CS 583: Algorithms

NP Completeness
Ch 34

Intractability

- Some problems are *intractable*:
  as they grow large, we are unable to solve them in reasonable time
- What constitutes reasonable time? Standard working definition: *polynomial time*

- On an input of size n the worst-case running time is $O(n^k)$ for some constant k
  Polynomial time: $O(n^2)$, $O(n^3)$, $O(1)$, $O(n \lg n)$
  Not in polynomial time: $O(2^n)$, $O(n^n)$, $O(n!)$

Polynomial-Time Algorithms

- Many problems we've studied have algorithms with polynomial-time solution (sorting, searching, optimization, graph traversal)

- We define **P** to be the class of problems solvable in polynomial time

- Are all problems solvable in polynomial time?
- No: Turing's "Halting Problem" is not solvable by any computer, no matter how much time is given
- Halting problem: given a description of a program, decide whether program finishes running on particular input or runs forever
- One of the first undecidable problems

Tractability

- Halting problem is so called *undecidable problem*: no computer can solve them
- More about undecidable problems here; take a theory class

- We would like to discuss the relative hardness of different problems
- Focus on *decision problems:* solution is an answer yes/no

- Other problems are decidable, but *intractable*: as they grow large, we are unable to solve them in reasonable time

- *What constitutes "reasonable time"?*

Examples of problems

• Many problems which look very similar but the known solutions are very different:
• **Minimum spanning tree**: Given a weighted graph and integer **k** is there a spanning tree whose total weight is less then **k ?** (polynomial time problem known)
• **Traveling salesmen**: Given a weight graph and integer **k** , is there a cycle that visits all nodes exactly ones and has a total weight **k** or less ? (intractable)
• **Shortest Path vs. long simple paths:** Shortest path are easy (even with negative weight cycles); finding longest simple path between two vertices is difficult (path with no repeated vertices)
• **Hamiltonian path:** Given a directed graph, is there a closed path that visits each node of the graph exactly once ?
• **Euler tour:** Given a graph is there a path which visits every edge once, where vertices can be visited more then once ?

Examples of new problems

- **Circuit value** – Given a Boolean circuit and its inputs is the value T ?
- **Circuit Satisfiability** – Given a boolean circuit, is there a set of inputs that the output is T ?
- **2-SAT** – Given a boolean formula in 2-CNF is there a satisfying truth assignment to input variables
- **3-SAT** – Given a boolean formula in 3-CNF is there a satisfying truth assignment to input variables

P and NP

- We will discuss 3 classes of problems
- **P** is set of problems that can be solved in polynomial time

- **NP** (*nondeterministic polynomial time*) is the set of problems that can be solved in polynomial time by a *nondeterministic* computer, that can be "verified" in polynomial time
- Any problem which is in P is also in NP

- **NPC** – NP complete problems; problems which are in NP and are as hard as any problem in NP

Nondeterminism (in NP)

- Think of a non-deterministic computer as a computer that magically "guesses" a solution, then has to verify that it is correct
- If a solution exists, computer always guesses it
- One way to imagine it: a parallel computer that can freely spawn an infinite number of processes
- Have one processor work on each possible solution
  All processors attempt to verify that their solution works
  (that can be checked in polynomial time)

- If a processor finds it has a working solution
- So: **NP** = problems *verifiable* in polynomial time

P and NP

- Summary so far:
  P = problems that can be solved in polynomial time
  NP = problems for which a solution can be verified in polynomial time

- Unknown whether **P = NP** (most suspect not)
- Hamiltonian-cycle problem is in **NP**:
  Cannot be solved in polynomial time
  Easy to verify solution in polynomial time (*How?*)

NP-Complete Problems

- We will see that NP-Complete problems are the "hardest" problems in NP:
- **NP complete problems** – problems which are in NP but are as hard as any other problem in NP
- If any *one* NP-Complete problem can be solved in polynomial time, then *every* NP-Complete problem can be solved in polynomial time
- And in fact *every* problem in **NP** can be solved in polynomial time (which would show **P = NP**)

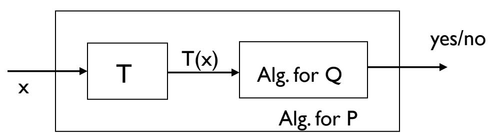
Polynomial Reducibility

• How to compare relative hardness of problems ?
• Notion of **reduction** of one problem to another.
• Given two problems P and Q there exist **polynomial time reduction** which reduces one problem to another.

• Idea given input to problem P, we transform the input of P to the input for problem Q, and if Q has answer YES (NO), then P has answer YES (NO)

$$P \leq^{P} Q$$

• If we can reduce P to Q in polynomial time, then P is no harder then Q (up to a polynomial)



Reduction

- The crux of NP-Completeness is *reducibility*
- Informally, a problem P can be reduced to another problem Q if *any* instance of P can be "easily rephrased" as an instance of Q, the solution to which provides a solution to the instance of P
  What do you suppose "easily" means?
- This rephrasing is called *transformation*
  Intuitively: If P reduces to Q, P is "no harder to solve" than Q

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.

computational model supplemented by special piece of hardware that solves instances of Y in a single step

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.

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.

  up to cost of reduction

- 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 $X \equiv_P Y$.

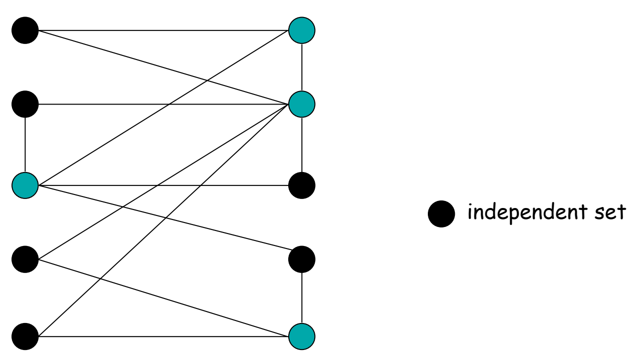
Basic reduction strategies.
- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.

Independent Set

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

•Ex. Is there an independent set of size ≥ 6? Yes.

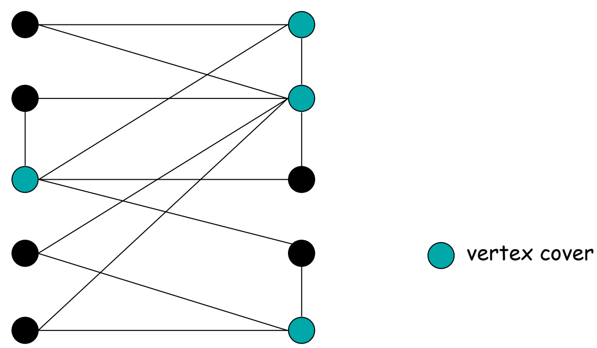
•Ex. Is there an independent set of size ≥ 7? No.



Vertex Cover

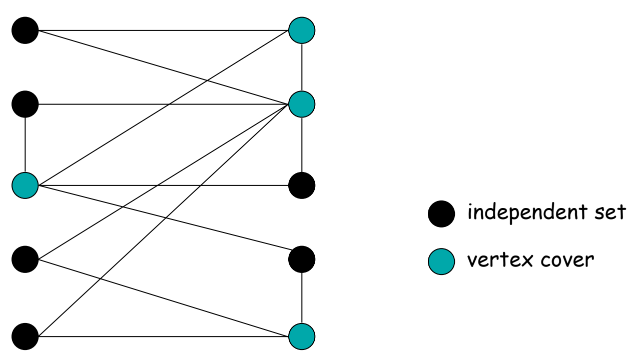
•VERTEX COVER: Given a graph G = (V, E) and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \leq k$, and for each edge, at least one of its endpoints is in S?

•Ex. Is there a vertex cover of size ≤ 4? Yes.
•Ex. Is there a vertex cover of size ≤ 3? No.



Vertex Cover and Independent Set

•Claim. VERTEX-COVER $\equiv_P$ INDEPENDENT-SET.
•Pf. 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.
- Pf. We show S is an independent set iff V - S is a vertex cover.
- $\Rightarrow$
  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).
- $\Leftarrow$
  Let V - S be any vertex cover.
  Consider two nodes $u \in S$ and $v \in S$.
  Observe that $(u, v) \notin E$ since V - S is a vertex cover.
  Thus, no two nodes in S are joined by an edge $\Rightarrow$ S independent set. ▪

Reduction from Special Case to General Case

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 ≤ 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 ith piece of software provides the set $S_i \subseteq U$ of capabilities.
- Goal: achieve all n capabilities using fewest pieces of software.

•Ex:

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

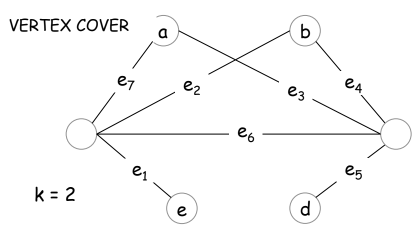
S_1 = {3, 7} S_4 = {2, 4}
S_2 = {3, 4, 5, 6} S_5 = {5}
S_3 = {1} S_6 = {1, 2, 6, 7}

Vertex Cover Reduces to Set Cover

•Claim. VERTEX-COVER $\le_P$ SET-COVER.

•Pf. 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 ≤ k iff vertex cover of size ≤ 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}

Polynomial-Time Reduction

- Basic strategies.
  - Reduction by simple equivalence.
  - Reduction from special case to general case.
  - Reduction by encoding with gadgets.

Satisfiability

- Literal: A Boolean variable or its negation. x_i or $\overline{x_i}$
- Clause: A disjunction of literals. $C_j = x_1 \vee \overline{x_2} \vee x_3$
- Conjunctive normal form: A propositional formula Φ that is the conjunction of clauses.

$$\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$$

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

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

$$\left(\overline{x_1} \vee x_2 \vee x_3\right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3\right) \wedge \left(x_2 \vee x_3\right) \wedge \left(\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}\right)$$

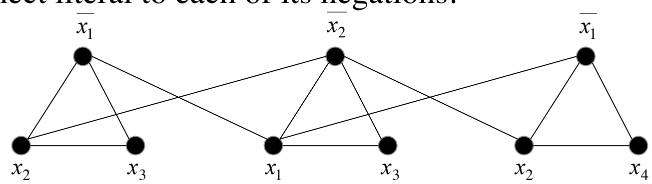
Ex:

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

3 Satisfiability Reduces to Independent Set

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

G

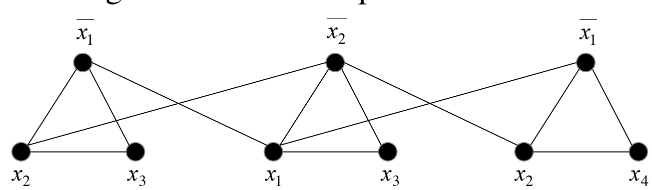


k = 3

3 Satisfiability Reduces to Independent Set

- Claim. G contains independent set of size k = $|\Phi|$ iff Φ is satisfiable.
- Pf. $\Rightarrow$ Let S be independent set of size k.
  - S must contain exactly one vertex in each triangle.
  - Set these literals to true.
  - Truth assignment is consistent and all clauses are satisfied.
- Pf $\Leftarrow$ Given satisfying assignment, select one true literal from each triangle. This is an independent set of size k. ▪

G



k = 3

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 $X \leq_P Z$.
- Pf idea. Compose the two algorithms.

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

NP Completeness - Definition

- Definition of NP-Complete:
  If P is NP-Complete,
1. $P \in$ **NP** and
2. all problems R from **NP** are polynomial-time reducible to P
- Formally: $R \leq_p P \; \forall R \in$ **NP**

- If $P \leq_p Q$ and P is NP-Complete, Q is also NP-Complete
  This is the *key idea* you should take away to be able to proof new problems are NP-complete
  - In order to prove a problem to be NP complete we need to find the first NP complete problem and show all other problems in NP are poly-reducible to it.

Coming Up

- Given one NP-Complete problem, we can prove many interesting problems NP-Complete
- Our first NP-complete problem will be circuit satisfiability:
- Given a boolean circuit, find out whether there is a set of inputs which causes the output to be 1
- After that:
  Graph coloring (= register allocation)
  Hamiltonian cycle, Hamiltonian path
  Knapsack problem, Traveling salesman
  Job scheduling with penalties
  Many, many more

An Aside: Terminology

- *What is the difference between a problem and an instance of that problem?*
- To formalize things, we will express instances of problems as strings
- *How can we express a instance of the hamiltonian cycle problem as a string?*
- To simplify things, we will worry only about *decision problems* with a yes/no answer
- Theory of NP completeness restricted to *decision problems*
- Many problems are *optimization problems*, but we can often re-cast those as decision problems

Optimization vs Decision Problems

• **Optimization problem**: Given a graph G(V,E) determine optimal coloring C(G) such that no two neighboring vertices are colored using the same color

• **Decision problem**: Given G(V,E) and **k**, is there such coloring which uses only k colors ?

• **Optimization problem**: Given a weighted graph, what is the minimum weight cycle which visits each node exactly once ?

• **Decision problem**: Given a weighted graph and integer **k** is there a cycle with weight at most **k** which visits each node exactly once ?

Since decision problems seem easier, if we can show decision problem is hard, then associated optimization problem is also hard

NP class

- NP problems: It is quite easy to check whether given instance is a solution (i.e. given set of vertices is independent set)
- Verification of an instance can be done in polynomial time

Nondeterministic algorithm

1. Guessing a string – try to interpret it as a guess of the solution (string s – is the certificate)
2. Verify whether the string is a solution to the instance of the decision problem – answer true, false, no answer
   (string –e.g. encoding of graph and suggested hamiltonian cycle)
   (general analogy – instances of a problem are strings from a language)
3. Output – if certificate passed (if the answer of the verification true) => answer to the decision problem is YES.

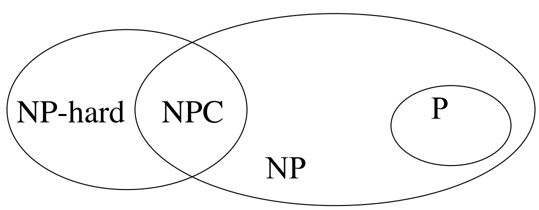
NP-Hard and NP-Complete

- Definition of NP-Hard and NP-Complete:
  If all problems R ∈ **NP** are reducible to P, then P is *NP Hard*
  We say P is *NP-Complete* if
  1. P ∈ **NP**
  2. If R $\leq_p$ P for every R in NP

- If only 2. is satisfied the problem is NP-hard
- A problem Q is **NP – complete** if it is NP-hard and is in NP.

- If P $\leq_p$ Q and P is NP-Complete, Q is also NP- Complete

NP-Hard and NP-Complete



NP complete problems – problems which are in NP but are **as hard as** any other problem in NP

• **NP hard** which are not NP-complete, e.g. halting problem (is still possible to reduce any problem in NP to halting problem)

• Some other non-decision problems

Why Prove NP-Completeness?

- Though nobody has proven that **P != NP**, if you prove a problem NP-Complete, most people accept that it is probably intractable
- Therefore it can be important to prove that a problem is NP-Complete

- Don't need to come up with an efficient algorithm
- Can instead work on *approximation algorithms*
- You can use known algorithm for it and accept that it will take long
- Change your problem formulation

Proving NP-Completeness

- *What steps do we have to take to prove a problem P is NP-Complete?*
- Pick a known NP-Complete problem Q, Reduce Q to P
- Describe a transformation that maps instances of Q to instances of P, s.t. "yes" for P = "yes" for Q
  - Prove the transformation works
  - Prove it runs in polynomial time
  - Prove P $\in$ **NP**

P And **NP** Summary

- **P** = set of problems that can be solved in polynomial time
- **NP** = set of problems for which a solution can be verified in polynomial time
- **P** $\subseteq$ **NP**
- Open question: Does **P = NP**?

- Next our first NP complete problem - Circuit Satisfiability

Circuit Satisfiablity

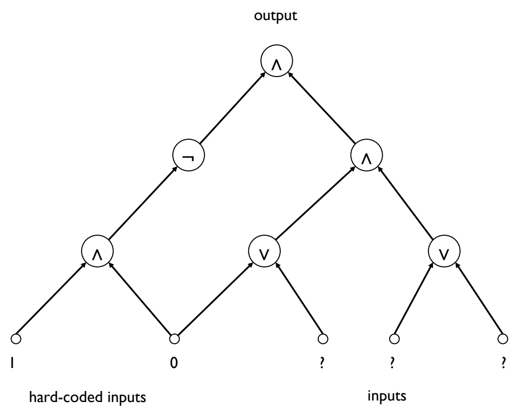
- **Circuit SAT** is NP can be verified in polynomial time
  i.e. given a circuit and an input we can verify in polynomial time whether the input is a satisfying assignment.
- **Circuit SAT is NP-hard** Every problem in NP is reducible to circuit SAT; Proof:

1. Problem is in NP; can be verified in polynomial time by some algorithm
2. Each step of the algorithm runs on a computer (huge boolean circuit)
3. Chaining together all circuits which correspond to the steps of the algorithm – we get large circuit which describes the run of the algorithm
4. If we plug in the input of a problem A then YES / NO answer when circuit is/is not satisfiable

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?

yes: 1 0 1



The "First" NP-Complete Problem

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

•Pf. (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. Moreover, if algorithm takes poly-time, then circuit is of poly-size.

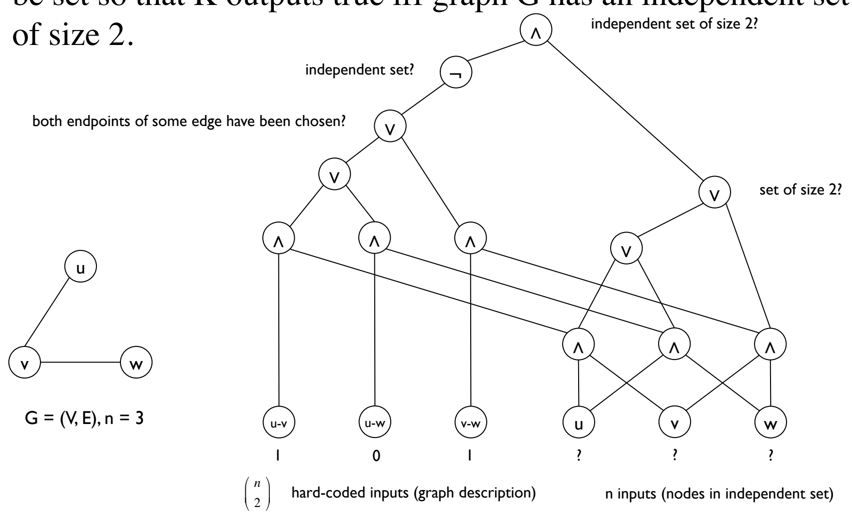
Consider some problem X in NP. It has a poly-time certifier C(s, t). To determine whether s is in X, need to know if there exists a certificate t of length p(|s|) such that C(s, t) = yes. View C(s, t) as an algorithm on |s| + p(|s|) bits (input s, certificate t) and convert it into a poly-size circuit 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

•Ex. 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.



Relationships between known NP – complete problems

Circuit SAT

↓

Formula Satisfiability

↓

3-SAT

↙ ↘ ↘

Independent Set   Hamilton Cycle   3D matching

↙ ↘   ↓   ↘

Vertex Cover   Clique   Traveling Salesmen   knapsack

Formula Satisfiability

- Show that it is easy to verify the solution
- Reduce circuit satisfiability to formula SAT
- Any instance of circuit satisfiability can be reduced to formula satisfiability
- Strategy: express every gate as a formula (Example).

3-CNF Satisfiability

- Show that it is easy to verify the solution
- Reduce Satisfiability to 3-CNF
- Strategy: Get Binary Parse Tree, introduce new variables, get clauses
- Convert Clauses to CNF form using De Morgan's Laws

The 3-CNF Problem

- Thm 36.10: Satisfiability of Boolean formulas in 3-CNF form (the *3-CNF Problem*) is NP-Complete
- The reason we care about the 3-CNF problem is that it is relatively easy to reduce to others
- Thus by proving 3-CNF NP-Complete we can prove many seemingly unrelated problems NP-Complete
- Alternatively reduce Circuit Satisfiability to 3-CNF

3-SAT is NP-Complete

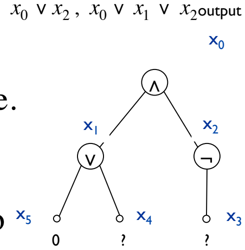
•Theorem. 3-SAT is NP-complete.
•Pf. Suffices to show that CIRCUIT-SAT $\leq_P$ 3-SAT since 3-SAT is in NP. Let K be any circuit.

Create a 3-SAT variable x_i for each circuit element i.
Make circuit compute correct values at each node:

$x_2 = \neg x_3$ ⇒ add 2 clauses: $x_2 \vee x_3$, $\overline{x_2} \vee \overline{x_3}$
$x_1 = x_4 \vee x_5$ ⇒ add 3 clauses: $x_1 \vee \overline{x_4}$, $x_1 \vee \overline{x_5}$, $\overline{x_1} \vee x_4 \vee x_5$
$x_0 = x_1 \wedge x_2$ ⇒ add 3 clauses: $\overline{x_0} \vee x_1$, $\overline{x_0} \vee x_2$, $x_0 \vee \overline{x_1} \vee \overline{x_2}$

$\overline{x_5}$
x_0

Hard-coded input values and output value.
$x_5 = 0$ ⇒ add 1 clause:
$x_0 = 1$ ⇒ add 1 clause:
Final step: turn clauses of length < 3 into clauses of length exactly 3. ▪



Clique NP complete

- Clique problem – subset of vertices in a undirected graph, each pair is connected by an edge. Decision problem: Is there a clique of size k ?

- First show that it is in NP
- Given a set of k vertices we can always check in polynomial time whether they form a clique or not (traverse the adjacency lists of all k vertices

- Show that some known NP-complete problem can be reduced to clique

3-CNF → Clique

- *What is a clique of a graph G?*
- A: a subset of vertices fully connected to each other, i.e. a complete subgraph of G
- The *clique problem*: how large is the maximum-size clique in a graph?
- *Can we turn this into a decision problem?*
- A: Yes, we call this the *k-clique problem*
- Is there a clique of size k in the graph G ?
- *Is the k-clique problem within **NP**?*
- Naïve approach ? Check all possible subsets of k vertices

3-CNF → Clique

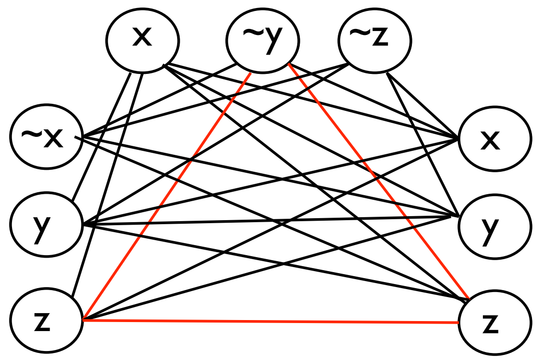
- *What should the reduction do?*
- A: Transform a 3-CNF formula to a graph, for which a k-clique will exist (for some k) iff the 3-CNF formula is satisfiable

3-CNF → Clique

- The reduction:
- Let $B = C_1 \wedge C_2 \wedge \ldots \wedge C_k$ be a 3-CNF formula with k clauses, each of which has 3 distinct literals
- For each clause put a triple of vertices in the graph, one for each literal
- Put an edge between two vertices if they are in different triples and their literals are *consistent*, meaning not each other's negation
- Run an example:
  $B = (x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee z) \wedge (x \vee y \vee z)$
- See example in the book (page 1005)

3-CNF → Clique

B = (x ∨ ¬y ∨ ¬z) ∧ (¬x ∨ y ∨ z) ∧ (x ∨ y ∨ z)



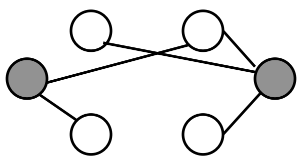
- Put an edge if the variables are consistent
- Formula is satisfiable if each literal has at least ona variable T
- If respective variables are non-conflicting they form a clique

3-CNF → Clique

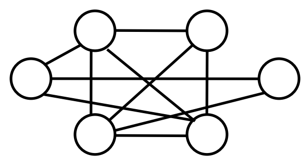
- Prove the reduction works:
- If B has a satisfying assignment, then each clause has at least one literal (vertex) that evaluates to 1
- Picking one such "true" literal from each clause gives a set V' of *k* vertices. V' is a clique (*Why?*)
- If G has a clique V' of size k, it must contain one vertex in each triple (clause) (*Why?*)
- We can assign 1 to each literal corresponding with a vertex in V', without fear of contradiction

Clique → Vertex Cover

- A *vertex cover* for a graph G is a set of vertices incident to every edge in G
- The *vertex cover problem*: what is the minimum size vertex cover in G?
- Restated as a decision problem: does a vertex cover of size *k* exist in G?
- Thm 36.12: vertex cover is NP-Complete

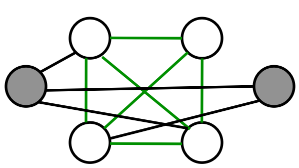


Example of vertex cover of size 2

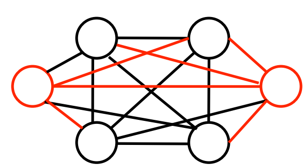


Clique → Vertex Cover

- First, show vertex cover in **NP** (*How?*)
- How to decide whether graph G has a vertex cover of size *k*
- Reduce *k*-clique to vertex cover
- The *complement* G_C of a graph G contains exactly those edges not in G
- Compute G_C in polynomial time
- G has a clique of size *k* iff G_C has a vertex cover of size |V| - *k*



Clique of size 4



Vertex cover of size 2

Clique → Vertex Cover

- In order to show that it is a correct reduction need to Prove the claim both ways
- Claim: If G has a clique of size k, G_C has a vertex cover of size $|V| - k$

1. Let V' be the k-clique
2. Then V - V' is a vertex cover in G_C
3. Let (u,v) be any edge in G_C
4. Then u and v cannot both be in V' (*Why?*)
5. Thus at least one of u or v is in V-V' (*why?*), so edge (u, v) is covered by V-V'
6. Since true for *any* edge in G_C, V-V' is a vertex cover

Clique → Vertex Cover

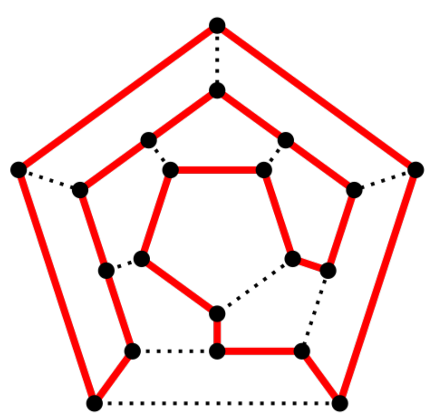
- Claim: If G_C has a vertex cover $V' \subseteq V$, with $|V'| = |V| - k$, then G has a clique of size k

- For all $u,v \in V$, if $(u,v) \in G_C$ then $u \in V'$ or $v \in V'$ or both (*Why?*)
- Contrapositive: if $u \notin V'$ and $v \notin V'$, then $(u,v) \in E$
- In other words, all vertices in V-V' are connected by an edge, thus V-V' is a clique
- Since $|V| - |V'| = k$, the size of the clique is k

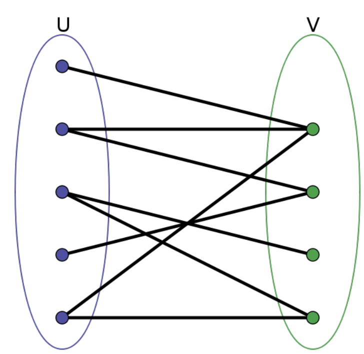
Hamiltonian Cycle ⇒ TSP

- Vertex cover => hamiltonian cycle (see book).
- The well-known *traveling salesman problem*:
- Optimization variant: a salesman must travel to *n* cities, visiting each city exactly once and finishing where he begins. How to minimize travel time?
- Model as complete graph with cost c(*i*,*j*) to go from city *i* to city *j*
- *How would we turn this into a decision problem?*
  A: ask if ∃ a TSP with cost < *k*

Hamiltonian Cycle



Hamiltonian cycle exists



Bipartite graph with odd number of Vertices - hamiltonian cycle does not exists

Hamiltonian path

Is there a path which passes though all points exactly once ?
(similar then traveling salesman with no costs).

If you can visit some of the vertices more then once,
But each edge exactly once >> simpler **Eulerian path**

Eulerian path is simpler because we can clearly relate the condition on the graph which must be satisfied in order for the solution to exist (then we can just check the property)

E.g. if the graph is connected and number of edges emanating from any point is even (except two points) then you can do it

Hamiltonian Cycle ⇒ TSP

- The steps to prove TSP is NP-Complete:
- Prove that TSP ∈ **NP** (*Argue this*)

- Reduce the undirected hamiltonian cycle problem to the TSP
- So if we had a TSP-solver, we could use it to solve the hamiltonian cycle problem in polynomial time

- *How can we transform an instance of the hamiltonian cycle problem to an instance of the TSP? Can we do this in polynomial time?*

The TSP

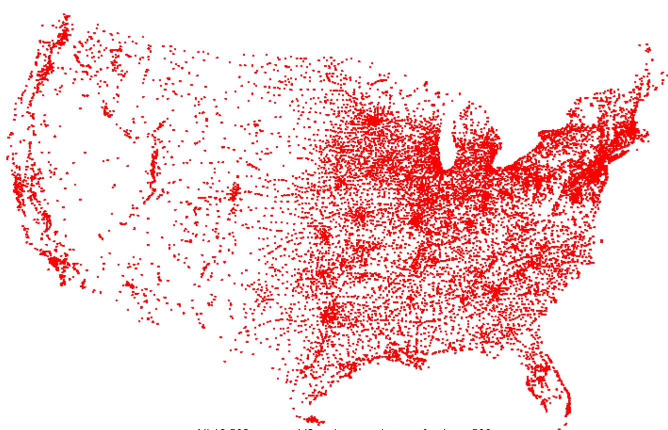
- Random asides:

- TSPs (and variants) have enormous practical importance
- E.g., for shipping and freighting companies
- Lots of research into good approximation algorithms

- Drilling n-holes into VLSI board

Traveling Salesperson Problem

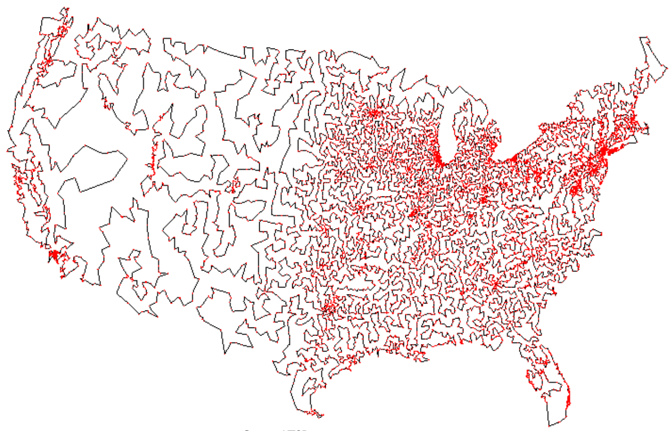
- TSP. Given a set of n cities and a pairwise distance function d(u, v), is there a tour of length ≤ D?



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 ≤ D?



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

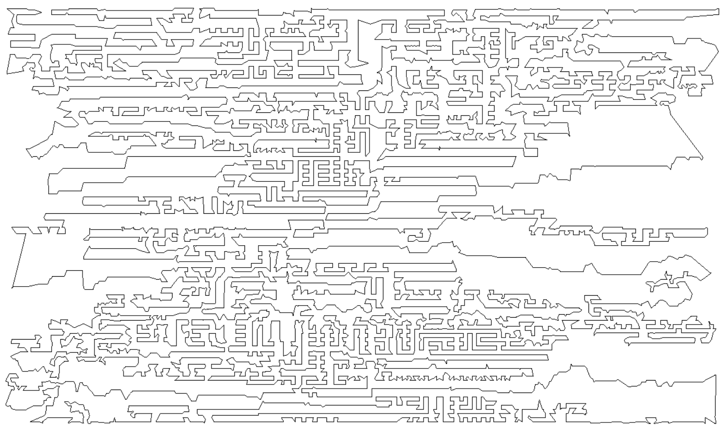
Traveling Salesperson Problem

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



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

General Comments

- Literally hundreds of problems have been shown to be NP-Complete
- Some reductions are profound, some are comparatively easy, many are easy once the key insight is given
- You can expect a simple NP-Completeness proof on the final

Other NP-Complete Problems

- *Subset-sum*: Given a set of integers, does there exist a subset that adds up to some target *T*?
- *0-1 knapsack*: when weights not just integers
- *Hamiltonian path*: Obvious
- *Graph coloring*: can a given graph be colored with *k* colors such that no adjacent vertices are the same color?
- Etc…

Directed Hamiltonian Cycle ⇒ Undirected Hamiltonian Cycle

- *What was the hamiltonian cycle problem again?*
- For my next trick, I will reduce the *directed hamiltonian cycle* problem to the *undirected hamiltonian cycle* problem before your eyes
  - *Which variant am I proving NP-Complete?*
- Draw a directed example on the board
  - *What transformation do I need to effect?*

Transformation: Directed ⇒ Undirected Ham. Cycle

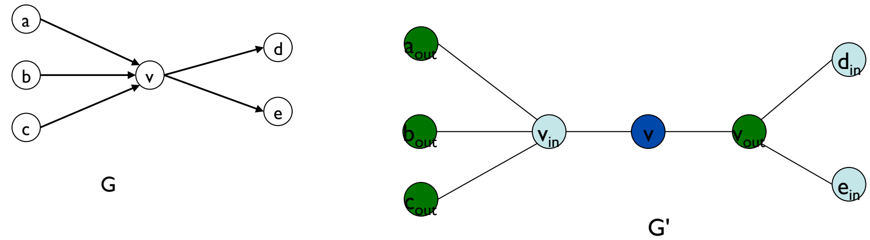
- Transform graph G = (V, E) into G' = (V', E'):
  Every vertex v in V transforms into 3 vertices
  v^1, v^2, v^3 in V' with edges (v^1,v^2) and (v^2,v^3) in E'
  Every directed edge (v, w) in E transforms into the
  undirected edge (v^3, w^1) in E' (draw it)
  Can this be implemented in polynomial time?
  Argue that a directed hamiltonian cycle in G implies an undirected hamiltonian cycle in G'
  Argue that an undirected hamiltonian cycle in G' implies a directed hamiltonian cycle in G

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.

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



Transformation: Directed ⇒ Undirected Ham. Cycle

- Transform graph G = (V, E) into G' = (V', E'):
  Every vertex v in V transforms into 3 vertices
  v^1, v^2, v^3 in V' with edges (v^1,v^2) and (v^2,v^3) in E'
  Every directed edge (v, w) in E transforms into the
  undirected edge (v^3, w^1) in E' (draw it)
  Can this be implemented in polynomial time?
  Argue that a directed hamiltonian cycle in G implies an undirected hamiltonian cycle in G'
  Argue that an undirected hamiltonian cycle in G' implies a directed hamiltonian cycle in G

Undirected Hamiltonian Cycle

- Thus we can reduce the directed problem to the undirected problem
- *What's left to prove the undirected hamiltonian cycle problem NP-Complete?*
- *Argue that the problem is in* ***NP***

Directed ⇒ Undirected Ham. Cycle

- Given: directed hamiltonian cycle is NP-Complete (draw the example)
- Transform graph G = (V, E) into G' = (V', E'):
  - Every vertex v in V transforms into 3 vertices v^1, v^2, v^3 in V' with edges (v^1,v^2) and (v^2,v^3) in E'
  - Every directed edge (v, w) in E transforms into the undirected edge (v^3, w^1) in E' (draw it)

Directed ⇒ Undirected Ham. Cycle

- Prove the transformation correct:
  - If G has directed hamiltonian cycle, G' will have undirected cycle (straightforward)
  - If G' has an undirected hamiltonian cycle, G will have a directed hamiltonian cycle
    - The three vertices that correspond to a vertex v in G must be traversed in order v^1, v^2, v^3 or v^3, v^2, v^1, since v^2 cannot be reached from any other vertex in G'
    - Since 1's are connected to 3's, the order is the same for all triples. Assume w.l.o.g. order is v^1, v^2, v^3.
    - Then G has a corresponding directed hamiltonian cycle

Hamiltonian Cycle ⇒ TSP

- The well-known *traveling salesman problem*:
  Optimization variant: a salesman must travel to n cities, visiting each city exactly once and finishing where he begins. How to minimize travel time?
  Model as complete graph with cost $c(i,j)$ to go from city i to city j
- *How would we turn this into a decision problem?*
  A: ask if $\exists$ a TSP with cost $< k$

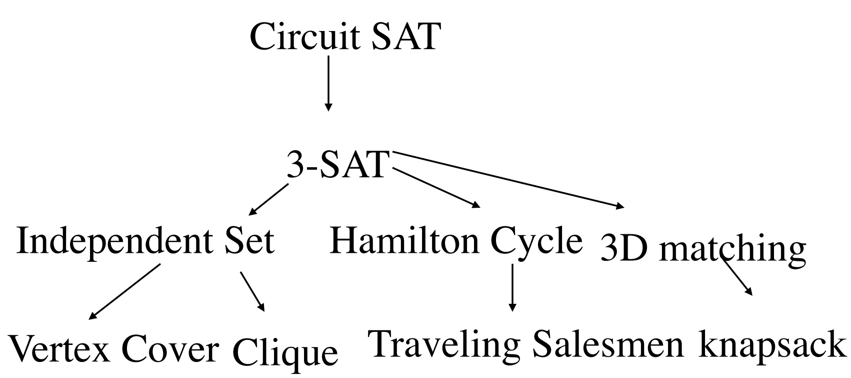
Hamiltonian Cycle ⇒ TSP

- The steps to prove TSP is NP-Complete:
  Prove that TSP $\in$ **NP** (*Argue this*)
- Reduce the undirected hamiltonian cycle problem to the TSP
  So if we had a TSP-solver, we could use it to solve the hamiltonian cycle problem in polynomial time
- *How can we transform an instance of the hamiltonian cycle problem to an instance of the TSP?*
- *Can we do this in polynomial time?*

Review: Hamiltonian Cycle ⇒ TSP

- To transform ham. cycle problem on graph
  G = (V,E) to TSP, create graph G' = (V,E'):
  - G' is a complete graph
  - Edges in E' also in E have weight 0
  - All other edges in E' have weight 1
  - TSP: is there a TSP on G' with weight 0?
    - If G has a hamiltonian cycle, G' has a cycle w/ weight 0
    - If G' has cycle w/ weight 0, every edge of that cycle has weight 0 and is thus in G. Thus G has a ham. cycle

Relationships between known NP – complete problems



Other NP-Complete Problems

- *Subset-sum*: Given a set of integers, does there exist a subset that adds up to some target T?
- *0-1 knapsack*: when weights not just integers
- *Hamiltonian path*: Obvious
- *Graph coloring*: can a given graph be colored with k colors such that no adjacent vertices are the same color?
- Etc…

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.

Extent and Impact of NP-Completeness

- Extent of NP-completeness. [Papadimitriou 1995]
  - Prime intellectual export of CS to other disciplines.
  - 6,000 citations per year (title, abstract, keywords).
    - more than "compiler", "operating system", "database"
  - Broad applicability and classification power.
  - "Captures vast domains of computational, scientific, mathematical endeavors, and seems to roughly delimit what mathematicians and scientists had been aspiring to compute feasibly."
- NP-completeness can guide scientific inquiry.
  - 1926: Ising introduces simple model for phase transitions.
  - 1944: Onsager solves 2D case in tour de force.
  - 19xx: Feynman and other top minds seek 3D solution.
  - 2000: Istrail proves 3D problem NP-complete.