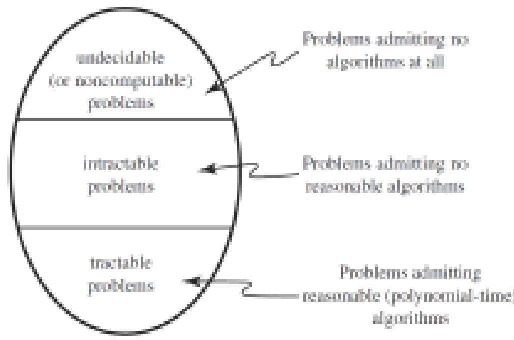
Tractability

- Not all problems that can be solved *in principle* can be solved *in practice*
- A problem is said to be *intractable* if we cannot solve it in a "reasonable amount" of time; otherwise the problem is *tractable*



2

Two "Similar" Problems

- Given a set S of n points in the plane
  1. Find a circuit of minimal length that visits each point exactly once (the robot tour problem)
  2. Find a spanning tree of minimum weight, i.e., a set of segments incident on S such that the resulting graph is connected and the sum of the lengths of the segments is minimized
- Both problems require that we select an optimal set of segments from an exponential set of choices and both can be modeled using the language of graph theory
- Problem 1 is not known to be tractable while problem 2 can be easily solved in $O(n \log n)$ time

3

Motivational Problem 1

1) $9 \times 14 =$ **126**

2) $91 =$ **7×13**

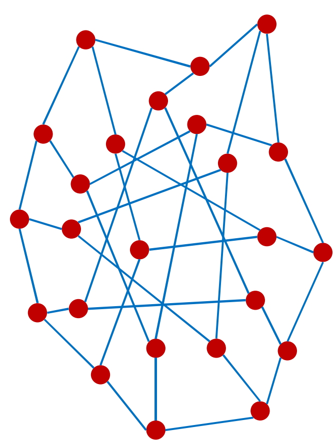
4359585683259407917999519653872144063854709102652201963187054821445240853452759997402446252554284559445 79 ✖ 5625457617268841037562770073044474817438769440075105451049468510945483965774794734721462285507993229392 73 = 2452466449002782119765176635730880184670267876783327597434144517150616008300385872?695220839933207154910362682719167986407977672324300560059203563124656121846581790410013185929961993381701214933503487587055106 7

1796949159794106673291612844957324615636756180801260007088891883553172646034149093349337224786865075523085586419992922181443668472287405206525793749569434838926317115252252565441098081917061174250970244071801036483163828851885268 9 = ? ✖ ?

4

Motivational Problem 2

- You are planning housing for 400 university applicants. Space is limited and only 100 students will receive places in the dormitory. To complicate matters, the Dean has provided you with a list of pairs of incompatible students, and asked that no pair from this list appear in your final choice.

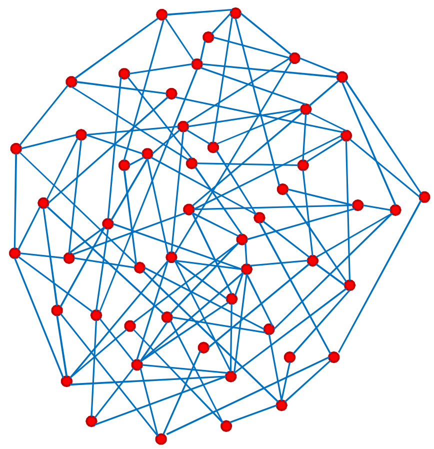


- How would you find 100 compatible students?
  - How do you model and solve the problem?

5

Motivational Problem 3

- You are organizing a party and want to invite five friends from your group of *n* friends. You would like all invitees to be friends among themselves. If you know who is a friend of who, how do you find your target list?



6

Motivational Problem 4

- In constraint satisfaction systems you need to determine if a set of specifications is *consistent* (can be made simultaneously true, i.e., their conjunction is *satisfiable*)
  1. "The diagnostic message is stored in the buffer or it is retransmitted."
  2. "The diagnostic message is not stored in the buffer."
  3. "If the diagnostic message is stored in the buffer, then it is retransmitted."
  4. "The diagnostic message is not retransmitted"

If p:="The diagnostic message is stored in the buffer" and q:="The diagnostic message is retransmitted", the above can be written as: $p \vee q, \neg p, p \rightarrow q, \neg q$.
Specifications $\{1,2,3\}$ are consistent but $\{1,2,3,4\}$ are not.

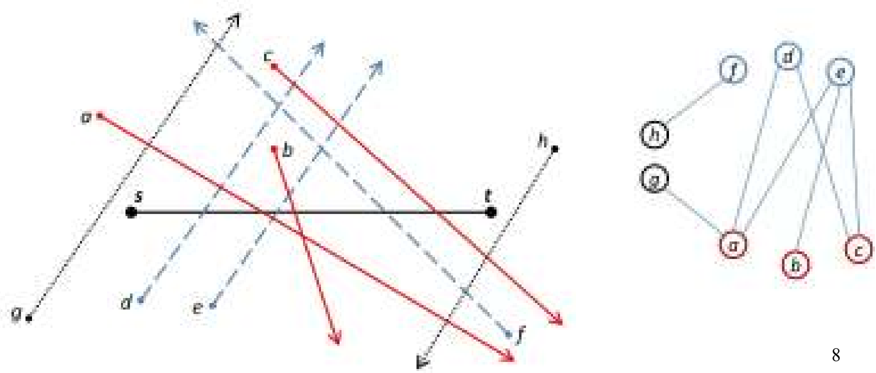
- Is a set of constraints *satisfiable*?

$$(p \vee q) \wedge (\neg p) \wedge (\neg p \vee q) \wedge (\neg q)$$

Motivational Problem 5

- Given a set of sensors (e.g., rays or disks), can an intruder get from point s to point t, undetected, after removing at most k sensors?

- Draw and edge between every pair of *blocking* sensors



8

Decision Problems

- What do the previous problems have in common?
  - They are all ***decision problems*** (the output is **yes** or **no**)
  - Easy to write a program to find a solution but finding it seems to require some type of exhaustive search that may take a long time
  - Most efficient solution is not radically more efficient than the naïve one based on exhaustive search
  - A ***candidate solution*** can be efficiently verified, so once you have a solution it is easy to convince others that you do
- The $\mathbf{P} = \mathbf{NP}$ question asks "Can we solve problems of this type without exhaustive searching?"

  Question: is the related ***optimization problem*** harder to solve?

9

Tractability…

- A problem is ***intractable*** if no reasonable (poly-time) algorithm exists for solving it
- A problem is ***tractable*** if it can be solved in polynomial time
  - OK, but is $\Omega(n^{1000})$ practical?
- In order to simplify our study of tractability we concentrate on *decision problems* (problems with yes/no answer)

  Is there a robot tour of length less than 5,000?

10

Ubiquitous Intractability

- Which problems will we be able to solve in practice?

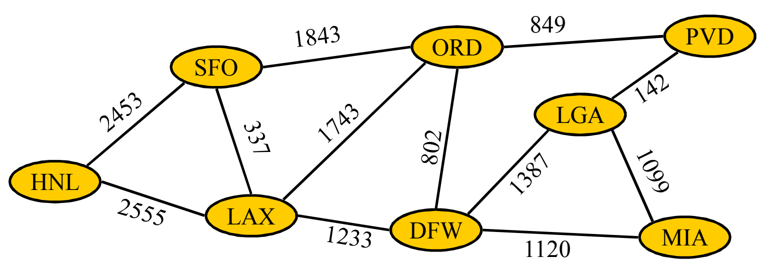
Yes	Probably no
Shortest path	Longest path
2D-Matching	3D-matching
Min cut	Max cut
2-SAT	3-SAT
Planar 4-color	Planar 3-color
Eulerian Tour	Hamiltonian Tour
Primality testing	Factoring

- Will consider the notion of intractability through the notions of NP-hardness and NP-completeness

11

The Class NP

- A *decision problem* is in NP if a *candidate solution* can be verified in polynomial time



- Which of the following is in NP?
  – *Hamiltonian path*: is there a path that visits each vertex once?
  – *Traveling salesman path*: is there a Hamiltonian path of cost ≤ C?
  – *Eulerian path*: is there a path that visits each edge once?

12

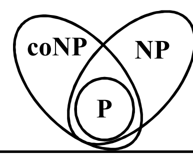
A Formal Definition of NP

- Consider a decision problem X. $X \in$ **NP** if it satisfies:
  1. If the answer to an instance x of X is **yes**, a ***certificate*** can be provided to verify this in polynomial time
  2. If the answer is **no**, then no such certificate for x exists
- Algorithm $C(s,t)$ is a ***certifier*** for X if for every instance string s, $X(s) =$ yes iff there exists a string t such that $C(s,t) =$ yes
- The class **NP** consists of all decision problems that admit a polynomial time certifier
- *Open problem*: is **P** ≠ **NP** ?

13

Tractability Classes

- **P** is the set of decision problems that can be solved in polynomial time.
- **NP** is the set of decision problems that can be verified in polynomial time, i.e., have the property: if the answer to an instance is Yes, then there is a certificate of this fact that can be verified in polynomial time.
- **coNP** is the set of decision problems with the property: if the answer to an instance is No, then there is a certificate of this fact that can be verified in polynomial time.



14

Reductions

- Would like to classify problems according to *relative* difficulty
- If X is tractable, what other problems are tractable?
- If X is intractable, what other problems are intractable?
- If you don't know, how do you gather evidence to support a conjecture that Π is intractable?
  - Even if we don't know the answer to **P≠NP** question, we would still like some guidance about which problems are difficult and which ones are not
  - One approach is to show that problems are hard *in a relative sense*, by showing that a problem is at least as hard as other hard problems
  - The tool to accomplish this is a *polynomial-time reduction*

15

Reduction

- A reduces to B, denoted $A \leq B$, if an algorithm for B can be used as a subroutine for solving A.

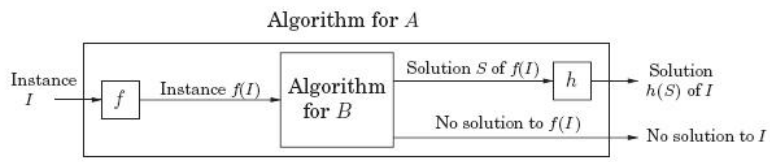
<u>Examples</u>:

sorting a list ≤ computing the convex hull of a set of points

computing the convex hull ≤ triangulation of a set of points

determining if m and n are relatively prime ≤ GDC of m and n

# of walks from u to v in a graph $G(V, E)$ ≤ matrix multiplication



- We are interested in the case where f runs in polynomial time. We then write $A \leq_P B$

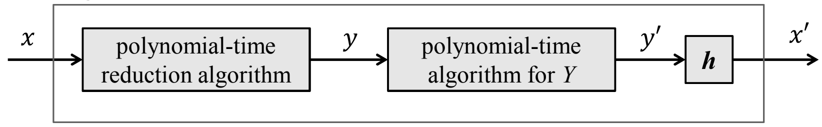
16

Polynomial Time Reductions

Definition. A problem X ***polynomially reduces*** to problem Y, denoted $X \leq_P Y$, if given a polynomial-time algorithm for Y, you can use it to solve X in polynomial time.

- Reduction is often enacted as follows:
  - Convert binary input x of X to a binary string $f(x)$ in polynomial time $p(n)$. What is $|f(x)|$?
  - Solve Y on input $y = f(x)$ and use the answer y' to compute the answer x' for X on input x

Algorithm for X :



Notation: $T_X(n) \leq O(p(n)) + T_Y(O(p(n)))$

17

Examples of $\leq_P$

1. All-pairs shortest path $\leq_P$ Single-source shortest path
2. Find-median $\leq_P$ Sorting
3. Sorting $\leq_P$ Convex-Hull
4. Sorting $\leq_P$ Single-source-shortest-path (in a graph)
5. Determining if a graph is a tree $\leq_P$ Depth-first Search
6. Maximum-independent-set $\equiv_P$ minimum-vertex-cover

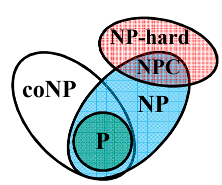
- *Reusability*. Seasoned algorithm designers are always looking for opportunities to employ reductions
- What does $Y \leq_P X$ tell us about the relative difficulty of the two problems?

18

The Class NPC

- A problem is *NP-complete* (in NPC) if
  1. It is a member of NP
  2. Every other problem of NP *reduces* to it in polynomial time
- Thousands of *practical* problems are NP-complete
  - Networks
  - VLSI circuit design
  - Geometry
  - Data storage and retrieval
  - Scheduling
  - Games and puzzles
  - Logic
  - Operations research

- A problem is *NP-hard* if every problem of NP *reduces* to it in polynomial time



19

Using Reductions

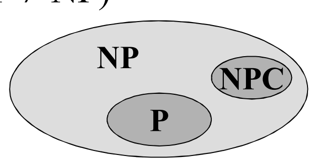
- To prove that problem *X* is NP-hard, reduce (in polynomial time) a known NP-hard problem to *X*
- In other words, to prove that your problem is hard, you need to describe an algorithm to solve a *different* problem, one already known to be hard, using a hypothetical algorithm for *your* problem as a subroutine.

20

Facts about P, NP, NPC

- There are thousands of NP-complete problems, drawn from a wide variety of fields: mathematics, computer science, geography, engineering, finance
- No polynomial-time algorithm has been found for any NPC problem
- No proof that a polynomial algorithm does not exist for any of NPC problems has been found
- Most theoretical computer scientists believe that NPC is intractable (i.e., P $\neq$ NP)

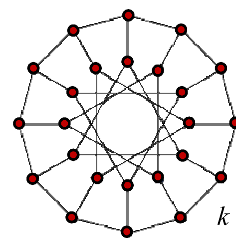
$P \subset NP$,
$NPC \subset NP$,
$P \cap NPC = \varnothing$



21

A Sample of Problems in NPC

- Hamiltonian circuit
  - Does the undirected graph *G* have a circuit of length *n*?
- Traveling salesman (the robot tour problem!)
  - Does the complete graph *G* have a Hamiltonian circuit of length at most *L*?
- Partition
  - Can you partition a set of *n* integers (e.g., {2,3,6,7,9,11}) into two subsets of equal sum?
- Independent set
  - Does a graph *G* have a subset of at least *k* vertices no two of which are neighbors?

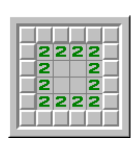


$k = 9$

22

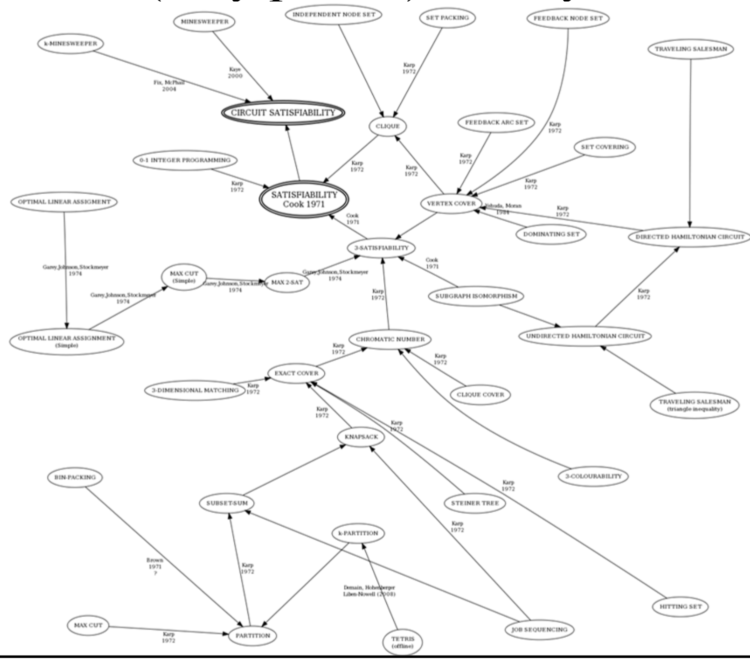
A Sample of Problems in NPC…

- 3-SAT
  - CNF-Satisfiability where each clause has 3 literals
- Sequencing to minimize tardy tasks
  - Tasks are partially ordered. Each has a durations and a deadline. Can you finish at least *k* tasks on time?
- Bin packing
  - Given *k* disks of capacity *x* and *m* files of sizes $x_1,\ldots, x_m$, can you copy all the files into the disks?
- $N \times N$ checkers
  - Can white win given the current game configuration?
- Minesweeper
  - Is a configuration consistent?



23

NPC (very partial) Family Tree



24

Formula Satisfiability (SAT)

- An instance of **SAT** is a Boolean formula ϕ composed of:
  - n Boolean variables: $x_1, x_2, \ldots, x_n$.
  - m connectives: any Boolean function with one or two inputs and one output, such as $\wedge, \vee, \neg, \rightarrow, \leftrightarrow$
  - Parentheses for overriding default precedence

  Example. $\phi = \left((x_1 \rightarrow x_2) \vee \neg\left((\neg x_1 \leftrightarrow x_3) \vee x_4\right)\right) \wedge \neg x_2$
- Instance ϕ is ***satisfiable*** if there exists a truth assignment that forces ϕ to evaluate to 1
- **SAT**=$\{\langle\phi\rangle$: ϕ is a satisfiable Boolean formula$\}$

  Example. ϕ above $\in$ **SAT**, use $x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 1$

25

Cook-Levin Theorem: SAT is NPC

Proof (Cook 1971, Levin 1973):

1. **SAT** $\in$ NP.
   - Certificate is truth assignment t
   - Given truth assignment t, the certifier replaces each variable with its value and evaluates the formula in polynomial time.
2. **SAT** is NP-hard
   - We will prove this next quarter by showing that every problem in NP reduces to **SAT**

- **SAT** provides us with a *first* NPC problem
- From now on, can use reductions to show that other problems are NPC

26

SAT is NPC…

- Cook's & Levin's brilliant insight was that any algorithm that takes a fixed # of bits as input and outputs a *yes*/*no* answer can be encoded as a Boolean formula
- Formula evaluates to **1** on precisely the inputs for which the algorithm outputs ***yes***
- If the algorithm takes a polynomial number of steps, the formula has polynomial size
- We are interested in the case where the algorithm (and resulting formula) is the certifier for a problem $X \in \mathbf{NP}$
- Again, details omitted until next quarter

27

NP-Completeness User Guide

- *Remember!*
  - If Z is a problem such that $X \leq_p Z$ for some $X \in$ NPC, then Z is NP-hard. If, additionally, $Z \in$ NP, then $Z \in$ NPC
- Steps to prove that Z is NP-complete
  - Prove $Z \in$ NP.
    - A certificate can be verified in poly time.
  - Prove Z is NP-hard.
    - Select a known NP-complete problem X
    - Describe a transformation function f that maps an arbitrary instance x of X into an instance $f(x)$ of Z
    - Prove f satisfies: for all input instances x of X, the answer to $x \in X$ is YES iff the answer to $f(x) \in Z$ is YES
    - Prove that the algorithm for f runs in polynomial time

28

3-CNF

- A *special case* of SAT useful for proving NPC results
- Definitions
  - A *literal* in a Boolean formula is an occurrence of a variable or its negation.
  - CNF (Conjunctive Normal Form) is a boolean formula expressed as *conjunction* of *clauses*, each of which is the *disjunction* of one or more literals.
  - 3-CNF is a CNF in which each clause has *exactly* 3 *distinct* literals (*Note*: a literal and its negation are distinct)

$$\overbrace{(a \vee b \vee c \vee d)}^{\text{clause}} \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b})$$

- Goal: determine if a 3-CNF formula is satisfiable

  3-CNF=$\{\langle \phi \rangle : \phi$ is a satisfiable 3-CNF formula$\}$

29

3-CNF is NP-complete

Proof: 3-CNF $\in$NP: Easy.

- 3-CNF is NP-hard. (we show SAT $\leq_p$ 3-CNF)
- The proof is broken into 4 steps, each of which transforms the input instance ϕ of SAT into a formula closer to 3-CNF

SAT $\longrightarrow C \rightarrow T_1 \rightarrow \phi_2 \rightarrow \phi_3 \rightarrow \phi_4 \longleftarrow$ 3-CNF

1) Rewrite C using an expression tree with every node degree ≤ 2. If any node has $k > 2$ inputs, replace it with a binary tree of $k - 1$ nodes
2) Rewrite T_1 as conjunction with one clause per node, introducing new variables for internal nodes
3) Change every clause into CNF form.

   There are only three types of clauses, one per internal node type:

   $a \Leftrightarrow b \wedge c \longmapsto (a \vee \neg b \vee \neg c) \wedge (\neg a \vee b) \wedge (\neg a \vee c)$

   $a \Leftrightarrow b \vee c \longmapsto (\neg a \vee b \vee c) \wedge (a \vee \neg b) \wedge (a \vee \neg c)$

   $a \Leftrightarrow \neg b \longmapsto (a \vee b) \wedge (\neg a \vee \neg b)$

4) Change 1- and 2-literal clauses into 3-literal clauses

   $(a \vee b) \longmapsto (a \vee b \vee p) \wedge (a \vee b \vee \neg p)$

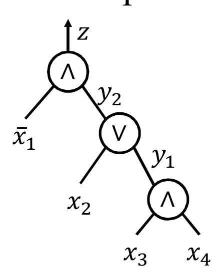
   $a \longmapsto (a \vee p \vee q) \wedge (a \vee p \vee \neg q) \wedge (a \vee \neg p \vee q) \wedge (a \vee \neg p \vee \neg q)$

30

Exercise

- Show the 3CNF reduction for the following SAT formula. We show the first two steps.

$\overline{x_1} \wedge (x_2 \vee (x_3 \wedge x_4))$



$(y_1 \leftrightarrow x_3 \wedge x_4) \wedge (y_2 \leftrightarrow x_2 \vee y_1) \wedge (z \leftrightarrow \bar{x}_1 \wedge y_2) \wedge z$

- Complete the reduction!

31

3-CNF is NP-complete

- Recall: $C \to C_1 \to \phi_2 \to \phi_3 \to \phi_4$
- C and reduced 3-CNF formula ϕ_4 are equivalent:
  - C to ϕ_2 preserves equivalence
  - ϕ_2 to ϕ_3 preserves equivalence
  - ϕ_3 to final 3-CNF ϕ_4 preserves equivalence
- Reduction takes polynomial time
  - From $C \to C_1$ you less than double the number of gates
  - From $C_1 \to \phi_2$, you produce as many clauses as gates
  - From ϕ_2 to ϕ_3, each iff-clause becomes 2 or 3 clauses
  - From ϕ_3 to ϕ_4, each iff-clause becomes 2 or 3 clauses

32

CNF-Satisfiability

- A *literal* is a Boolean variable or its negation
- A *clause* is the disjunction of one or more literals
- A Boolean expression is in *conjunctive normal form* (CNF) if it is the conjunction of clauses

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

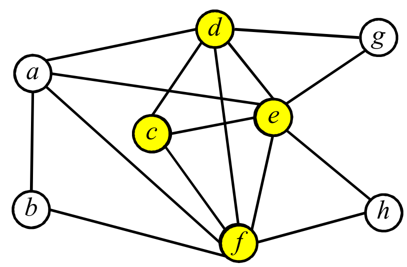
- *CNF*: given a Boolean expression in CNF is there a truth assignment that makes the formula true?
- *CNF* $\in$ NP

Theorem If $X \in$ NP then $X \leq_P$ *CNF*

33

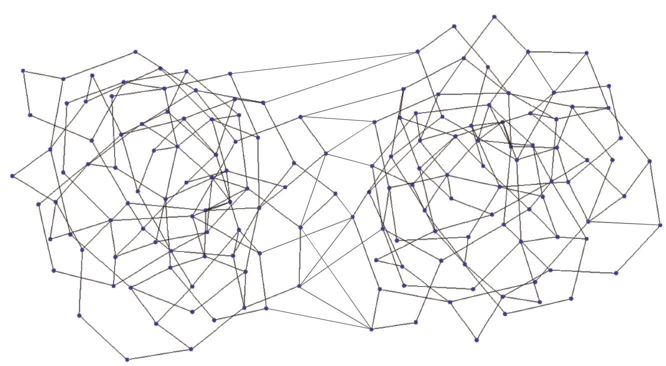
The Clique Problem

- A *clique* of a graph $G(V, E)$ is a subset W of V such that every pair of vertices of W is connected by an edge in E.
- *CLIQUE problem*: Given a graph $G(V, E)$ and positive integer k, does G contain a clique with at least k vertices?



34

Does this graph contain a clique of size 4?



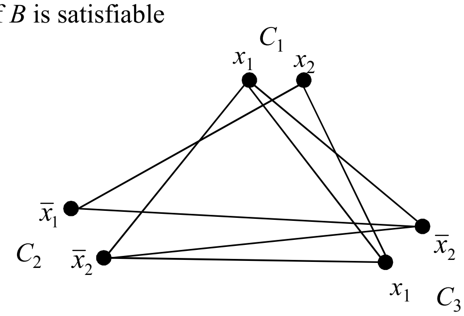
- How long does this take?
- How long would it take to find the largest clique?

35

Clique $\in$ NPC

- We show $CNF \leq_P CLIQUE$
- Let B be a CNF formula with k clauses. We construct G
  - For each literal in each clause of B we add a vertex to G
  - There is an edge of G between two vertices iff they arise from different clauses and they are not complementary
  - G has a clique of size k iff B is satisfiable
  - $CLIQUE \in$ NP

$$\underbrace{(x_1 \vee x_2)}_{C_1} \wedge \underbrace{(\overline{x}_1 \vee \overline{x}_2)}_{C_2} \wedge \underbrace{(x_1 \vee \overline{x}_2)}_{C_3}$$

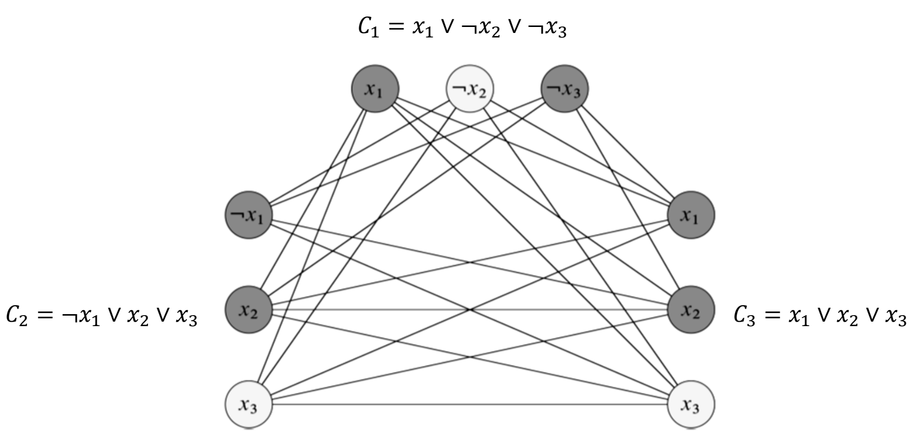


Theorem. The Clique problem is NP-complete

36

Example

$\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$ and reduced graph G



37

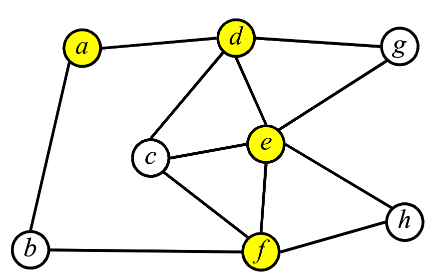
Reduction Correctness

- If ϕ is satisfiable, then there exists a truth assignment with at least one true literal per clause
  - Choose a true literal in each clause and consider the subgraph W comprised of the corresponding vertices of chosen literals
  - For each pair $v_i^r, v_j^s \in W$ $v_i^r, v_j^s \in V'$, with $r \neq s$, since both l_i^r, l_j^s l_i^r, l_j^s are true, we know that l_i^r is not the negation of $l_i^r, l_j^s \Rightarrow$ there is an edge between v_i^r and v_j^s and W is a clique of size k
- If G has a clique W of size k, then W contains exactly one vertex from each triple. Assign true to all the literals corresponding to the vertices in W, and false to other literals. Then each clause evaluates to true $\Rightarrow$ ϕ is satisfiable
- It is easy to see the reduction runs in poly-time

38

The Vertex Cover Problem

- A *vertex cover* of a graph $G(V, E)$ is a subset W of V so that every edge of E has at least one vertex in W
- *Question*: given a graph $G(V,E)$, is there a vertex cover with at most k vertices?
- Many applications, e.g., resilience in sensor networks

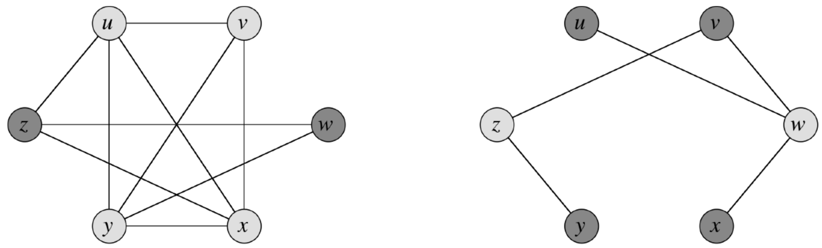


- We show *Clique* $\leq_P$ *Vertex Cover*

39

Vertex Cover ∈ NPC

- $G(V, E)$ has a clique W **iff** $\bar{G}(V, \bar{E})$ has vertex cover $V - W$



(a) (b)

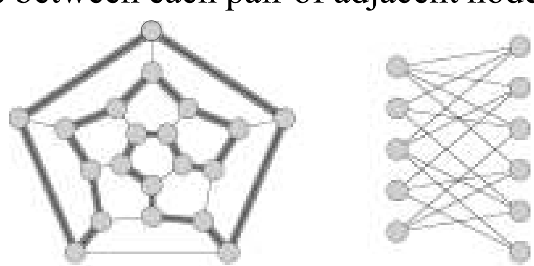
Theorem. The Vertex Cover problem is NP-complete

40

The Hamiltonian Cycle Problem

Ham-Cycle$\langle G(V, E)\rangle$. Does graph G contain a simple cycle that visits every node?

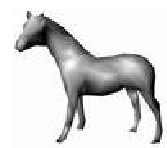
- Certificate. A permutation π of the n nodes
- Certifier checks that π is a permutation with an edge between each pair of adjacent nodes in π



a) Certificate shown in bold
b) No certificate exists ⇒ ∄ Hamiltonian Cycle

Rendering Triangular Meshes

- Triangular meshes are the most common model used for representing 3D objects in computer graphics.
- Hardware optimized for shading and rendering one triangle at a time
- Bottleneck resides in transferring the geometric data to the GPU

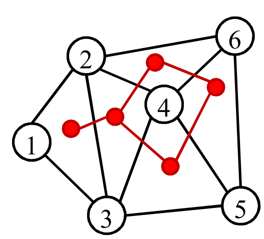


42

OpenGL Triangle Strips

- *Goal*: specify most triangles with just one additional vertex

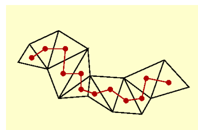
```
Point3d strip[n];
glBegin(GL_TRIANGLE_STRIP);
    for(int i=0; i<n; i++)
        glVertex2fv(strip[i]);
glEnd();
```

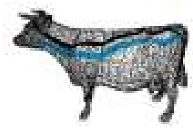


43

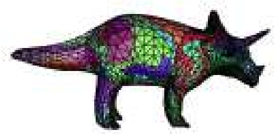
Goal

- Render model using one triangle strip.
  - Can this always be done?





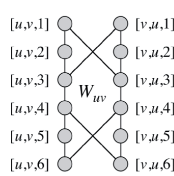
- Partition model into the smallest possible number of triangle strips?
  - Can this always be done?



44

Ham-Cycle is NP-Complete

- To show **Vertex-Cover** $\leq_P$ **Ham-Cycle** we convert an arbitrary instance of $G(V, E)$ of **Vertex-Cover** to an instance $G'(V', E')$ of **Ham-Cycle** such that G has a vertex cover of size k <u>iff</u> G' has a hamiltonian cycle
- Proof uses special purpose "widgets" for the edges of E
  - Only vertices $[*,*,1]$ and $[*,*,6]$ have outside edges

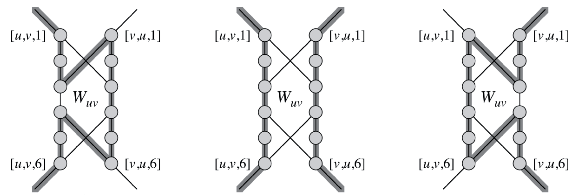


$\{u, v\} \in E \rightarrow W_{uv} \subset E'$

45

Edge Widgets

- *Widgets* in G' are used to enforce properties of G
  - Since only vertices $[*,*,1]$ and $[*,*,6]$ connect outside W, any Hamiltonian cycle of G' must traverse W in one of 3 ways



- If a cycle enters through $[u, v, 1]$, it must exit through $[u, v, 6]$ and visit all or half of the vertices of W
- For each $u \in V$, let $u^{(1)}, \dots, u^{(\deg u)}$ be the neighbors of u in G in arbitrary order. Then, we add edges to form a path ρ_u through all widgets that correspond to edges $\{u, u^{(i)}\}$ of G, in given order

$$\{[u, u^{(i)}, 6], [u, u^{(i+1)}, 1]\}, 1 \le i < \deg(u)$$

46

Edge Widgets…

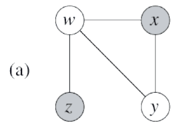
- If $u \in V$ is in the cover, then ρ_u traces a path from $[u, u^{(1)}, 1]$ to $[u, u^{(\deg u)}, 6]$ that "covers" all widgets of edges incident on u
- For each widget W_{uv}, then ρ_u visits all 12 vertices if u is in the cover but v is not, and 6 if both are in the cover
- There are k selector vertices $s_1, s_2, \dots, s_k$
- The final type of edge joins the first vertex $[u, u^{(1)}, 1]$ and the last vertex $[u, u^{(\deg u)}, 6]$ to each of the selector vertices

Theorem. The size of G' is polynomial in the size of G.

Theorem. G has a vertex cover of size k <u>iff</u> G' has a Hamiltonian cycle

47

Example

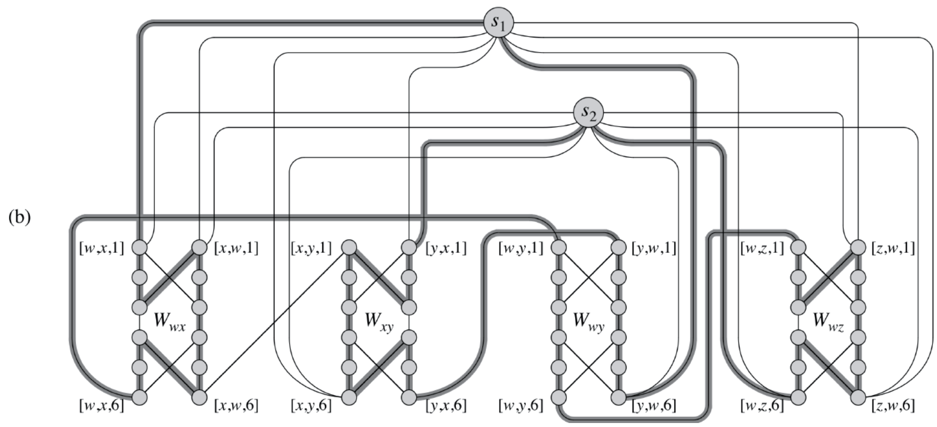


Adjacencies: $z: w$

$w: x, y, z$

$x: w, y$

$y: x, w$

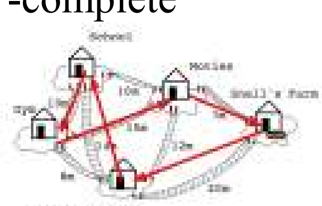


48

Traveling-salesman problem is NPC

- **TSP**=$\{\langle G, c, k\rangle$:
  $G(V, E)$ is a complete graph,
  c is a function from $V \times V \to \mathbb{Z}$,
  $k \in \mathbb{Z}$, and G has a traveling salesman tour with cost at most $k\}$
- *Theorem.* **TSP** is NP-complete

Ham−Cycle $\leq_P$ TSP



49

The Subset Sum Problem

- Given set S of positive integers and a target t, can you find a subset $P \subseteq S$ such that the numbers in P add up to t ?

Example: $S = \{1, 4, 16, 64, 256, 1040, 1041, 1093, 1284, 1344\}$ and $t = 3754$

Yes! 1+16+64+256+1040+1093+1284 = 3754

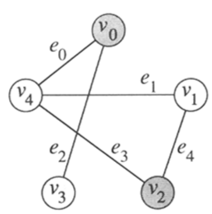
Theorem. The Subset Sum problem is NP-complete

Reduction from Vertex Cover

50

VC $\leq_P$ Subset-Sum

- Need to reduce an instance I of vertex cover to an instance J of subset sum such that the answer to I can be computed from the answer to J



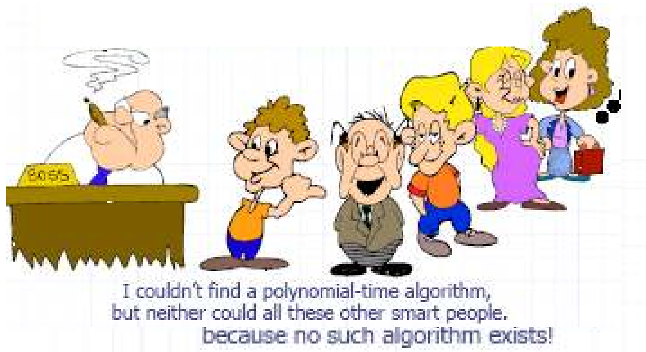
$$x_i = 4^m + \sum_{j=0}^{m-1} b_{ij} 4^j, \quad y_j = 4^j$$

$$t = k4^m + \sum_{j=0}^{m-1} 2\left(4^j\right)$$

			modified base 4						decimal
			e_4	e_3	e_2	e_1	e_0		
x_0	=	1	0	0	1	0	1	=	1041
x_1	=	1	1	0	0	1	0	=	1284
x_2	=	1	1	1	0	0	0	=	1344
x_3	=	1	0	0	1	0	0	=	1040
x_4	=	1	0	1	0	1	1	=	1093
y_0	=	0	0	0	0	0	1	=	1
y_1	=	0	0	0	0	1	0	=	4
y_2	=	0	0	0	1	0	0	=	16
y_3	=	0	0	1	0	0	0	=	64
y_4	=	0	1	0	0	0	0	=	256
t	=	3	2	2	2	2	2	=	3754

51

What should you do?



52

Approximation Algorithms

- When faced with an optimization problem whose decision version is NP-complete, a reasonable goal is to design an algorithm that finds an answer close to the optimal, i.e., an *approximation algorithm*
- Trades loss of accuracy for better running time
- Usually comes with quality guarantee

  Example: 1.5-approximation means answer is at most 50% worse than the optimal (in practice, may be much better)
- Why settle for less?
  - Exact solution may take too long
  - Approximate answer may be the first step in finding optimal answer

53

Approximation Ratios

- Let π be a minimization problem. We say that algorithm $\mathcal{A}$ for π which, given instance I, returns a solution with value $\mathcal{A}(I)$. The approximation ratio of $\mathcal{A}$ is defined as:

$$\alpha_{\mathcal{A}} = \max_{I} \frac{\mathcal{A}(I)}{OPT(I)}$$

- Similarly, if π is a maximization problem and $\mathcal{B}$ and algorithm for π, the approximation ratio of $\mathcal{B}$ is defined as:

$$\alpha_{\mathcal{B}} = \max_{I} \frac{OPT(I)}{\mathcal{B}(I)}$$

- An algorithm is said to be an γ-approximation algorithm if it has approximation ratio γ

54

Approximate Shortest Robot Tours

- Nearest neighbor tour yields approximation:

$$\frac{NN}{OPT} \leq \frac{1}{2}\left(\lceil \log n + 1 \rceil\right)$$

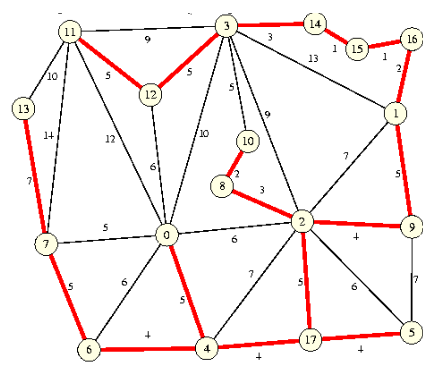
- Can do much better using MST-based approximation $\frac{MST}{OPT} \leq 2$

- Can improve by augmenting MST with *minimum-weight perfect matching* of odd degree vertices

55

Minimum Spanning Tree

- Let $G(V,E)$ be a connected undirected graph
- A subgraph of G is *spanning* if it is connected and has the same vertex set as G
- A *minimum spanning tree* of G is a spanning subgraph of G of smallest total cost
- Can be found efficiently in $O(m + n \log n)$ time

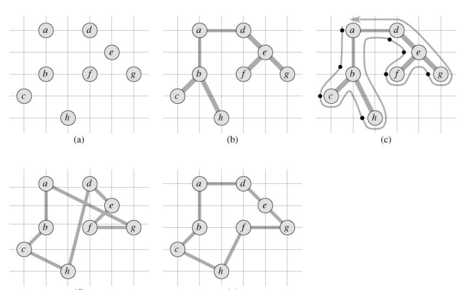


56

2-Approximation Based on MST

Definition. Let $G(V,E)$ be a weighted connected graph. A *minimum spanning tree* of G is a connected subgraph $H(V,F)$, where $F \subseteq E$, of minimum weight

1. Construct MST of G
2. Double each edge, in 2 directions
3. Perform an Euler tour
4. Short-circuit already visited vertices



- Because of triangle inequality, short-circuiting a vertex cannot increase the cost. Thus, $MST \leq OPT \Rightarrow MST\text{-}TOUR \leq 2 \cdot MST \leq 2 \cdot OPT$

57