Analysis of Algorithms

V. Adamchik

CSCI 570

Lecture 11

University of Southern California

Spring 2023

Linear Programming

NP-Completeness

Reading: chapter 8 and 9

Linear Programming

We say that a maximization linear program with n variables is in standard form if for every variable x_k we have the inequality $x_k \geq 0$ and other m linear inequalities:

$$\max\ (c^T x)$$

subject to

$$A x \leq b$$
$$x \geq 0$$

The LP can be solved in polynomial time for real variables x_k.
In case of integer variables, we do not have a polynomial solver.

Discussion Problem 1

Write a 0-1 Knapsack Problem as a linear program.

Given n items with weights $w_1, w_2, \ldots, w_n$ and values $v_1, v_2, \ldots, v_n$. Put these items in a knapsack of capacity W to get the maximum total value in the knapsack.

$$\text{Given } \sum_{k=1}^{m} w_k \leq W$$

$$\text{optimize } \sum_{k=1}^{m} v_k \to \max$$

Knapsack as LP

Dual LP

To every linear program there is a dual linear program

Duality

<u>Definition.</u> The dual of the standard (primal) maximum problem

$\max_x c^T x$
$Ax \le b$ and $x \ge 0$

is defined to be the standard minimum problem

$\min_y b^T y$
$A^T y \ge c$ and $y \ge 0$

Exercise: duality

Consider the LP:

$$\max(7x_1 - x_2 + 5x_3)$$
$$x_1 + x_2 + 4x_3 \leq 8$$
$$3x_1 - x_2 + 2x_3 \leq 3$$
$$2x_1 + 5x_2 - x_3 \leq -7$$
$$x_1, x_2, x_3 \geq 0$$

Write the dual problem.

$$\boxed{\begin{array}{l} \max(c^T x) \\ \\ A x \leq b \\ x \geq 0 \end{array}} \quad \longleftrightarrow \quad \boxed{\begin{array}{l} \min(b^T y) \\ \\ A^T y \geq c \\ y \geq 0 \end{array}}$$

primal LP

dual LP

From Primal to Dual

Consider the max LP constrains

$$a_{11}x_1+\ldots+a_{1n}x_n \le b_1$$

$$\vdots$$

$$a_{m1}x_1+\ldots+a_{mn}x_n \le b_m$$

1) Multiply each equation by a new variable $y_k \ge 0$.
2) Add up those m equations.
3) Collect terms wrt to x_k.
4) Choose y_k in a way such that $A^T y \ge c$.

Weak Duality

$$\boxed{\begin{array}{l}\max(c^T x)\\ \\ A x \le b\\ x \ge 0\end{array}} \quad \longleftrightarrow \quad \boxed{\begin{array}{l}\min(b^T y)\\ \\ A^T y \ge c\\ y \ge 0\end{array}}$$

primal linear program

dual linear program

Weak Duality. The optimum of the dual is an upper bound to the optimum of the primal.

opt(primal) ≤ opt(dual)

Weak Duality

$$\boxed{\begin{array}{l} \max\,(c^T x) \\ \\ A x \le b \\ x \ge 0 \end{array}} \quad \Longleftrightarrow \quad \boxed{\begin{array}{l} \min\,(b^T y) \\ \\ A^T y \ge c \\ y \ge 0 \end{array}}$$

Theorem (The weak duality).
Let P and D be primal and dual LP correspondingly.
If x is a feasible solution for P and y is a feasible solution for D, then $c^T x \le b^T y$.

Proof (in matrix form).

Weak Duality: opt(primal) $\leq$ opt(dual)

Corollary 1. If a standard problem and its dual are both feasible, then both are feasible bounded.

Corollary 2. If one problem has an unbounded solution, then the dual of that problem is infeasible.

Strong Duality

$$\boxed{\begin{array}{l} \max(c^T x) \\ \\ A x \le b \\ x \ge 0 \end{array}} \quad \Longleftrightarrow \quad \boxed{\begin{array}{l} \min(b^T y) \\ \\ A^T y \ge c \\ y \ge 0 \end{array}}$$

Theorem (The strong duality).
Let P and D be primal and dual LP correspondingly.
If P and D are feasible, then $c^T x = b^T y$.

The proof of this theorem is beyond the scope of this course.

Possibilities for the Feasibility

$\max(c^T x)$

$A x \leq b$
$x \geq 0$

$\min(b^T y)$

$A^T y \geq c$
$y \geq 0$

P\D	F.B.	F.U.	I.
F.B.			
F.U.			
I.			

feasible bounded - F.B.
feasible unbounded - F.U.
infeasible – I.

Finding the Dual in Equality Form

$$\boxed{\begin{array}{l} \max(c^T x) \\ \\ A\,x = b \\ x \geq 0 \end{array}} \quad \Longleftrightarrow \quad \boxed{\phantom{\begin{array}{l} \max(c^T x) \\ \\ A\,x = b \\ x \geq 0 \end{array}}}$$

Very often linear programs are encountered in equality form $A\,x = b$. A problem can be transformed into inequality form by replacing each equation by two inequalities.

$$\begin{array}{l} A\,x \leq b \\ -A\,x \leq -b \end{array}$$

The dual can then be found by applying the definition of the dual to this problem. Let y^+ and y^- be the dual variables associated with each of the above inequality.

Finding the Dual in Equality Form

$$\max\ (c^T x)$$
$$A x = b$$
$$x \ge 0$$

$$\max\ (c^T x)$$
$$A x \le b$$
$$-A x \le -b$$
$$x \ge 0$$

$$\Longleftrightarrow$$

$$\min\ (b^T y^+ - b^T y^-)$$
$$A^T y^+ - A^T y^- \ge c$$
$$y^+ \ge 0,\ y^- \ge 0$$

$$\min\ (b^T (y^+ - y^-))$$
$$A^T (y^+ - y^-) \ge c$$
$$y^+ \ge 0,\ y^- \ge 0$$

Nonlinear Convex Optimization

$\min f(x_1, x_2, \ldots, x_n)$

$h_i(x_1, x_2, \ldots, x_n) \leq 0, i = 1,, \ldots, m.$

$x_k \geq 0, k = 1, , \ldots, n.$

Here f and/or h are nonlinear functions.

The problem is solved using Lagrange multipliers λ_k.

Lagrange Duality (KKT-1951)

Primal in x:

$$\boxed{\begin{array}{l} \quad \min f(x) \\ \text{subject to} \\ \quad h_k(x) \le 0 \end{array}}$$

Dual in λ:

$$\boxed{\begin{array}{l} \quad \max g(\lambda) \\ \text{subject to} \\ \quad \lambda_k \ge 0 \end{array}}$$

The Lagrangian: $L(x,\lambda) = f(x) + \sum_k \lambda_k h_k(x)$

The dual: $g(\lambda) = \min_x L(x,\lambda)$

Weak Duality:
Let P and D be the optimum of primal and dual problems respectively. Then opt(P) ≤ opt(D).
Equality (strong duality) holds for convex functions under some conditions.

NP-Completeness

In 1935 Alan Turing described a model of computation, known today as the Turing Machine (TM).

Alan Turing
(1936, age 22)

A problem P is *computable* (or *decidable*) if it can be solved by a Turing machine that halts on every input.

We say that P has an algorithm.

Turing Machines were adopted in the 1960's, years after his death.

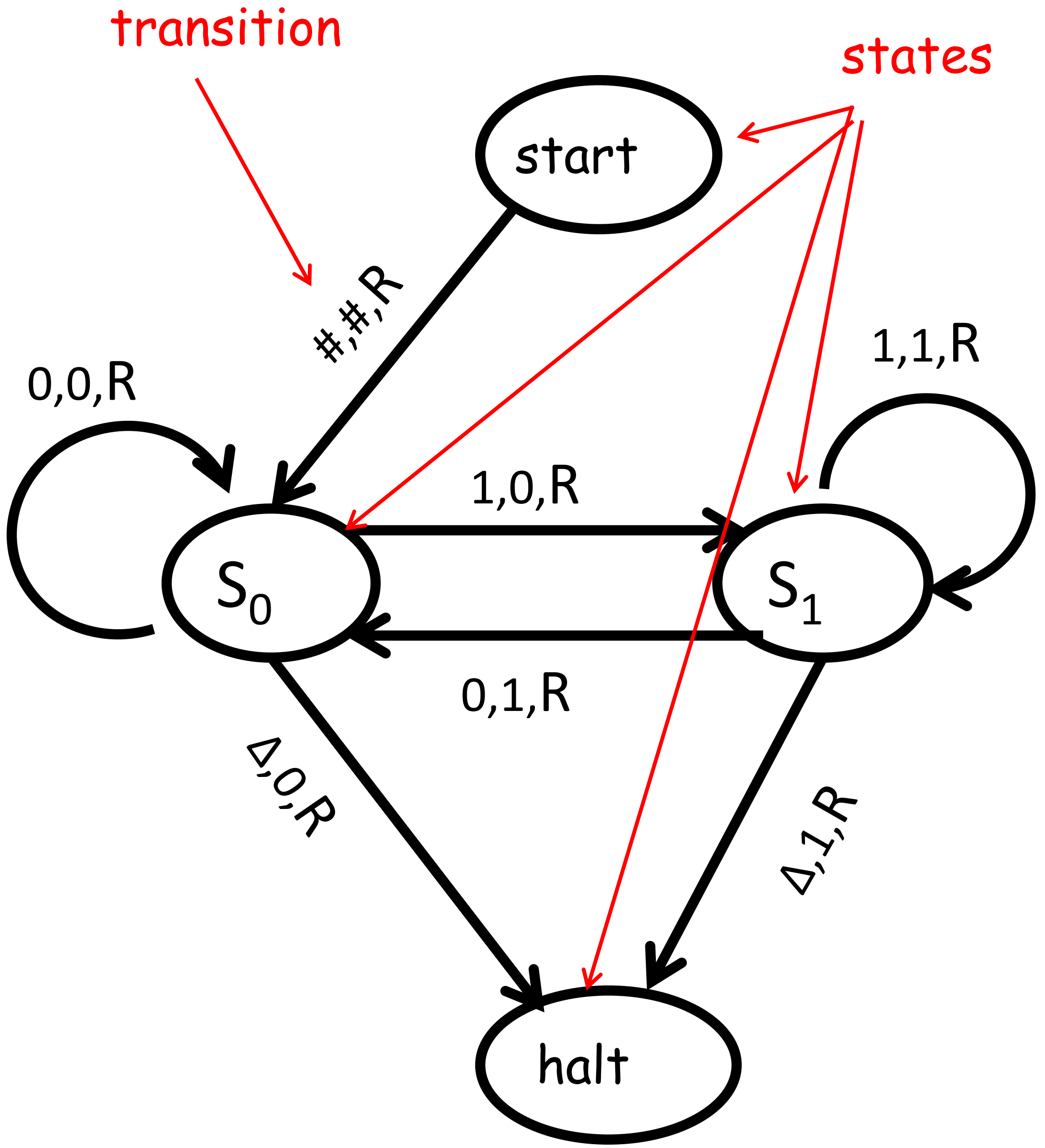
High Level Example of a Turing Machine

The machine that takes a binary string and appends 0 to the left side of the string.

Input: #10010Δ
Output: #010010Δ

# - leftmost char
Δ – rightmost char

Transition on each edge
read,write,move (L or R)



The Church-Turing Thesis

"*Any natural / reasonable notion of computation can be simulated by a TM.*"

This is not a theorem.

Is it… *…an observation?*

…a definition?

…a hypothesis?

…a law of nature?

…a philosophical statement?

Everyone believes it.

No counterexample yet.

Undecidable Problems

Undecidable means that there is no computer program that always gives the correct answer: it may give the wrong answer or run forever without giving any answer.

The halting problem is the problem of deciding whether a given Turing machine halts when presented with a given input.

Turing's Theorem: The Halting Problem is not decidable.

Super-Turing computation

In 1995 Prof. Hava Siegelmann proposed
Artificial Recurrent Neural Networks (ARNN).

She proved mathematically that ARNNs have computational powers that extend the TM.

She claims that ARNNs can “compute” Turing non-computable functions.

As of today, the statement is not proven nor disproven.

Runtime Complexity

Let M be a Turing Machine that halts on all inputs.
Assume we compute the running time purely as a function of the length of the input string.

<u>Definition:</u> The running complexity is the function $f : N \to N$ such that f(n) is the maximum number of steps that M uses on any input of length n.

P and NP complexity classes

P = set of problems that can be solved in polynomial time by a deterministic TM.

NP = set of problems for which solution can be verified in polynomial time by a deterministic TM.

Polynomial Reduction: $Y \leq_p X$

To reduce a decision problem Y to a decision problem X (we write $Y \leq_p X$) we want a function f that maps Y to X such that:

1) f is a polynomial time computable

2) $\forall y \in Y$ (y is instance of Y) is YES if and only if $f(y) \in X$ is YES.

If we cannot solve Y, we cannot solve X.

We use this to prove NP completeness: knowing that Y is hard, we prove that X is at least as hard as Y.

$Y \le_p X$

If we can solve X in polynomial time,
we can solve Y in polynomial time.

Examples:

Bipartite Matching $\le_p$ Max-Flow

Circulation $\le_p$ Max-Flow

$Y \leq_p X$

If we can solve X, we can solve Y.

The *contrapositive* of the statement "if A, then B" is "if not B, then not A."

If we cannot solve Y, we cannot solve X.

Knowing that Y is hard, we prove that X is harder.

In plain form: X is at least as hard as Y.

Two ways of using $Y \leq_p X$

1) X is easy

If we can solve X in polynomial time, we can solve Y in polynomial time.

2) Y is hard

Then X is at least as hard as Y

NP-Hard and NP-Complete

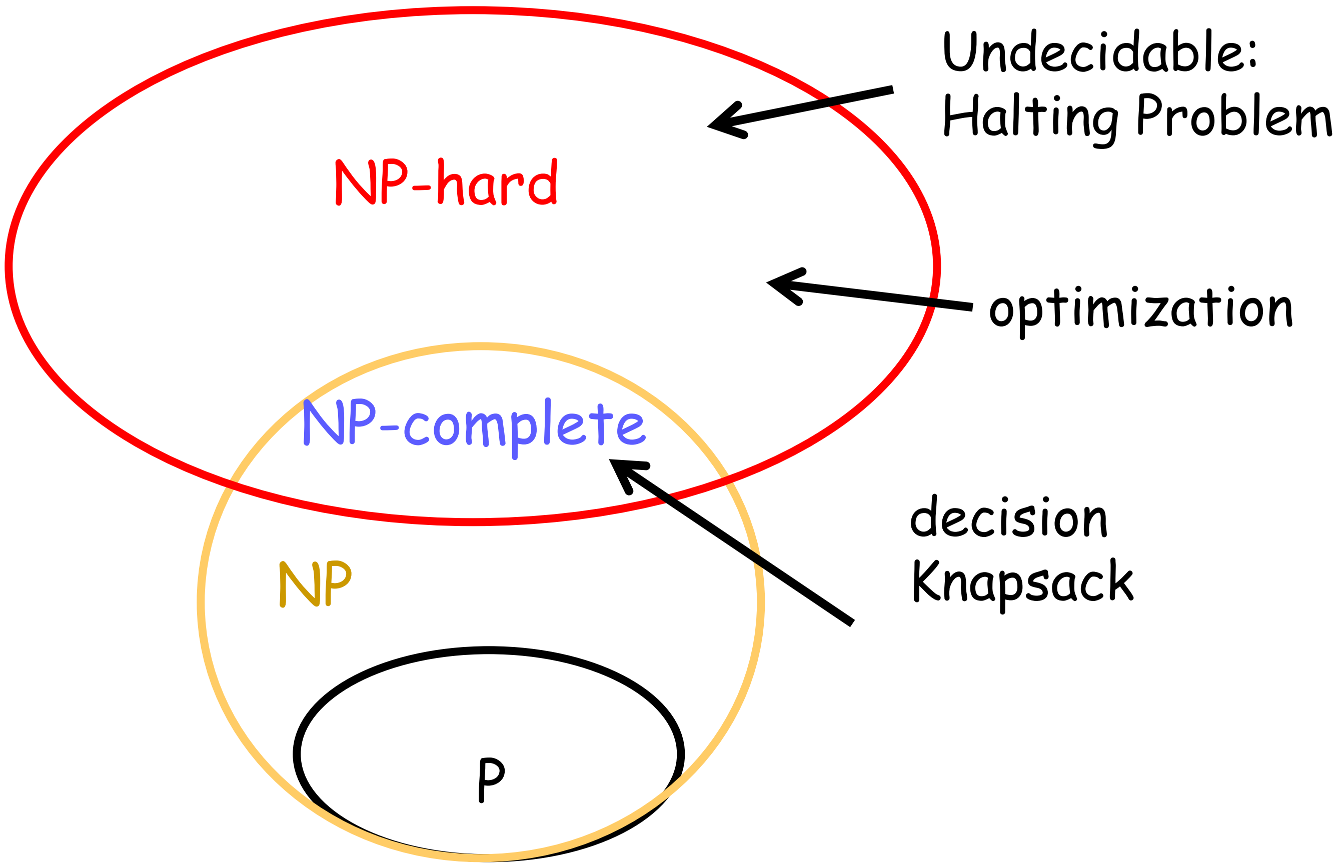
X is *NP-Hard*, if $\forall Y \in NP$ and $Y \leq_p X$.

X is *NP-Complete*, if X is NP-Hard and $X \in NP$.

Venn Diagram (P ≠ NP)

NPH problems do not have to be in NP.

NPC problems are the most difficult NP problems.



It's not known if NPC problems can be solved by a *deterministic* TM in polynomial time.

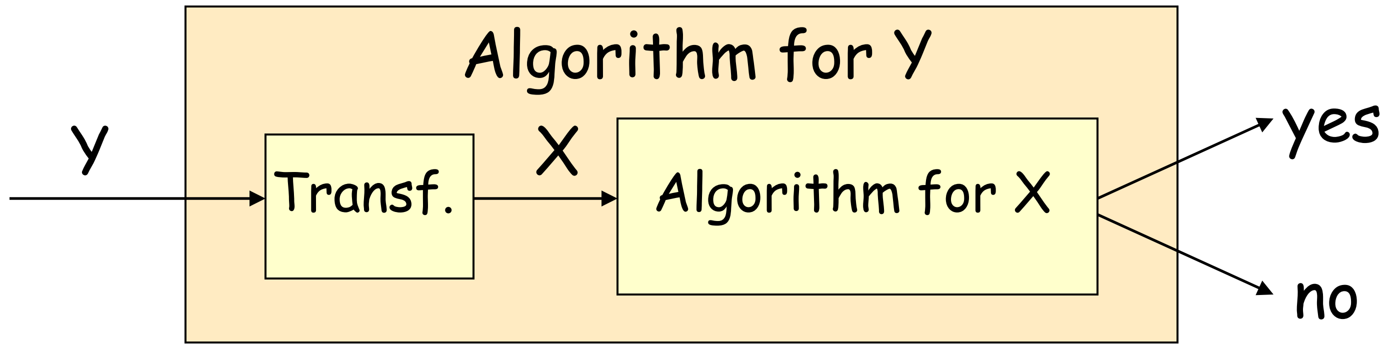
NP-Completeness Proof Method

To show that X is NP-Complete:

1) Show that X is in NP

2) Pick a problem Y, known to be an NP-Complete

3) Prove Y $\leq_p$ X (reduce Y to X)



Boolean Satisfiability Problem (SAT)

A propositional logic formula is built from variables, operators AND (conjunction, $\wedge$), OR (disjunction, $\vee$), NOT (negation, $\neg$), and parentheses:

$$(X_1 \vee \neg X_3) \wedge (X_1 \vee \neg X_2 \vee X_4 \vee X_5) \wedge \ldots$$

A formula is said to be satisfiable if it can be made TRUE by assigning appropriate logical values (TRUE, FALSE) to its variables.

A formula is in conjunctive normal form (CNF) if it is a conjunction of clauses.
A literal is a variable or its negation.
A clause is a disjunction of literals.

Cook-Levin Theorem (1971)

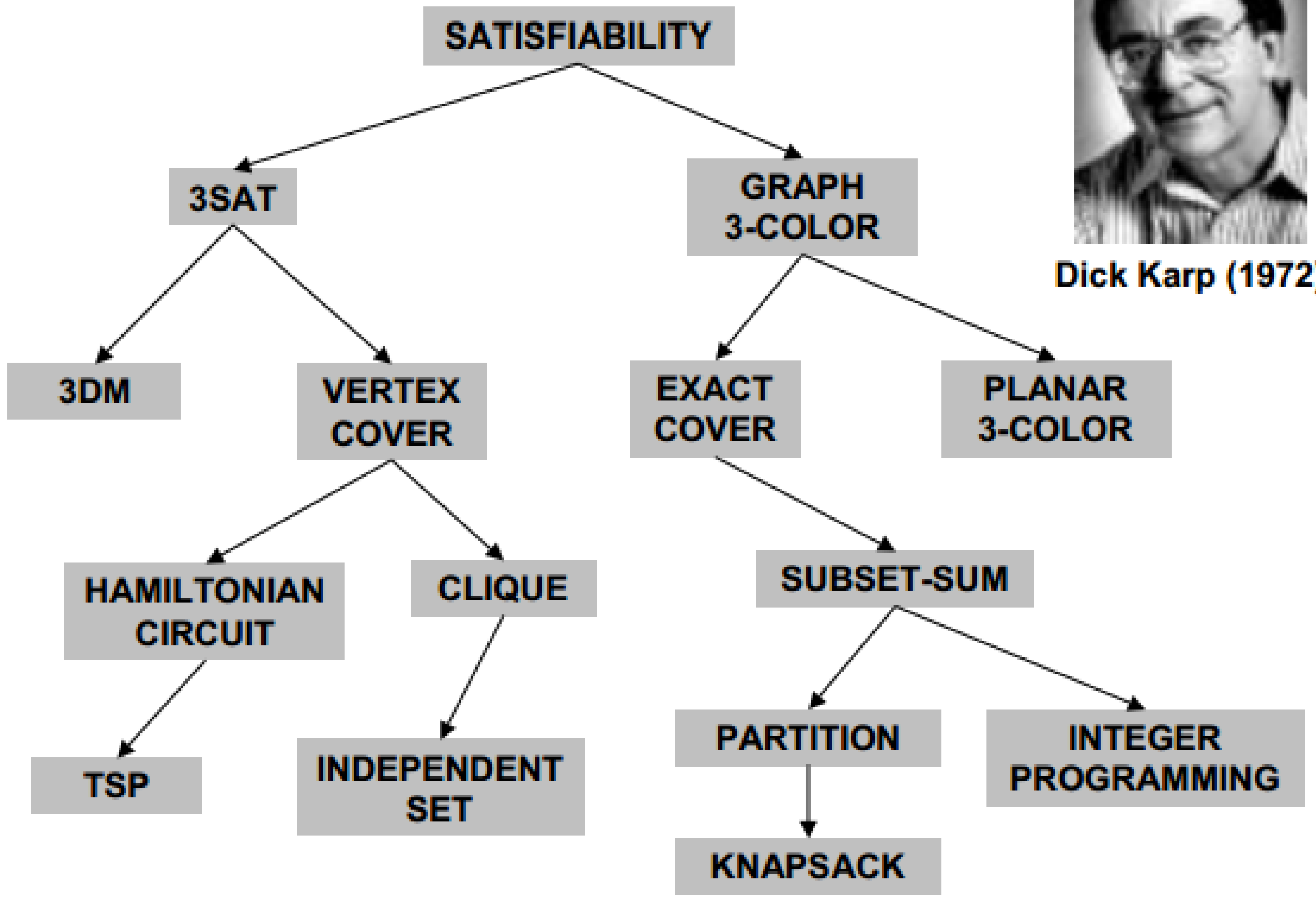
Theorem. CNF SAT is NP-complete.

The proof of this theorem is outside of the scope of this course.

Cook received a Turing Award for his work.

You are not responsible for knowing the proof.

Reduction



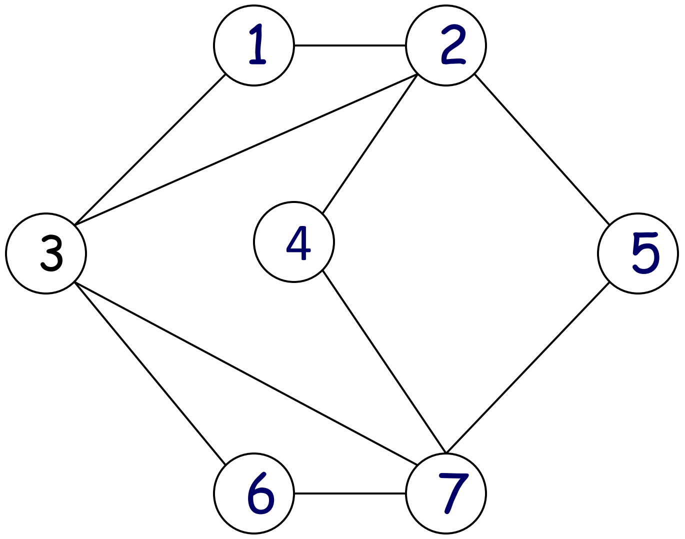
Dick Karp (1972)

Karp introduced the now standard methodology for proving problems to be NP-Complete.

He received a Turing Award for his work (1985).

Independent Set

Given a graph, we say that a subset of vertices is "independent" if no two of them are joined by an edge.



The maximum independent set problem, MaxIndSet, asks for the size of the largest independent set in a given graph.

Independent Set

Optimization Version (MaxIndSet):

Given a graph, find the largest independent set.

Decision Version (IndSet):

Given a graph and a number k, does the graph contains an independent set of size k?

Optimization vs. Decision

Optimization vs. Decision Problems

If one can solve an optimization problem (in polynomial time), then one can answer the decision version (in polynomial time)

Conversely, by doing binary search on the bound b, one can transform a polynomial time answer to a decision version into a polynomial time algorithm for the corresponding optimization problem

In that sense, these are essentially equivalent.
However, they belong to two different complexity classes.

Independent Set is NP Complete

Given a graph and a number k, does the graph contains an independent set of size k?

Is it in NP?

We need to show we can verify a solution in polynomial time.

Given a set of vertices, we can easily count them and then verify that any two of them are not joined by an edge.

Independent Set is NP Complete

Given a graph and a number k, does the graph contains an independent set of size k?

Is it in NP-hard?

We need to pick Y such that $Y \leq_p$ IndSet for $\forall$ Y $\in$ NP

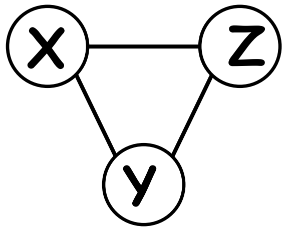
Reduce from 3-SAT.

3-SAT is SAT where each clause has at most 3 literals.

3SAT $\leq_p$ IndSet

We construct a graph G that will have an independent set of size k iff the 3-SAT instance with k clauses is satisfiable.

For each clause $(X \lor Y \lor Z)$ we will be using a special gadget:



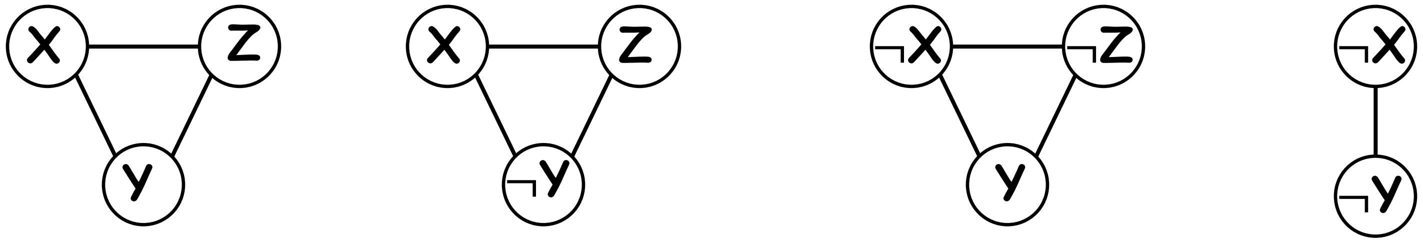
Next, we need to connect gadgets.

As an example, consider the following instance:

$(X \lor Y \lor Z) \land (X \lor \neg Y \lor Z) \land (\neg X \lor Y \lor \neg Z) \land (\neg X \lor \neg Y)$

3SAT $\leq_p$ IndSet

$(X \lor Y \lor Z) \land (X \lor \neg Y \lor Z) \land (\neg X \lor Y \lor \neg Z) \land (\neg X \lor \neg Y)$

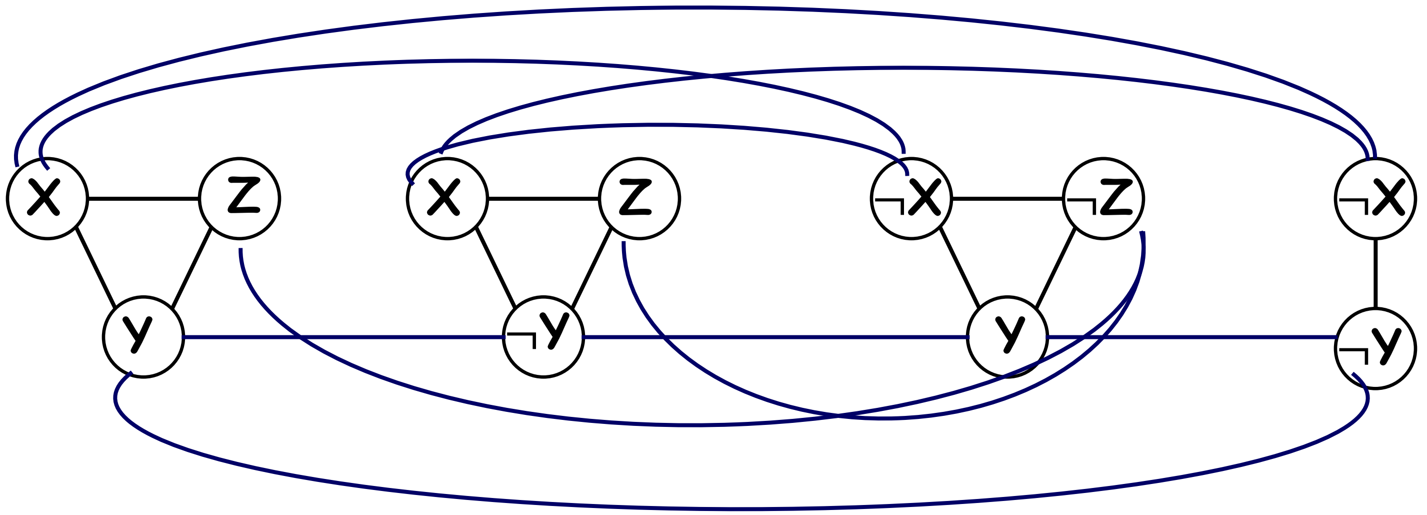


How do we connect gadgets?

<u>Claim:</u>

$3SAT \le_p IndSet$

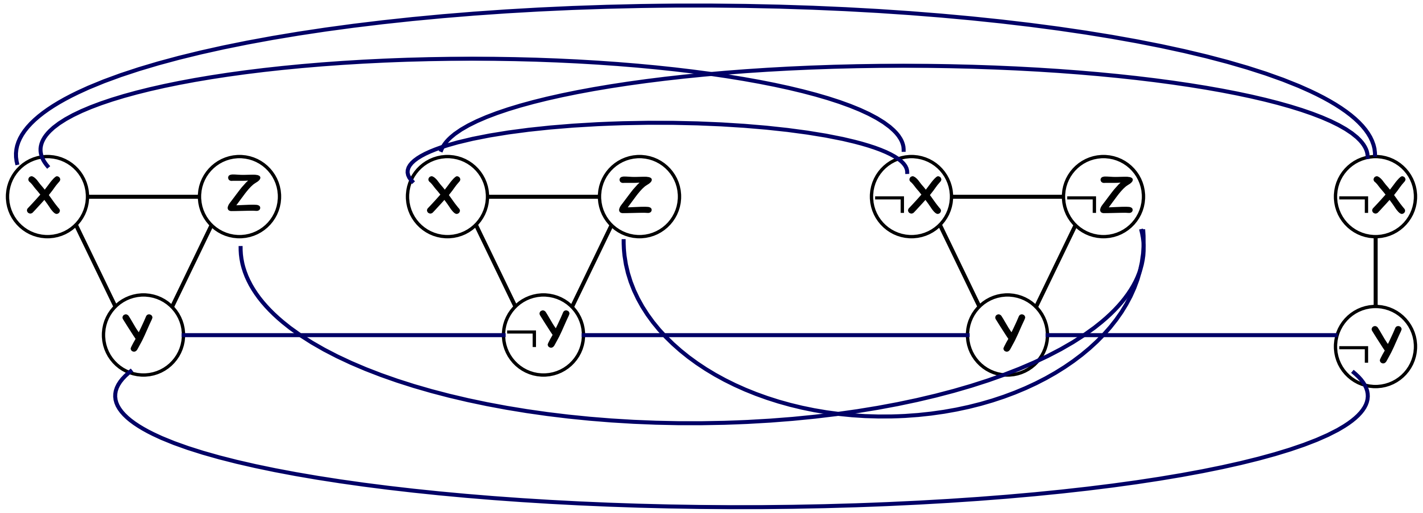
$(X \vee Y \vee Z) \wedge (X \vee \neg Y \vee Z) \wedge (\neg X \vee Y \vee \neg Z) \wedge (\neg X \vee \neg Y)$



Proof. $\Rightarrow$)

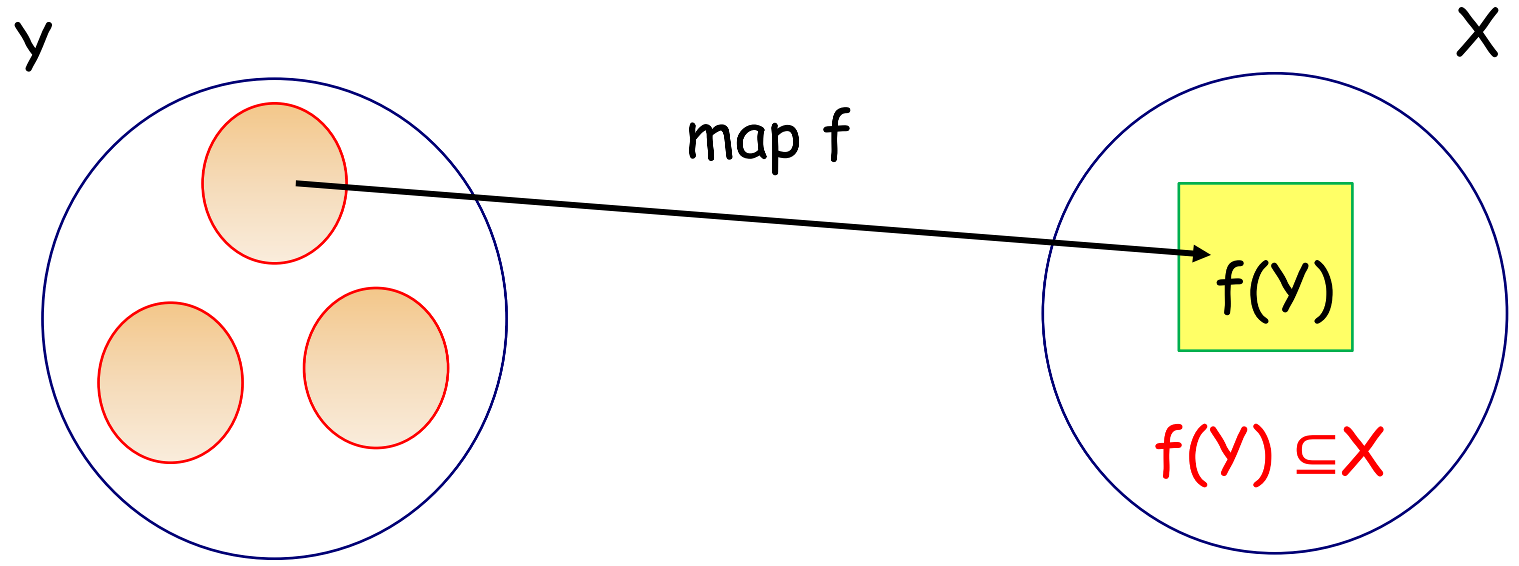
3SAT $\leq_p$ IndSet

$(X \vee Y \vee Z) \wedge (X \vee \neg Y \vee Z) \wedge (\neg X \vee Y \vee \neg Z) \wedge (\neg X \vee \neg Y)$



Proof. $\Leftarrow$)

The confusing point is that the reduction $Y \le_p X$ only "works one way", but the correctness proof needs to "work both ways".
The correctness proofs are not actually symmetric.
The proof needs to handle arbitrary instances of Y, but only needs to handle the special instances of X produced by the reduction.
This asymmetry is the key to understanding reductions.



Reduction from 3SAT to IndSet consists of three parts:

- we transform an arbitrary CNF formula into a special graph G and a specific integer k, in polynomial time.

- we transform an arbitrary <u>satisfying</u> assignment for 3SAT into an independent set in G of size k.

- we transform an arbitrary independent set (in G) of size k into a satisfying assignment for 3SAT.

The General Pattern $Y \le_p X$

1. Describe a polynomial-time algorithm to transform an arbitrary instance of Y into a special instance of X.
2. Prove that if an instance of Y is True, then an instance of X is True.
3. Prove that if an instance of X is True, then an instance of Y is True. (This part causes the most trouble.)