Computational Complexity*

Xiaofeng Gao

Department of Computer Science and Engineering
Shanghai Jiao Tong University, P.R.China

Algorithm Course: Shanghai Jiao Tong University

*Special thanks is given to *Prof. Kevin Wayne@Princeton* and also given to Mr. *Chao Wang from CS2014@SJTU* and *Hongjian Cao from CS2015@SJTU* for producing this lecture.

Outline

Classify Problems

Question: Which problems will we be able to solve in practice?

A working definition. [von Neumann 1953, Gödel 1956, Cobham 1964, Edmonds 1965, Rabin 1966]

Common Sense: Those with polynomial-time algorithms.

Yes	Probably Not
Shortest Path	Longest Path
Matching	3D-Matching
Min Cut	Max Cut
2-SAT	3-SAT
Planar 4-Color	Planar 3-Color
Bipartite Vertex Cover	Vertex Cover
Primality Testing	Factoring

Classify Problems

Desiderata. Classify problems according to those that can be solved in polynomial-time and those that cannot.

Provably requires exponential-time.

- Given a constant-size program, does it halt in at most k steps?
- Given a board position in an n-by-n generalization of checkers (using forced capture rule), can black guarantee a win?

Frustrating news. Huge number of fundamental problems have defied classification for decades.

Goal of This Slide

This slide shows that these fundamental problems are "computationally equivalent" and appear to be different manifestations of one really hard problem.

Decision Problems

Decision problem.

- X is a set of strings.
- Instance: string s.
- Algorithm A solves problem X: $A(s) = yes$ iff $s \in X$.

Polynomial time. Algorithm A runs in poly-time if for every string s, $A(s)$ terminates in at most $p(|s|)$ "steps", where $p(\cdot)$ is some polynomial. ($|s|$ is length of s)

PRIMES: $X = \{2, 3, 5, 7, 11, 13, 17, 23, 29, 31, 37, \ldots.\}$

Algorithm. [Agrawal-Kayal-Saxena, 2002] $p(|s|) = |s|^8$.

Decision Problem vs Search Problem

Decision problem.

- X is a set of strings.
- Instance: string s.
- Algorithm A solves problem X: $A(s) = yes$ iff $s \in X$.

Search problem.

- X is a set of strings.
- Instance: string s.
- Feasible solution: s_x.
- Algorithm A searches the optimal solution for problem X: $A(s) = \min\{|s_x|\}$ or $\max\{|s_x|\}$, $s \in X$.

Decision Problem vs Search Problem

Example 1:

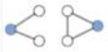
- Decision problem. Does there exist a shortest path of $weight \leq k$?
- Search problem. Find shortest path with minimum weight.

Example 2:

- Decision problem. Does there exist a clique of size k? (clique in a graph is a set of vertices such that every pair of vertices in the set is connected by an edge)
- Search problem. Given G, find a clique of size k in G, if it exists.

Definition of P

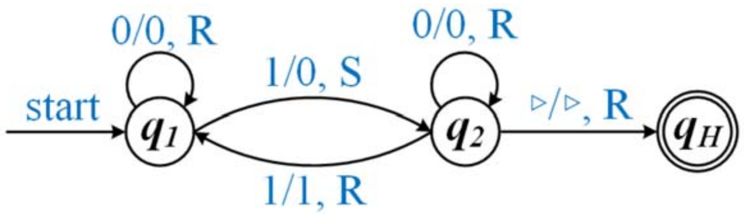
P: Decision problems for which there is a poly-time algorithm.

Problem	Description	Algorithm	yes	no
MULTIPLE	Is x a multiple of y?	grade-school division	51, 17	51, 16
REL-PRIME	Are x and y relatively prime?	Euclid (300 BCE)	34, 39	34, 51
PRIMES	Is x prime?	AKS (2002)	53	51
EDIT-DISTANCE	Is the edit distance between x and y less than 5?	dynamic programming	niether neither	acgggt ttttta
L-SOLVE	Is there a vector x that satisfies $Ax = b$?	Gauss-Edmonds elimination	$\begin{bmatrix} 0 & 1 & 1 \\ 2 & 4 & -2 \\ 0 & 3 & 15 \end{bmatrix}, \begin{bmatrix} 4 \\ 2 \\ 36 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$
ST-CONN	Is there a path between s and t in a graph G?	depth-first search (Theseus)		

Deterministic Turing Machine

The Deterministic Turing Machine (DTM) is a Turing machine M whose transaction function set has at most one instruction for each combination of symbol and state.

Example:



P: Decision problems that can be solved by a DTM polynomially.

NP

Certification algorithm intuition.

- Certifier views things from "managerial" viewpoint.
- Certifier doesn't determine whether $s \in X$ on its own; rather, it checks a proposed proof t that $s \in X$.

Definition: Algorithm $C(s, t)$ is a certifier for problem X if for every string s, $s \in X$ iff there exists a string such that $C(s, t) = yes$.

NP: Decision problems for which there exists a poly-time certifier. ($C(s, t)$ is a poly-time algorithm and $|t| \leq p(|s|)$ for some polynomial $p(*)$.)

Remark: NP stands for nondeterministic polynomial-time.

Certifiers and Certificates: Composite

COMPOSITES. Given an integer s, is s composite?

Certificate: A nontrivial factor t of s. Note that such a certificate exists iff s is composite. Moreover $|t| \leq |s|$.

Certifier:

Algorithm 1: Certifier

```
Function boolean C(s,t):
    if t ≤ 1 or t ≥ s then
        return false;
    else if s is a multiple of t then
        return true;
    else
        return false;
```

Example: $s = 437{,}669$.

- Certificate. $t = 541$ or 809. $(437,669 = 541 \times 809)$
- Conclusion. COMPOSITES is in NP.

Certifiers and Certificates: 3-Satisfiability

SAT. Given a CNF formula Φ, is there a satisfying assignment?

Certificate. An assignment of truth values to the n boolean variables.

Certifier. Check that each clause in Φ has at least one true literal.

Example:

- instance s:
  $(\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee \overline{x_3} \vee \overline{x_4})$
- certificate t: $x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 1$

Conclusion. SAT is in NP.

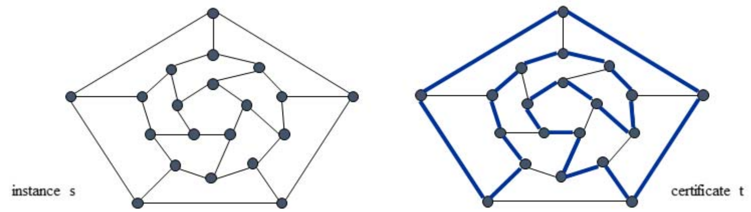
Certifiers and Certificates: Hamiltonian Cycle

HAM-CYCLE. Given an undirected graph $G = (V, E)$, does there exist a simple cycle C that visits every node?

Certificate. A permutation of the n nodes.

Certifier. Check that the permutation contains each node in V exactly once, and that there is an edge between each pair of adjacent nodes in the permutation.

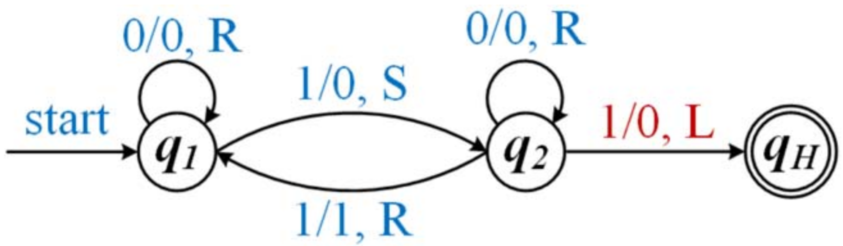
Conclusion. HAM-CYCLE is in NP.



Non-Deterministic Turing Machine

The Non-deterministic Turing Machine (NTM) is a Turing machine M which may have a set of specifications that prescribes more than one action for a given state.

Example:



NP. Decision problems that can be solved by a NTM polynomially. (certificate of NP problem can be detected by a DTM in polynomial)

P, NP, EXP

P: Decision problems for which there is a poly-time algorithm.
EXP: Decision problems for which there is an exponential-time algorithm.
NP: Decision problems for which there is a poly-time certifier.

Claim. $P \subseteq NP$.
Proof: Consider any problem X in P.

- By definition, there exists a poly-time algorithm $A(s)$ solving X.
- Certificate: $t = \epsilon$, certifier $C(s, t) = A(s)$.

Claim. $NP \subseteq NP$.
Proof: Consider any problem X in NP.

- By definition, there exists a poly-time certifier $C(s, t)$ for X.
- To solve input s, run $C(s, t)$ on all strings t with $|t| \leq p(|s|)$.
- Return yes, if $C(s, t)$ returns *yes* for any of these.

The Main Question: P Versus NP

Does P = NP? [Cook 1971, Edmonds, Levin, Yablonski, Gdel]

- Is the decision problem as easy as the certification problem?
- Clay $1 million prize.

If yes: Efficient algorithms for 3-COLOR, TSP, FACTOR, SAT, ...

If no: No efficient algorithms possible for 3-COLOR, TSP, SAT, ...

Consensus opinion on P = NP? Probably no.

Polynomial-Time Reduction

Desiderata. Suppose we could solve X in polynomial-time. What else could we solve in polynomial time?

Reduction. Problem X polynomial reduces to problem Y if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to oracle that solves problem Y.

Notation. $X \leq_P Y$.

Remarks.

- We pay for time to write down instances sent to black box $\Rightarrow$ instances of Y must be of polynomial size.
- Note: Cook reducibility. (in contrast to Karp reductions)

Polynomial-Time Reduction

Purpose. Classify problems according to relative difficulty.

Design algorithms. If $X \leq_P Y$ and Y can be solved in polynomial-time, then X can also be solved in polynomial time.

Establish intractability. If $X \leq_P Y$ and X cannot be solved in polynomial-time, then Y cannot be solved in polynomial time.

Establish equivalence. If $X \leq_P Y$ and $Y \leq_P X$, we use notation $X \equiv_P Y$.

Polynomial-Time Transformation

Definition: Problem X polynomial reduces (Cook) to problem Y if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to oracle that solves problem Y.

Definition: Problem X polynomial transforms (Karp) to problem Y if given any input x to X, we can construct an input y such that x is a yes instance of X iff y is a *yes* instance of Y

Note. Polynomial transformation is polynomial reduction with just one call to oracle for Y, exactly at the end of the algorithm for X.

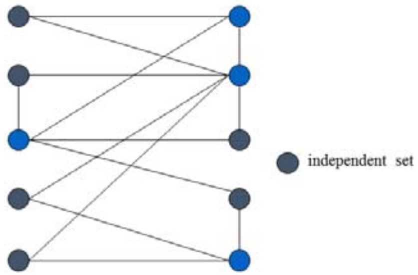
Open question. Are these two concepts the same with respect to NP?

Independent Set

INDEPENDENT SET: Given a graph $G = (V, E)$ and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \geq k$, and for each edge at most one of its endpoints is in S?

Example: Is there an independent set of size ≤ 6? Yes.

Example: Is there an independent set of size ≥ 7? Yes.

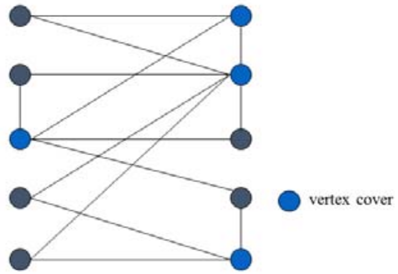


Vertex Cover

VERTEX COVER: Given a graph $G = (V, E)$ and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \leq k$, and for each edge, at least one of its endpoints is in S?

Example: Is there an vertex cover of size ≤ 4? Yes.

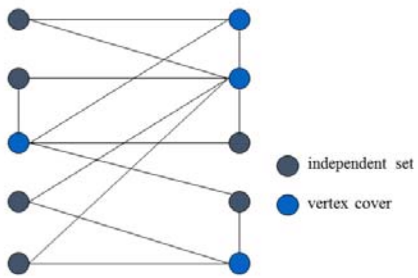
Example: Is there an vertex cover of size ≥ 3? No.



Vertex Cover and Independent Set

Claim. VERTEX-COVER $\equiv_P$ INDEPENDENT-SET.

Proof: We show S is an independent set iff $V - S$ is a vertex cover



Vertex Cover and Independent Set

Claim. VERTEX-COVER $\equiv_P$ INDEPENDENT-SET.

Proof: We show S is an independent set iff $V - S$ is a vertex cover

$\Rightarrow$ INDEPENDENT-SET $\leq_P$ VERTEX-COVER.

- Instance of IS: Given $G = (V, E)$, an integer k.
  Instance of VC: Same $G = (V, E)$, $k' = n - k$
- To prove:
  $\Rightarrow$ If S is any independent set with $|S| \leq k$, then $V - S$ is a vertex cover with $|V - S| \geq n - k$;
  $\Leftarrow$ If $V - S$ is a vertex cover with $|V - S| \geq n - k$, then S is an independent set with $|S| \leq k$.
- Let S be any independent set, consider an arbitrary edge (u, v), S independent $\Rightarrow u \notin S or v \notin S \Rightarrow u \in V - S or v \in V - S$.Thus, $V - S$ covers (u, v).

Vertex Cover and Independent Set

Claim. VERTEX-COVER $\equiv_P$ INDEPENDENT-SET.

Proof: We show S is an independent set iff $V - S$ is a vertex cover

$\Rightarrow$ VERTEX-COVER $\leq_P$ INDEPENDENT-SET.

- Instance of VC: Given $G = (V, E)$, an integer k.
  Instance of IS: Same $G = (V, E)$, $k' = n - k$
- To prove:
  $\Rightarrow$ If S is any vertex cover with $|S| \leq k$, then $V - S$ is an independent set with $|V - S| \geq n - k$;
  $\Leftarrow$ If $V - S$ is an independent set with $|V - S| \geq n - k$, then S is a vertex cover with $|S| \leq k$.
- Let S be any vertex cover. Consider two nodes $u \in V - S$ and $v \in V - S$. Observe that $(u, v) \notin E$ since S is a vertex cover. Thus, no two nodes in $V - S$ connected $\Rightarrow$ $V - S$ independent set.

Set Cover

SET COVER: Given a set U of elements, a collection $S_1, S_2, \ldots, S_m$ of subsets of U, and an integer k, does there exist a collection of $\leq k$ of these sets whose union is equal to U?

Sample application.

- m available pieces of software.
- Set U of n capabilities that we would like our system to have.
- The i-th piece of software provides the set $S_i \subseteq U$ of capabilities.
- Goal: achieve all n capabilities using fewest pieces of software.

Example:

$U = \{1, 2, 3, 4, 5, 6, 7\}$
$k = 2$

$S_1 = \{3, 7\}$	$S_4 = \{2, 4\}$
$S_2 = \{3, 4, 5, 6\}$	$S_5 = \{5\}$
$S_3 = \{1\}$	$S_6 = \{1, 2, 6, 7\}$

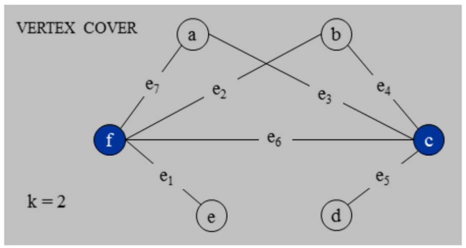
Vertex Cover Reduces to Set Cover

Claim: VERTEX-COVER $\leq_P$ SET-COVER.

Proof: Given a VERTEX-COVER instance $G = (V, E)$, k, we construct a set cover instance whose size equals the size of the vertex cover instance.

Construction.

- Create SET-COVER instance:
  $k = k$, $U = E$, $S_v = \{e \in E : e \text{ incident to } v\}$
- Set-cover of size $\leq k$ iff vertex cover of size $\leq k$.



SET COVER

$U = \{1, 2, 3, 4, 5, 6, 7\}$
$k = 2$

$S_a = \{3, 7\}$	$S_b = \{2, 4\}$
$S_c = \{3, 4, 5, 6\}$	$S_d = \{5\}$
$S_e = \{1\}$	$S_f = \{1, 2, 6, 7\}$

Satisfiability

Literal: A Boolean variable or its negation.

Clause: A disjunction of literals.

Conjunctive normal form: A propositional formula ϕ that is the conjunction of clauses.

$x_i, \overline{x_i}$
$C_j = x_1 \vee \overline{x_2} \vee x_3$
$\phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$

SAT: Given CNF formula ϕ, does it have a satisfying truth assignment?

3-SAT: SAT where each clause contains exactly 3 literals.

Example: $(\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$

Yes: x_1 = true, x_2 = true x_3 = false.

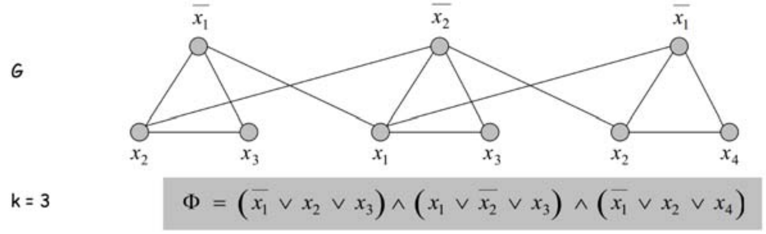
3 Satisfiability Reduces to Independent Set

Claim. 3-SAT $\leq_P$ INDEPENDENT-SET.

Proof: Given an instance Φ of 3-SAT, we construct an instance (G, k) of INDEPENDENT-SET that has an independent set of size k iff Φ is satisfiable.

Construction.

- G contains 3 vertices for each clause, one for each literal.
- Connect 3 literals in a clause in a triangle.
- Connect literal to each of its negations.



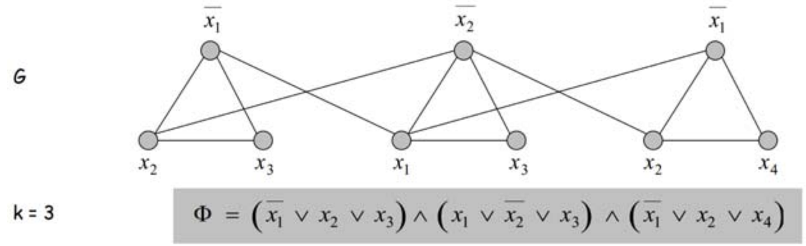
3 Satisfiability Reduces to Independent Set

Claim. G contains independent set of size $k = |\Phi|$ iff Φ is satisfiable.

Proof:$\Rightarrow$ Let S be independent set of size k.

- S must contain exactly one vertex in each triangle.
- Set these literals to true and any other variables in a consistent way.
- Truth assignment is consistent and all clauses are satisfied.

Proof:$\Leftarrow$ Given satisfying assignment, select one true literal from each triangle. This is an independent set of size k.



Review

Basic reduction strategies.

- Simple equivalence: INDEPENDENT-SET $\equiv_P$ VERTEX-COVER.
- Special case to general case: VERTEX-COVER $\leq_P$ SET-COVER.
- Encoding with gadgets: 3-SAT $\leq_P$ INDEPENDENT-SET.

Transitivity. If $X \leq_P Y$ and $Y \leq_P Z$, then $X \leq_P Z$.

Proof idea. Compose the two algorithms.

Example: 3-SAT $\leq_P$ INDEPENDENT-SET $\leq_P$ VERTEX-COVER $\leq_P$ SET-COVER.

Self-Reducibility

Decision problem. Does there exist a vertex cover of size $\leq k$?

Search problem. Find vertex cover of minimum cardinality.

Self-reducibility. Search problem $\leq_P$ decision version.

- Applies to all (NP-complete) problems in this chapter
- Justifies our focus on decision problems.

Example: to find min cardinality vertex cover.

- (Binary) search for cardinality k^* of min vertex cover.
- Find a vertex v such that $G - \{v\}$ has a vertex cover of size $\leq k^* - 1$, any vertex in any min vertex cover will have this property
- Include v in the vertex cover.
- Recursively find a min vertex cover in $G - \{v\}$.

NP-Complete

NP-complete. A problem Y in NP with the property that for every problem X in NP, $X \leq_p Y$.

Theorem. Suppose Y is an NP-complete problem. Then Y is solvable in poly-time iff $P = NP$.

Proof:$\Leftarrow$ If $P = NP$ then Y can be solved in poly-time since Y is in NP.

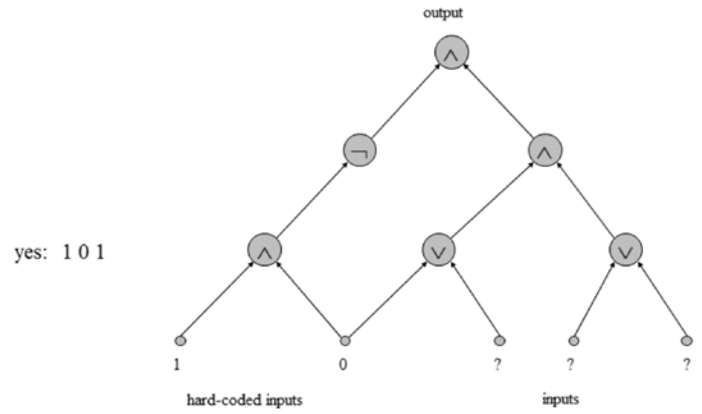
Proof:$\Rightarrow$ Suppose Y can be solved in poly-time.

- Let X be any problem in NP. Since $X \leq_p Y$, we can solve X in poly-time. This implies $NP \subseteq P$.
- We already know $P \subseteq NP$. Thus $P = NP$.

Fundamental question. Do there exist "natural" NP-complete problems?

Circuit Satisfiability

CIRCUIT-SAT. Given a combinational circuit built out of AND, OR, and NOT gates, is there a way to set the circuit inputs so that the output is 1?



The "First" NP-Complete Problem

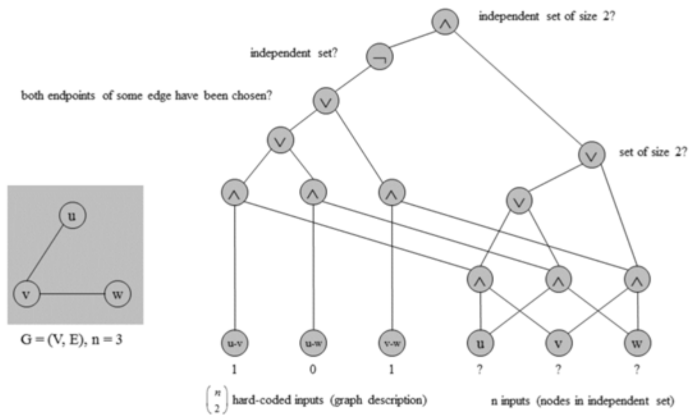
Theorem. CIRCUIT-SAT is NP-complete. [Cook 1971, Levin 1973]

Proof: (sketch)

- Any algorithm that takes a fixed number of bits n as input and produces a yes/no answer can be represented by such a circuit. sketchy part of proof; fixing the number of bits is important, and reflects basic distinction between algorithms and circuits
- Consider some problem X in NP. It has a poly-time certifier $C(s,t)$. To determine whether s is in X, need to know if there exists a certificate t of length $p(|s|)$ such that $C(s,t) = yes$.
- View $C(s,t)$ as an algorithm on $|s| + p(|s|)$ bits (input s, certificate t) and convert it into a poly-size circuit K.
  - first $|s|$ bits are hard-coded with s
  - remaining $p(|s|)$ bits represent bits of t
- Circuit K is satisfiable iff $C(s,t) = yes$.

Example

Example: Construction below creates a circuit K whose inputs can be set so that K outputs true iff graph G has an independent set of size 2.



Establishing NP-Completeness

Remark. Once we establish first "natural" NP-complete problem, others fall like dominoes.

Recipe to establish NP-completeness of problem Y.

- Step 1. Show that Y is in NP.
- Step 2. Choose an NP-complete problem X.
- Step 3. Prove that $X \leq_p Y$.

Justification. If X is an NP-complete problem, and Y is a problem in NP with the property that $X \leq_P Y$ then Y is NP-complete.

Proof: Let W be any problem in NP. Then $W \leq_p X \leq_p Y$.

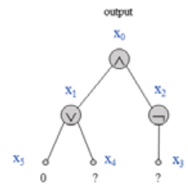
- By transitivity, $W \leq_P Y$.
- Hence Y is NP-complete.

3-SAT is NP-Complete

Theorem. 3-SAT is NP-complete.

Proof: Suffices to show that CIRCUIT-SAT $\leq_P$ 3-SAT since 3-SAT is in NP.

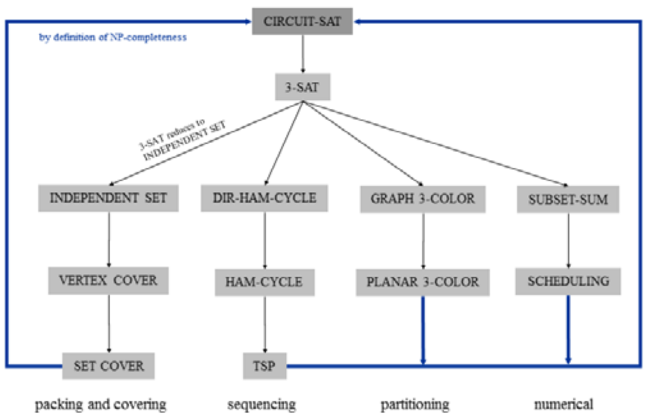
- Let K be any circuit.
- Create a 3-SAT variable x_i for each circuit element i.
- Make circuit compute correct values at each node:
  - $x_2 = \neg x_3 \Rightarrow$ add 2 clauses: $x_2 \vee x_3, \bar{x_2} \vee \bar{x_3}$
  - $x_1 = x_4 \vee x_5 \Rightarrow$ add 3 clauses: $x_1 \vee \bar{x_4}, x_1 \vee \bar{x_5}, \bar{x_1} \vee x_4 \vee x_5$
  - $x_0 = x_1 \wedge x_2 \Rightarrow$ add 3 clauses: $\bar{x_0} \vee x_1, \bar{x_0} \vee x_2, x_0 \vee \bar{x_1} \vee \bar{x_2}$
- Hard-coded input values and output value.
  - $x_2 = 0 \Rightarrow$ add 1 clauses: $\bar{x_5}$
  - $x_0 = 1 \Rightarrow$ add 1 clauses: x_0
- Final step: turn clauses of length < 3 into clauses of length exactly 3.



NP-Completeness

Observation. All problems below are NP-complete and polynomial reduce to one another!

Dick Karp (1972)
1985 Turing Award



Some NP-Complete Problems

Six basic genres of NP-complete problems and paradigmatic examples.

- Packing problems: SET-PACKING, INDEPENDENT SET.
- Covering problems: SET-COVER, VERTEX-COVER.
- Constraint satisfaction problems: SAT, 3-SAT.
- Sequencing problems: HAMILTONIAN-CYCLE, TSP.
- Partitioning problems: 3D-MATCHING 3-COLOR.
- Numerical problems: SUBSET-SUM, KNAPSACK.

Practice. Most NP problems are either known to be in P or NP-complete.

Notable exceptions. Factoring, graph isomorphism, Nash equilibrium.

More Hard Computational Problems

Aerospace engineering: optimal mesh partitioning for finite elements.
Biology: protein folding.
Chemical engineering: heat exchanger network synthesis.
Civil engineering: equilibrium of urban traffic flow.
Financial engineering: find minimum risk portfolio of given return.
Game theory: find Nash equilibrium that maximizes social welfare.
Mechanical engineering: structure of turbulence in sheared flows.
Medicine: reconstructing 3-D shape from biplane angiocardiogram.
Physics: partition function of 3-D Ising in statistical mechanics.
Statistics: optimal experimental design.

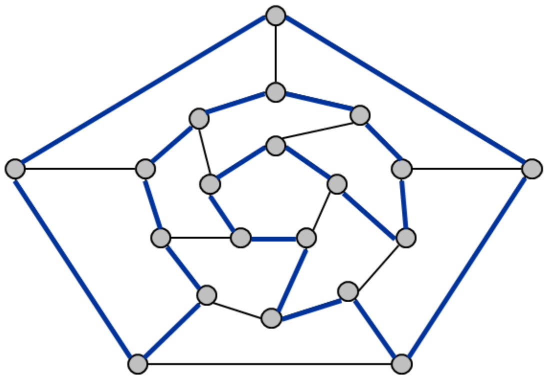
Sequencing problems

Basic genres.

- Packing problems: SET-PACKING, INDEPENDENT SET.
- Covering problems: SET-COVER, VERTEX-COVER.
- Constraint satisfaction problems: SAT, 3-SAT.
- Sequencing problems: HAMILTONIAN-CYCLE, TSP.
- Partitioning problems: 3D-MATCHING, 3-COLOR.
- Numerical problems: SUBSET-SUM, KNAPSACK.

Hamiltonian Cycle

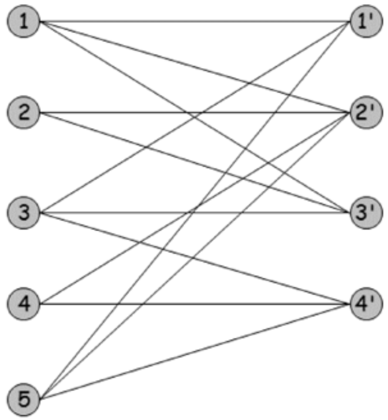
HAM-CYCLE: given an undirected graph $G = (V, E)$, does there exist a simple cycle Γ that contains every node in V.



YES: vertices and faces of a dodecahedron.

Hamiltonian Cycle

HAM-CYCLE: given an undirected graph $G = (V, E)$, does there exist a simple cycle Γ that contains every node in V.

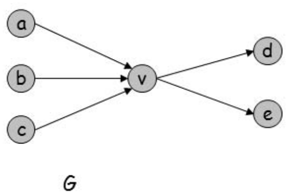


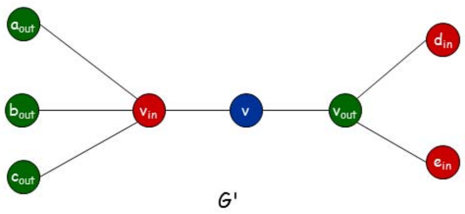
Directed Hamiltonian Cycle

DIR-HAM-CYCLE: given a digraph $G = (V, E)$, does there exists a simple directed cycle Γ that contains every node in V?

Claim. DIR-HAM-CYCLE $\leq_P$ HAM-CYCLE.

Proof: Given a directed graph $G = (V, E)$, construct an undirected graph G' with $3n$ nodes.





Directed Hamiltonian Cycle

Claim. G has a Hamiltonian cycle iff G' does.

Proof: $\Leftarrow$

- Suppose G has a directed Hamiltonian cycle Γ.
- Then G' has an undirected Hamiltonian cycle (same order).

Proof: $\Rightarrow$

- Suppose G' has an undirected Hamiltonian cycle Γ'.
- Γ' must visit nodes in G' using one of following two orders:
  ..., B, G, R, B, G, R, B, G, R, B, ...
  ..., B, R, G, B, R, G, B, R, G, B, ...
- Blue nodes in Γ' make up directed Hamiltonian cycle Γ in G, or reverse of one.

3-SAT Reduces to Directed Hamiltonian Cycle

Claim. 3-SAT $\leq_P$ DIR-HAM-CYCLE.

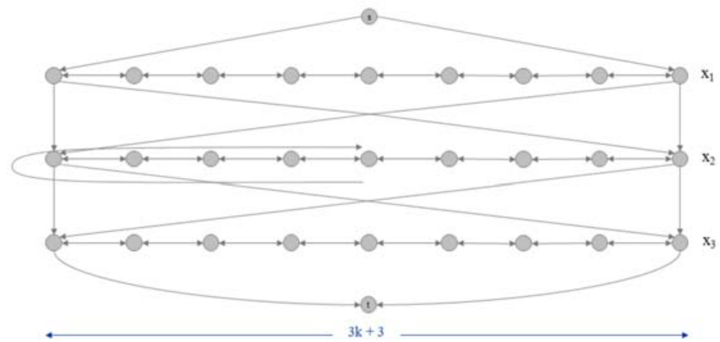
Proof: Given an instance Φ of 3-SAT, we construct an instance of DIRHAM-CYCLE that has a Hamiltonian cycle iff Φ is satisfiable.

Construction. First, create graph that has $2n$ Hamiltonian cycles which correspond in a natural way to $2n$ possible truth assignments.

3-SAT Reduces to Directed Hamiltonian Cycle

Construction. Given 3-SAT instance Φ with n variables x_i and k clauses.

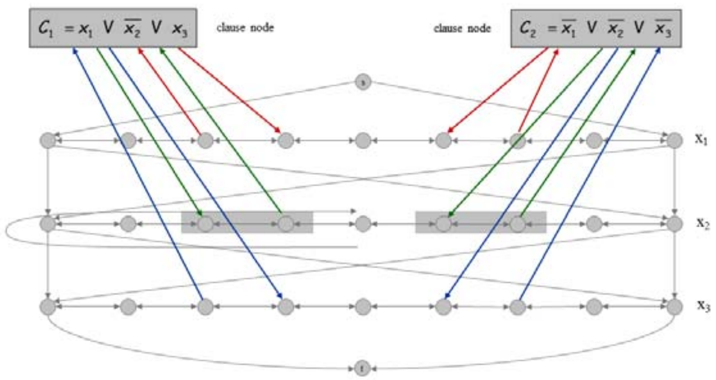
- Construct G to have $2n$ Hamiltonian cycles.
- Intuition: traverse path i from left to right $\Leftrightarrow$ set variable $x_i = 1$.



3-SAT Reduces to Directed Hamiltonian Cycle

Construction. Given 3-SAT instance Φ with n variables x_i and k clauses.

- For each clause: add a node and 6 edges.



3-SAT Reduces to Directed Hamiltonian Cycle

Claim. Φ is satisfiable iff G has a Hamiltonian cycle.

Proof: $\Rightarrow$

- Suppose 3-SAT instance has satisfying assignment x^*.
- Then, define Hamiltonian cycle in G as follows:
  - if $x_i^* = 1$, traverse row i from left to right
  - if $x_i^* = 0$, traverse row i from right to left
  - for each clause C_j , there will be at least one row i in which we are going in "correct" direction to splice node C_j into tour

3-SAT Reduces to Directed Hamiltonian Cycle

Claim. Φ is satisfiable iff G has a Hamiltonian cycle.
Proof:$\Leftarrow$

- Suppose G has a Hamiltonian cycle Γ.
- If Γ enters clause node C_j , it must depart on mate edge.
  - thus, nodes immediately before and after C_j are connected by an edge e in G
  - removing C_j from cycle, and replacing it with edge e yields Hamiltonian cycle on $G - \{C_j\}$
- Continuing in this way, we are left with Hamiltonian cycle Γ' in $G - \{C_1, C_2, ..., C_k\}$.
- Set $x_i^* = 1$ iff Γ' traverses row i left to right.
- Since Γ visits each clause node C_j , at least one of the paths is traversed in "correct" direction, and each clause is satisfied.

Longest Path

SHORTEST-PATH. Given a digraph $G = (V, E)$, does there exists a simple path of length at most k edges?
LONGEST-PATH. Given a digraph $G = (V, E)$, does there exists a simple path of length at least k edges?

Claim. 3-SAT $\leq_P$ LONGEST-PATH
Proof 1: Redo proof for DIR-HAM-CYCLE, ignoring back-edge from t to s.
Proof 2: Show HAM-CYCLE $\leq_P$ LONGEST-PATH.

The Longest Path

*

Lyrics. *Copyright*© 1988 by Daniel J. Barrett.
Music. Sung to the tune of The Longest Time by Billy Joel.

Woh-oh-oh-oh, find the longest path!
Woh-oh-oh-oh, find the longest path!

If you said P is NP tonight,
There would still be papers left to write,
I have a weakness,
I'm addicted to completeness,
And I keep searching for the longest path.

The algorithm I would like to see
Is of polynomial degree,
But it's elusive:
Nobody has found conclusive
Evidence that we can find a longest path.

I have been hard working for so long.
I swear it's right, and he marks it wrong.
Some how I'll feel sorry when it's done:
GPA 2.1
Is more than I hope for.

Garey, Johnson, Karp and other men (and women)
Tried to make it order N log N.
Am I a mad fool
If I spend my life in grad school,
Forever following the longest path?

Woh-oh-oh-oh, find the longest path!
Woh-oh-oh-oh, find the longest path!
Woh-oh-oh-oh, find the longest path.

*Recorded by Dan Barrett while a grad student at Johns Hopkins during a difficult algorithms final.

Traveling Salesperson Problem

TSP. Given a set of n cities and a pairwise distance function $d(u, v)$, is there a tour of length $\leq D$?

All 13,509 cities in US with a population of at least 500
Reference: http://www.tsp.gatech.edu

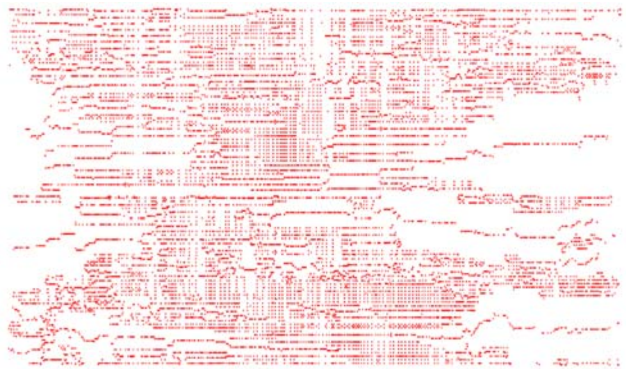
Traveling Salesperson Problem

TSP. Given a set of n cities and a pairwise distance function $d(u, v)$, is there a tour of length $\leq D$?

Optimal TSP tour
Reference: http://www.tsp.gatech.edu

Traveling Salesperson Problem

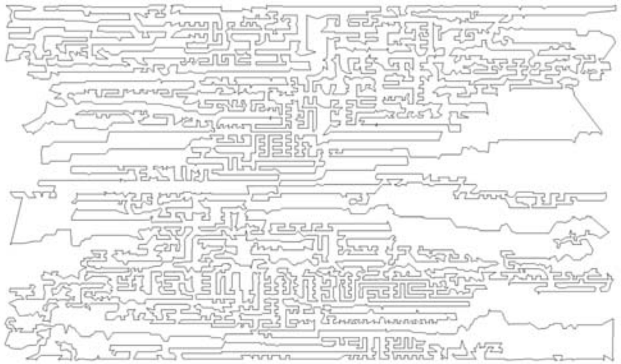
TSP. Given a set of n cities and a pairwise distance function $d(u, v)$, is there a tour of length $\leq D$?



11,849 holes to drill in a programmed logic array
Reference: http://www.tsp.gatech.edu

Traveling Salesperson Problem

TSP. Given a set of n cities and a pairwise distance function $d(u,v)$, is there a tour of length $\leq D$?



Optimal TSP tour
Reference: http://www.tsp.gatech.edu

Directed Hamiltonian Cycle

TSP. Given a set of n cities and a pairwise distance function $d(u,v)$, is there a tour of length $\leq D$?
HAM-CYCLE: given a graph $G=(V,E)$, does there exists a simple cycle that contains every node in V?
Claim. HAM-CYCLE $\leq_P$ TSP.
Proof:

- SGiven instance $G=(V,E)$ of HAM-CYCLE, create n cities with distance function

$$d(u,v)=\begin{cases}1 & if\ (u,v)\in E\\ 2 & if\ (u,v)\ni E\end{cases}$$

- TSP instance has tour of length $\leq n$ iff G is Hamiltonian.

Remark. TSP instance in reduction satisfies $\triangle - inequality$.

Partitioning Problems

Basic genres.

- Packing problems: SET-PACKING, INDEPENDENT SET.
- Covering problems: SET-COVER, VERTEX-COVER.
- Constraint satisfaction problems: SAT, 3-SAT.
- Sequencing problems: HAMILTONIAN-CYCLE, TSP.
- Partitioning problems: 3D-MATCHING, 3-COLOR.
- Numerical problems: SUBSET-SUM, KNAPSACK.

3-Dimensional Matching

3D-MATCHING. Given n instructors, n courses, and n times, and a list of the possible courses and times each instructor is willing to teach, is it possible to make an assignment so that all courses are taught at different times?

Instructor	Course	Time
Wayne	COS 423	MW 11-12:20
Wayne	COS 423	TTh 11-12:20
Wayne	COS 226	TTh 11-12:20
Wayne	COS 126	TTh 11-12:20
Tardos	COS 523	TTh 3-4:20
Tardos	COS 423	TTh 11-12:20
Tardos	COS 423	TTh 3-4:20
Kleinberg	COS 226	TTh 3-4:20
Kleinberg	COS 226	MW 11-12:20
Kleinberg	COS 423	MW 11-12:20

3-Dimensional Matching

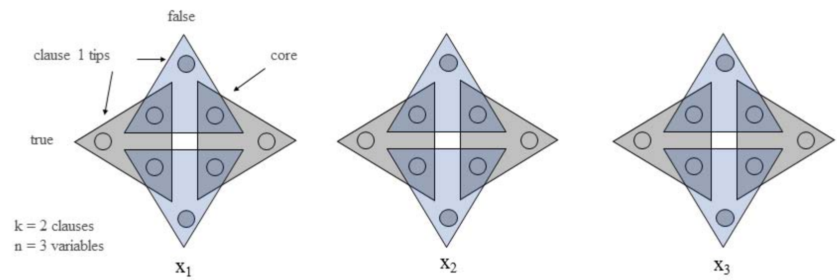
3D-MATCHING. Given disjoint sets X, Y, and Z, each of size n and a set $T \subseteq X \times Y \times Z$ of triples, does there exist a set of n triples in T such that each element of $X \cup Y \cup Z$ is in exactly one of these triples?

Claim. 3-SAT $\leq_P$ 3D-MATCHING.
Proof: Given an instance Φ of 3-SAT, we construct an instance of 3Dmatching that has a perfect matching iff Φ is satisfiable.

3-Dimensional Matching

Construction.(part 1)

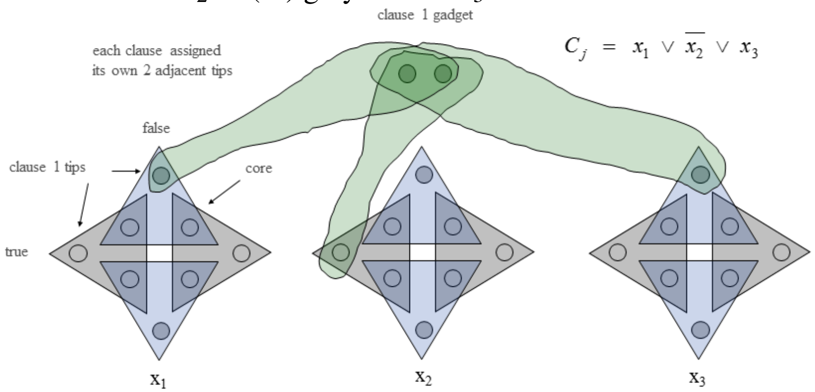
- Create gadget for each variable x_i with $2k$ (number of clauses) core and tip elements.
- No other triples will use core elements.
- In gadget i, 3D-matching must use either both grey (set x_i = true) triples or both blue ones (set x_i = false).



3-Dimensional Matching

Construction.(part 2)

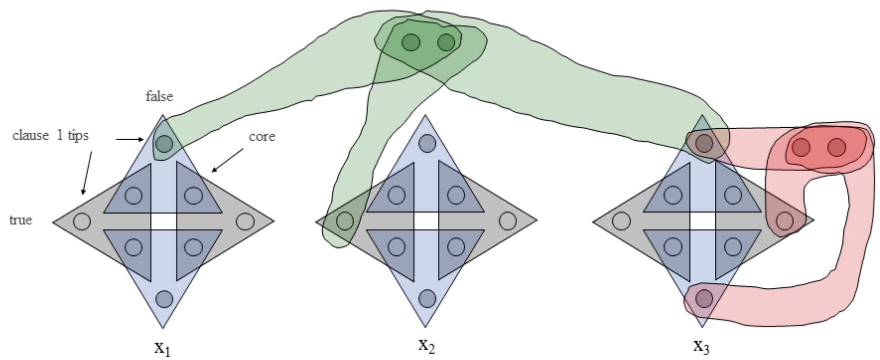
- For each clause C_j create two elements and three triples.
- Exactly one of these triples will be used in any 3D-matching.
- Ensures any 3D-matching uses either (i) grey core of x_1 or (ii) blue core of x_2 or (iii) grey core of x_3.



3-Dimensional Matching

Construction.(part 3)

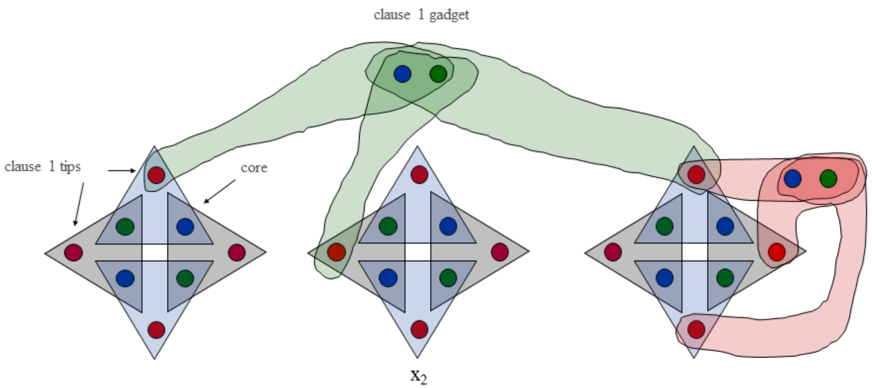
- For each tip, add a cleanup gadget.



3-Dimensional Matching

Claim. Instance has a 3D-matching iff Φ is satisfiable

Detail. What are X, Y, and Z? Does each triple contain one element from each of X, Y, Z?



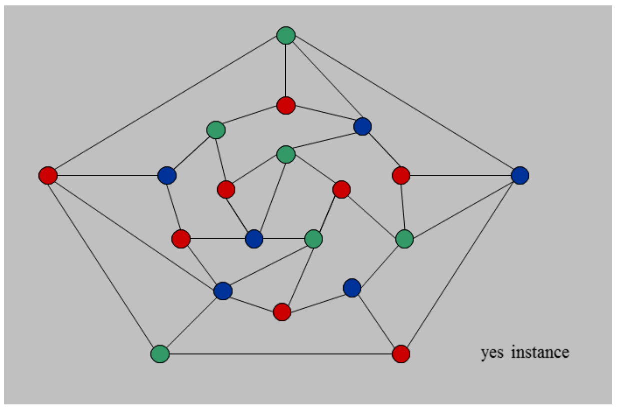
Graph Coloring

Basic genres.

- Packing problems: SET-PACKING, INDEPENDENT SET.
- Covering problems: SET-COVER, VERTEX-COVER.
- Constraint satisfaction problems: SAT, 3-SAT.
- Sequencing problems: HAMILTONIAN-CYCLE, TSP.
- Partitioning problems: 3D-MATCHING, 3-COLOR.
- Numerical problems: SUBSET-SUM, KNAPSACK.

3-Colorability

3-COLOR: Given an undirected graph G does there exists a way to color the nodes red, green, and blue so that no adjacent nodes have the same color?



Register Allocation

Register allocation. Assign program variables to machine register so that no more than k registers are used and no two program variables that are needed at the same time are assigned to the same register.

Interference graph. Nodes are program variables names, edge between u and v if there exists an operation where both u and v are "live" at the same time.

Observation. [Chaitin 1982] Can solve register allocation problem iff interference graph is k-colorable.

Fact. 3-COLOR $\leq_P$ k-REGISTER-ALLOCATION for any constant $k \geq 3$.

3-Colorability

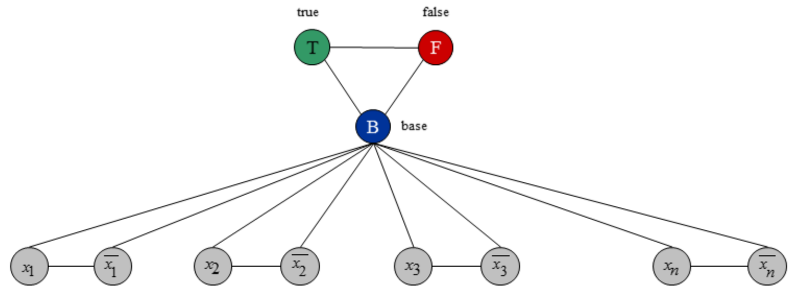
Claim. 3-SAT $\leq_P$ 3-COLOR.
Proof: Given 3-SAT instance Φ, we construct an instance of 3-COLOR that is 3-colorable iff Φ is satisfiable.
Construction.

- For each literal, create a node.
- Create 3 new nodes T, F, B; connect them in a triangle, and connect each literal to B.
- Connect each literal to its negation.
- For each clause, add gadget of 6 nodes and 13 edges.

3-Colorability

Claim. Graph is 3-colorable iff Φ is satisfiable.
Proof: $\Rightarrow$ Suppose graph is 3-colorable.

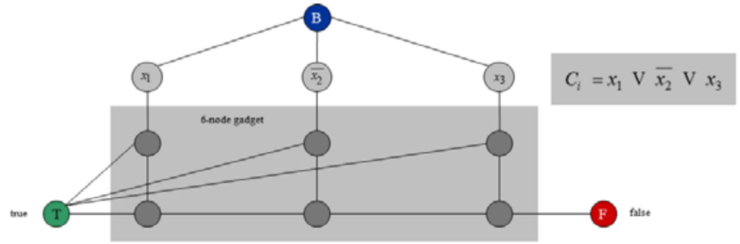
- Consider assignment that sets all T literals to true.
- ensures each literal is T or F.
- ensures a literal and its negation are opposites.



3-Colorability

Claim. Graph is 3-colorable iff Φ is satisfiable.
Proof: $\Rightarrow$ Suppose graph is 3-colorable.

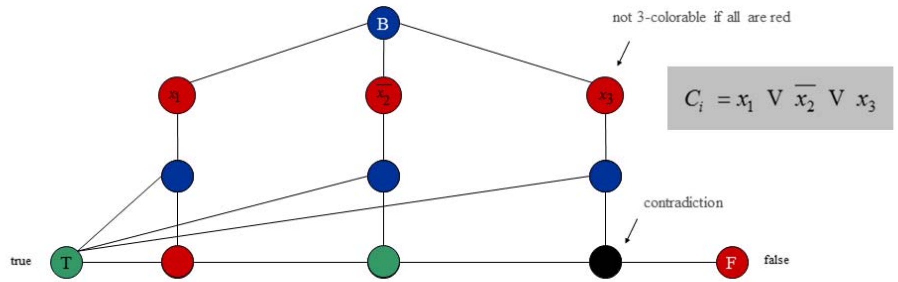
- Consider assignment that sets all T literals to true.
- ensures each literal is T or F.
- ensures a literal and its negation are opposites.
- ensures at least one literal in each clause is T.



3-Colorability

Claim. Graph is 3-colorable iff Φ is satisfiable.
Proof: $\Rightarrow$ Suppose graph is 3-colorable.

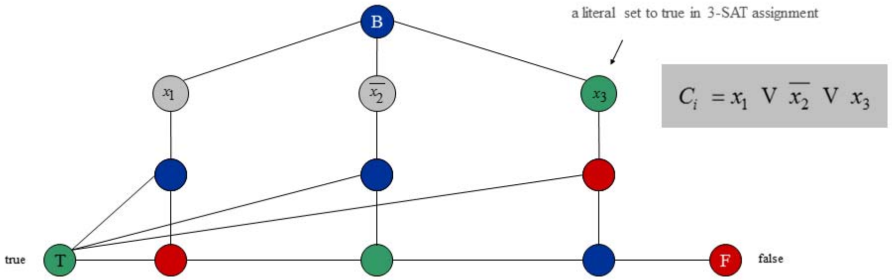
- Consider assignment that sets all T literals to true.
- ensures each literal is T or F.
- ensures a literal and its negation are opposites.
- ensures at least one literal in each clause is T.



3-Colorability

Claim. Graph is 3-colorable iff Φ is satisfiable.
Proof: $\Leftarrow$ Suppose 3-SAT formula Φ is satisfiable.

- Color all true literals T
- Color node below green node F, and node below that B.
- Color remaining middle row nodes B.
- Color remaining bottom nodes T or F as forced.



Graph Coloring

Basic genres.

- Packing problems: SET-PACKING, INDEPENDENT SET.
- Covering problems: SET-COVER, VERTEX-COVER.
- Constraint satisfaction problems: SAT, 3-SAT.
- Sequencing problems: HAMILTONIAN-CYCLE, TSP.
- Partitioning problems: 3D-MATCHING, 3-COLOR.
- Numerical problems: SUBSET-SUM, KNAPSACK.

Subset Sum

SUBSET-SUM. Given natural numbers $w_1, \ldots, w_n$ and an integer W, is there a subset that adds up to exactly W?

Example: $\{1, 4, 16, 64, 256, 1040, 1041, 1093, 1284, 1344\}$, $W = 3754$.
Yes. $1 + 16 + 64 + 256 + 1040 + 1093 + 1284 = 3754$

Remark. With arithmetic problems, input integers are encoded in binary. Polynomial reduction must be polynomial in binary encoding.

Claim. 3-SAT $\leq_P$ SUBSET-SUM.
Proof: Given an instance Φ of 3-SAT, we construct an instance of SUBSETSUM that has solution iff Φ is satisfiable.

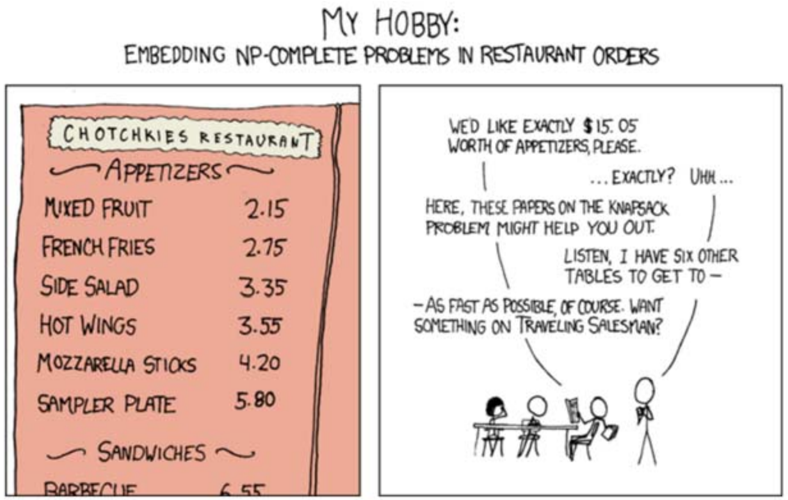
Subset Sum

Construction. Given 3-SAT instance Φ with n variables and k clauses, form $2n + 2k$ decimal integers, each of $n + k$ digits, as illustrated below.
Claim. Φ is satisfiable iff there exists a subset that sums to W.
Proof: No carries possible

$C_1 = \bar{x} \lor y \lor z$
$C_2 = x \lor \bar{y} \lor z$
$C_3 = \bar{x} \lor \bar{y} \lor \bar{z}$

	x	y	z	C_1	C_2	C_3	
x	1	0	0	0	1	0	100,010
¬ x	1	0	0	1	0	1	100,101
y	0	1	0	1	0	0	10,100
¬ y	0	1	0	0	1	1	10,011
z	0	0	1	1	1	0	1,110
¬ z	0	0	1	0	0	1	1,001
dummies to get clause columns to sum to 4	0	0	0	1	0	0	100
	0	0	0	2	0	0	200
	0	0	0	0	1	0	10
	0	0	0	0	2	0	20
	0	0	0	0	0	1	1
	0	0	0	0	0	2	2
W	1	1	1	4	4	4	111,444

My Hobby



Randall Munro
`http://xkcd.com/c287.html`

Introduction
Basic Reduction Strategies
NP-Completeness
Definition of NP-Complete
More Examples
co-NP and the Asymmetry of NP

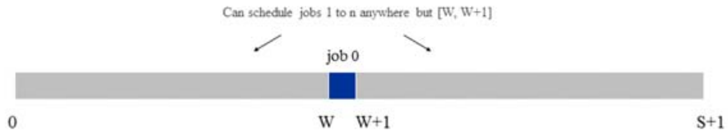
Scheduling With Release Times

SCHEDULE-RELEASE-TIMES. Given a set of n jobs with processing time t_i, release time r_i, and deadline d_i, is it possible to schedule all jobs on a single machine such that job i is processed with a contiguous slot of t_i time units in the interval $[r_i, d_i]$?

Claim. SUBSET-SUM $\leq_P$ SCHEDULE-RELEASE-TIMES.

Proof: Given an instance of SUBSET-SUM $w_1, \ldots, w_n$, and target W,

- Create n jobs with processing time $t_i = w_i$, release time $r_i = 0$, and no deadline ($d_i = 1 + \sum_j w_j$).
- Create job 0 with $t_0 = 1$, release time $r_0 = W$, and deadline $d_0 = W + 1$.



Introduction
Basic Reduction Strategies
NP-Completeness
Definition of NP-Complete
More Examples
co-NP and the Asymmetry of NP

Asymmetry of NP

Asymmetry of NP. We only need to have short proofs of yes instances.

Example 1: SAT vs. TAUTOLOGY.

- Can prove a CNF formula is satisfiable by giving such an assignment.
- How could we prove that a formula is not satisfiable?

Example 2: HAM-CYCLE vs. NO-HAM-CYCLE.

- Can prove a graph is Hamiltonian by giving such a Hamiltonian cycle.
- How could we prove that a graph is not Hamiltonian?

Remark: SAT is NP-complete and SAT $\equiv_P$ TAUTOLOGY, but how do we classify TAUTOLOGY (not even known to be in NP)?

Introduction
Basic Reduction Strategies
NP-Completeness
Definition of NP-Complete
More Examples
co-NP and the Asymmetry of NP

NP and co-NP

NP: Decision problems for which there is a poly-time certifier.
Example: SAT, HAM-CYCLE, COMPOSITES

Definition: Given a decision problem X, its complement $\bar{X}$ is the same problem with the yes and no answers reverse.

Example: $\bar{X} = \{0, 1, 4, 6, 8, 9, 10, 12, 14, 15, \ldots\}$
$X = \{2, 3, 5, 7, 11, 13, 17, 23, 29, \ldots\}$

co-NP: Complements of decision problems in NP.
Example: TAUTOLOGY, NO-HAM-CYCLE, PRIMES.

NP = co-NP ?

Fundamental question. Does NP = co-NP?

- Do *yes* instances have succinct certificates iff *no* instances do?
- Consensus opinion: no.

Theorem. If NP $\neq$ co-NP, then P $\neq$ NP.

Proof idea

- P is closed under complementation.
- If P = NP, then NP is closed under complementation.
- In other words, NP = co-NP.
- This is the contrapositive of the theorem.

Good Characterizations

Good characterization. [Edmonds 1965] NP $\bigcap$ co-NP.

- If problem X is in both NP and co-NP, then:
  - for *yes* instance, there is a succinct certificate
  - for *no* instance, there is a succinct disqualifier
- Provides conceptual leverage for reasoning about a problem.

Example: Given a bipartite graph, is there a perfect matching.

- If yes, can exhibit a perfect matching.
- If no, can exhibit a set of nodes S such that $|N(S)| < |S|$.

Good Characterizations

Observation. $P \subseteq NP \bigcap co - NP$

- Proof of max-flow min-cut theorem led to stronger result that max-flow and min-cut are in P.
- Sometimes finding a good characterization seems easier than finding an efficient algorithm

Fundamental open question. Does $P = NP \bigcap co - NP$?

- Mixed opinions.
- Many examples where problem found to have a non-trivial good characterization, but only years later discovered to be in P.
  - linear programming [Khachiyan, 1979]
  - primality testing [Agrawal-Kayal-Saxena, 2002]

Fact. Factoring is in NP $\bigcap$ co-NP, but notknown to be in P. (if poly-time algorithm for factoring, can break RSA cryptosystem)

PRIMES is in NP $\bigcap$ co-NP

Theorem. PRIMES is in NP $\bigcap$ co-NP.

Proof: We already know that PRIMES is in co-NP, so it suffices to prove that PRIMES is in NP.

Pratt's Theorem. An odd integer s is prime iff there exists an integer $1 < t < s$ s.t

$$t^{s-1} \equiv (mod\ s)$$
$$t^{(s-1)/p} \not\equiv (mod\ s)$$
$$for\ all\ prime\ divisors\ p\ of\ s-1$$

FACTOR is in NP $\bigcap$ co-NP

FACTORIZE. Given an integer x, find its prime factorization.
FACTOR. Given two integers x and y, does x have a nontrivial factor less than y?

Theorem. FACTOR $\equiv_P$ FACTORIZE.

Theorem. FACTOR is in NP $\bigcap$ co-NP.
Proof:

- Certificate: a factor p of x that is less than y.
- Disqualifier: the prime factorization of x (where each prime factor is less than y), along with a certificate that each factor is prime.

Primality Testing and Factoring

We established: PRIMES $\leq_P$ COMPOSITES $\leq_P$ FACTOR.

Natural question: Does FACTOR $\leq_P$ PRIMES ?

Consensus opinion. No.

State-of-the-art.

- PRIMES is in P.
- FACTOR not believed to be in P.

RSA cryptosystem.

- Based on dichotomy between complexity of two problems.
- To use RSA, must generate large primes efficiently.
- To break RSA, suffixes to find efficient factoring algorithm.