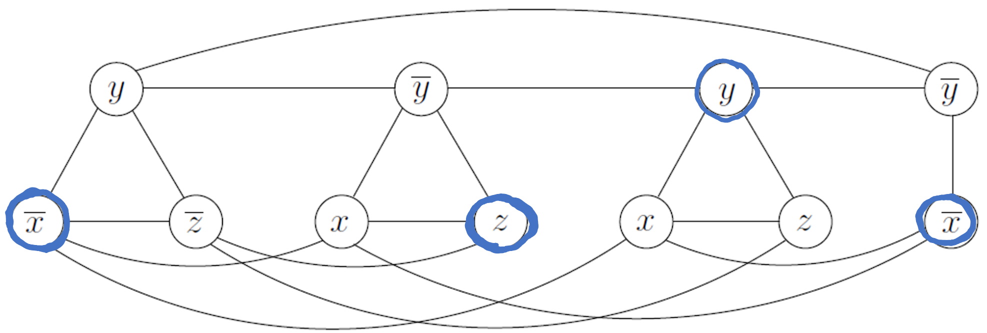


Proof

Lemma. The formula is satisfiable iff

there is an independent set of size m.

Figure 8.8 The graph corresponding to $(\overline{x} \vee y \vee \overline{z})\ (x \vee \overline{y} \vee z)\ (x \vee y \vee z)\ (\overline{x} \vee \overline{y})$.



$x = F \quad y = T \quad z = T$

Proof $\Rightarrow$ given a satisfying assignment

$x_i = T$ only put x_i in IS

$x_i = F$ only put $\overline{x_i}$ in IS

Satisfying assignment $\Rightarrow$ each clause has a true literal

choose one and put it in IS

① pick exactly one literal in each clause, no problem with first type

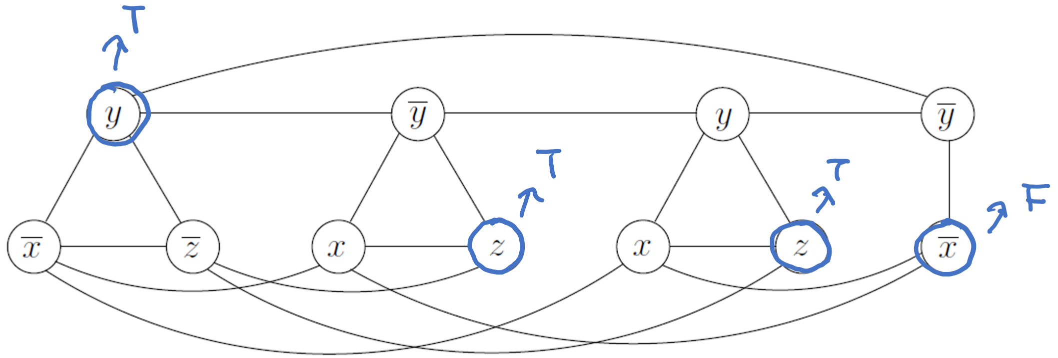
② truth assignment $\Rightarrow$ no problem with second type

① + ② $\Rightarrow$ IS of size m. □

Proof

Lemma. The formula is satisfiable iff there is an independent set of size m.

Figure 8.8 The graph corresponding to $(\overline{x} \vee y \vee \overline{z})\ (x \vee \overline{y} \vee z)\ (x \vee y \vee z)\ (\overline{x} \vee \overline{y})$.



proof. $\Leftarrow$ given an IS of size m

$\because$ second type, we won't put both x_i and $\bar{x}_i$ in IS

if x_i in IS, then $x_i = T$

$\bar{x}_i$ in IS, then $x_i = F$

x_i and $\bar{x}_i$ not in IS, then $x_i = T$ or F

$\because$ first type, each clause can pick ≤ 1 vertex in IS

m clauses, IS of size m $\Rightarrow$ each clause $= 1$ vertex in IS

$\Rightarrow$ all clauses satisfied, satisfying assignment. $\square$

Concluding Remarks

We have introduced the notion of a polynomial time reduction, and use it to establish connections between different problems, and so far we have

$A \leftarrow B$

$A \le_p B$

3SAT

IS

Clique VC

HC

HP TSP

In principle, we can add new problems to relate to these problems, and slowly build up a big web of all computational problems.

As $\leq_p$ is transitive, any strongly connected components of this big graph forms an equivalent class of problems in terms of polynomial time solvability.

Is there a better way to do it than to consider the problems one by one?