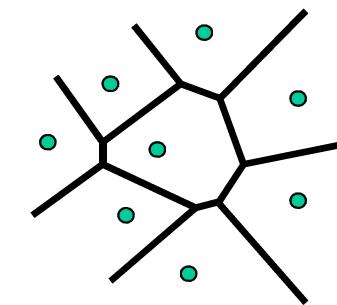
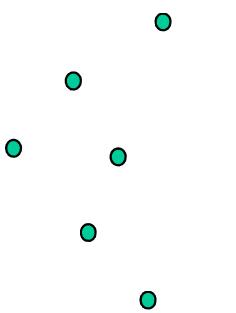
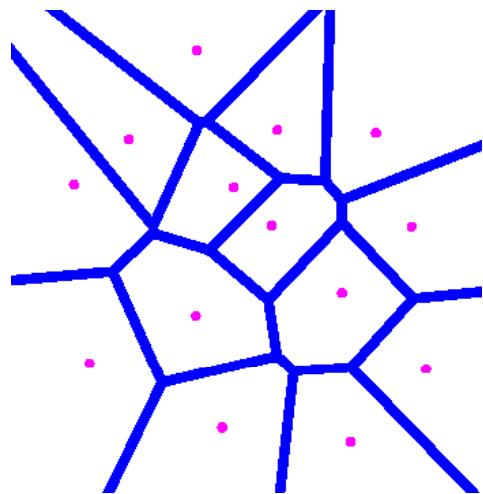


Voronoi Diagrams and Delaunay Triangulations

Voronoi Diagrams



Voronoi diagrams



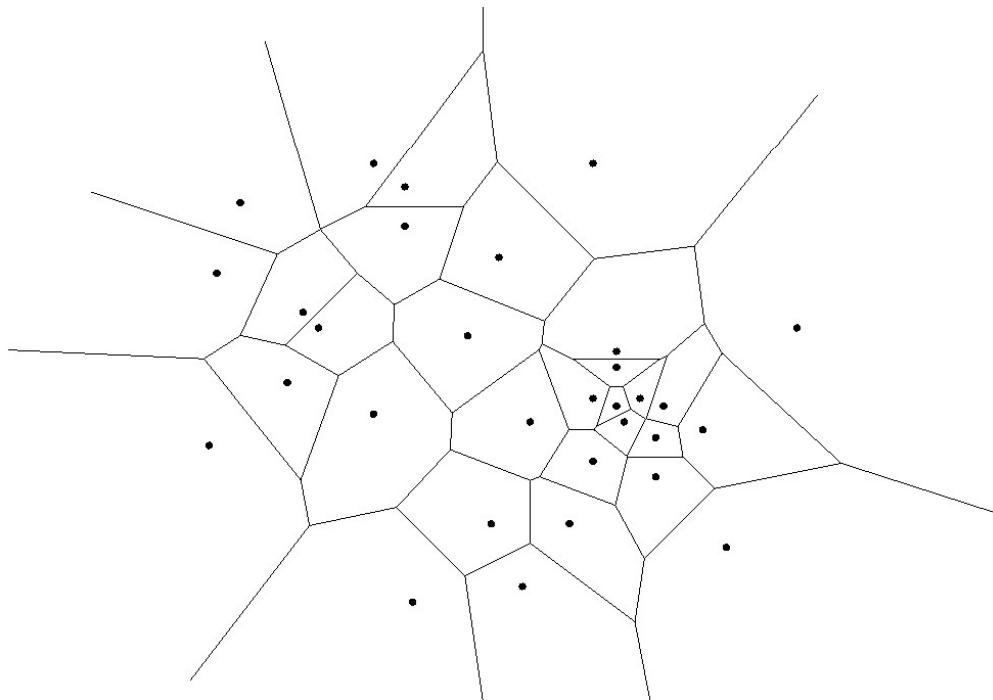
- Given a set of n points.
- Split the space into regions of points closest to the given points.

Applications

- **Nearest-neighbor queries:** Given a point, find which region it belongs.
- **Site assignment:** Given a set of hospital locations, decide which is the nearest hospital for each home.

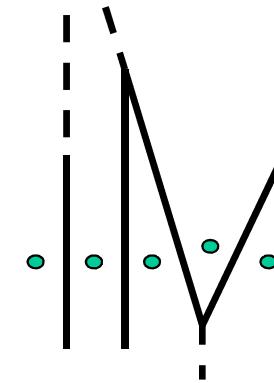
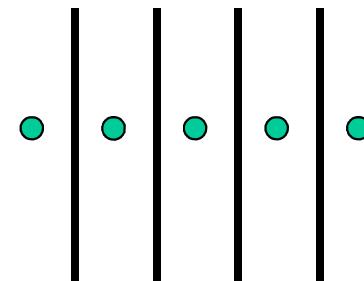
Voronoi diagrams and clustering

- A Voronoi diagram stores proximity among points in a set



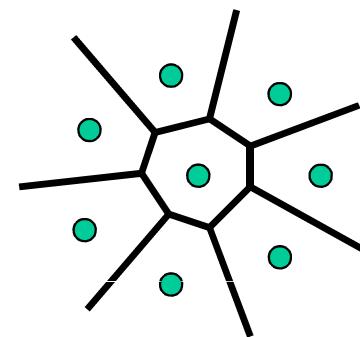
Voronoi Diagram: Corner cases

- If all the sites are colinear, the Voronoi diagram will look like this:
- Otherwise, the diagram is a connected planar graph, in which all the edges are line segments or rays



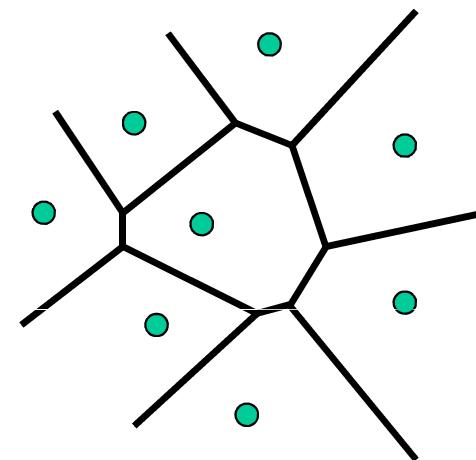
Voronoi Diagram Complexity

- A Voronoi diagram of n distinct sites contains n cells.
- One cell can have complexity $n-1$, but not all the cells can.
 - The number of vertices $V \leq 2n-5$
 - The number of edges $E \leq 3n-6$
 - The number of faces $F = n$



Voronoi Diagram Algorithms

- **Input:** A set of points locations (*sites*) in the plane.
- **Output:** A planar subdivision into cells. Each cell contains all the points for which a certain site is the closest.



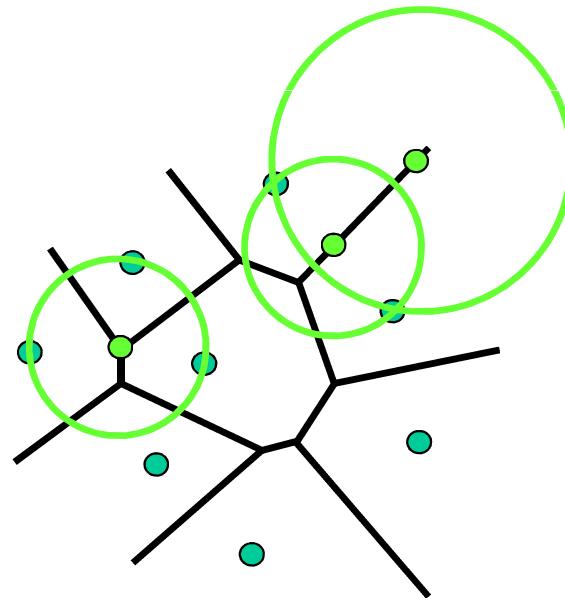
Computing Voronoi diagrams

- Naïve, brute force - edge bisectors.
- By plane sweep.
- By randomized incremental construction
- By divide-and-conquer

All are $O(n \log n)$ time except brute force

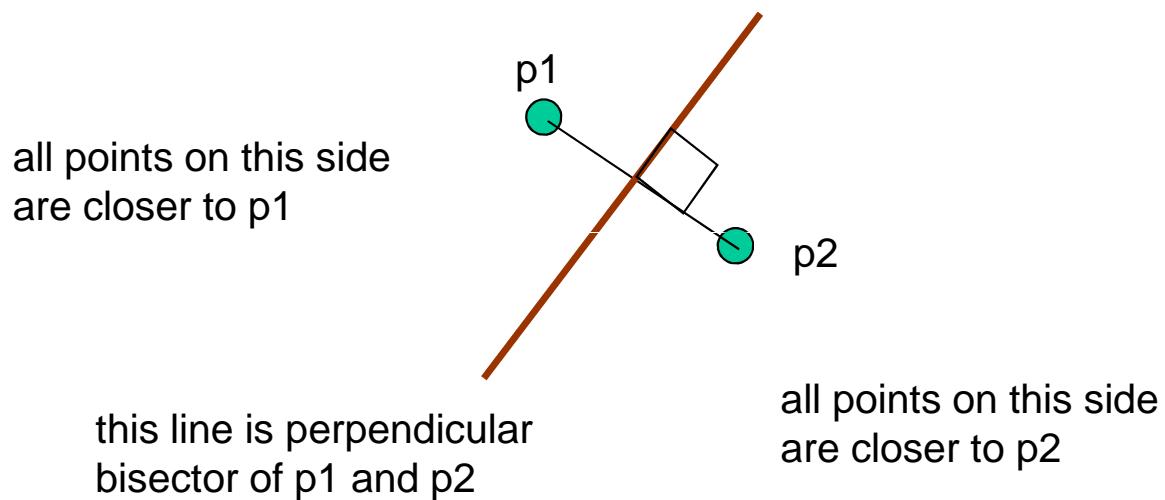
Voronoi Diagram Properties

- A vertex of a Voronoi diagram is the center of a circle passing through three sites.
- Each point on an edge is the center of a circle passing through two sites.



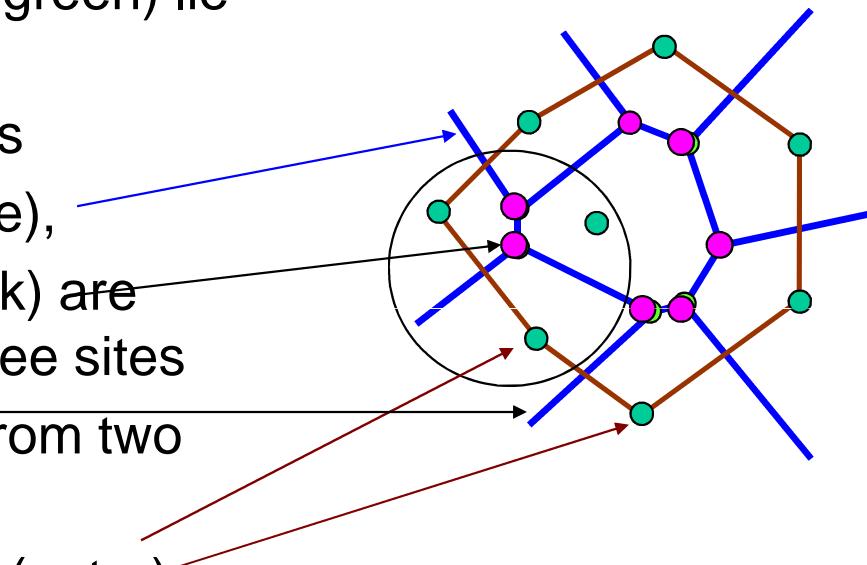
Voronoi Diagram: naive algorithm

- The *V bisector* of two points is a line.



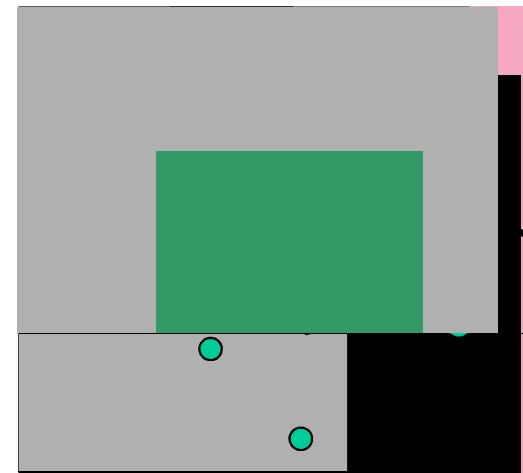
Voronoi Diagram: naïve algorithm

- Assume no four sites (green) lie on a circle.
- The Voronoi diagram is
 - a planar-graph (blue),
 - whose vertices (pink) are equidistant from three sites
 - edges equidistant from two sites.
- The **convex hull** of the (outer) sites are those who have an unbounded cell.



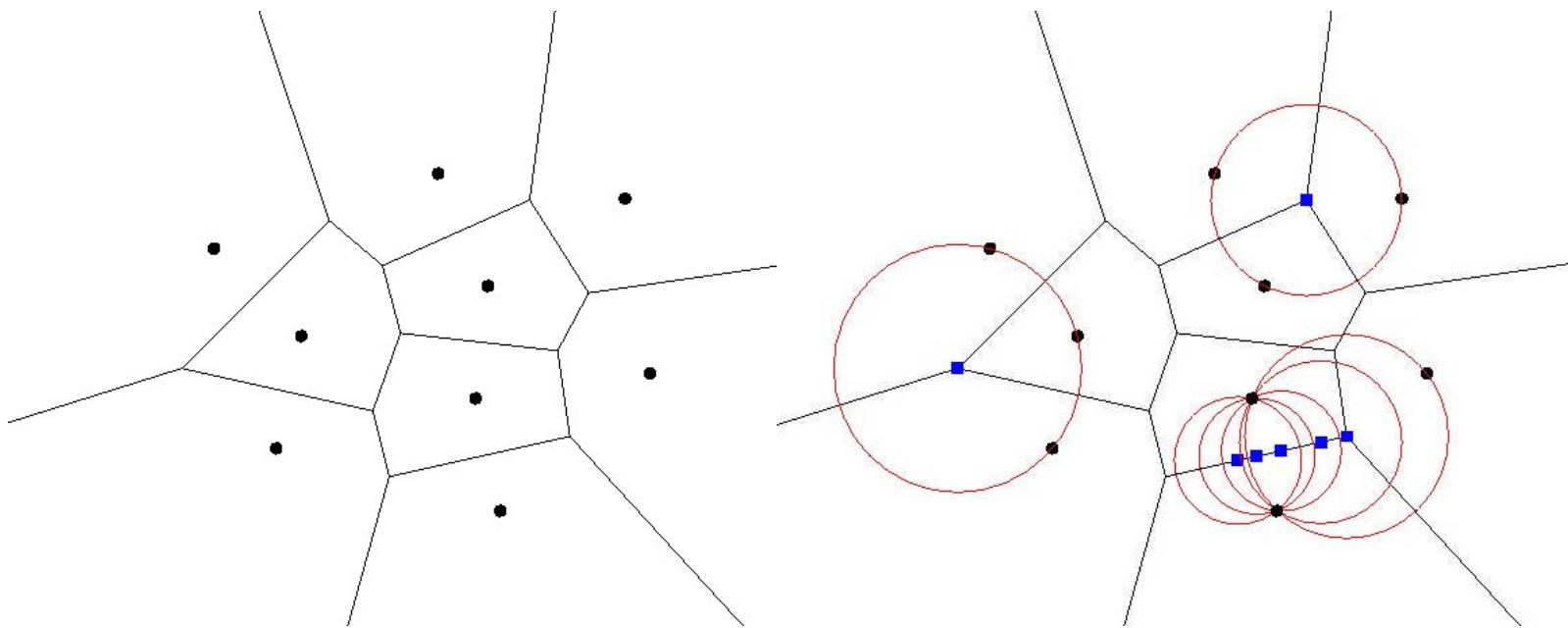
Voronoi Diagram – Naïve algorithm

- Construct bisectors between each site and all others (n^2).
- A **Voronoi cell** is the intersection of all half-planes defined by the bisectors.
- **Corollary:** Each cell in a Voronoi diagram is a convex polygon, possibly unbounded.
- **Time complexity:** $O(n^2 \log n)$



Computing Voronoi diagrams

- Study the geometry, find properties
 - 3-point empty circle \leftrightarrow Voronoi vertex
 - 2-point empty circle \leftrightarrow Voronoi edge

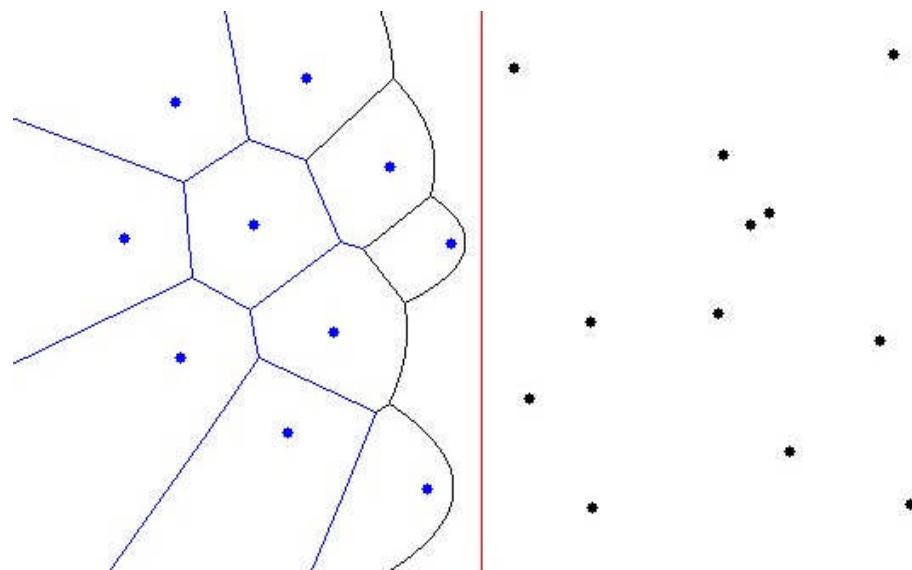


Computing Voronoi diagrams

- Some geometric properties are needed, regardless of the computational approach
- Other geometric properties are only needed for some approach

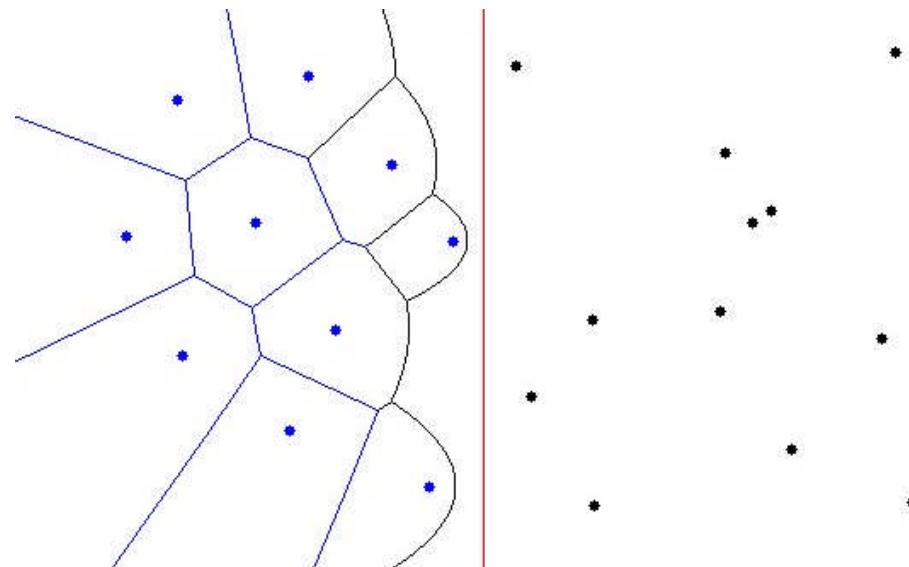
Computing Voronoi diagrams

- Fortune's sweep line algorithm (1987)
 - An imaginary line moves from left to right
 - The Voronoi diagram is computed while the known space expands (left of the line)



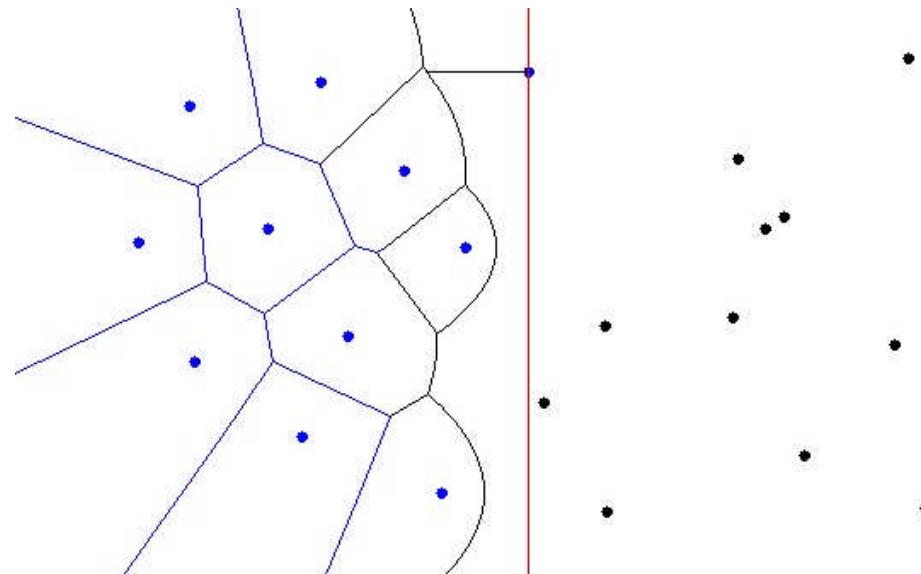
Computing Voronoi diagrams: Beach front

- Beach line: boundary between known and unknown → sequence of parabolic arcs
 - Geometric property: beach line is y-monotone
→ it can be stored in a balanced binary tree



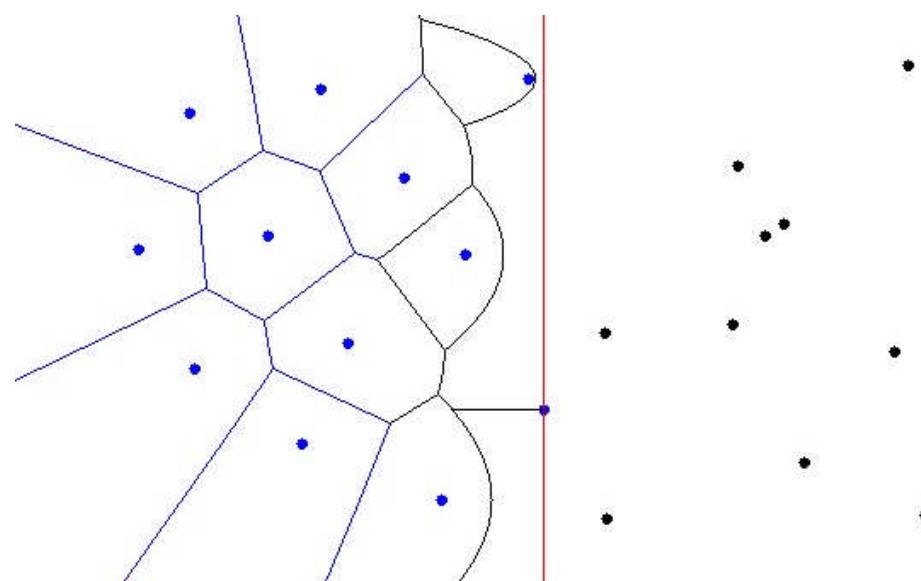
Computing Voronoi diagrams

- Events: changes to the beach line = discovery of Voronoi diagram features
 - Point events



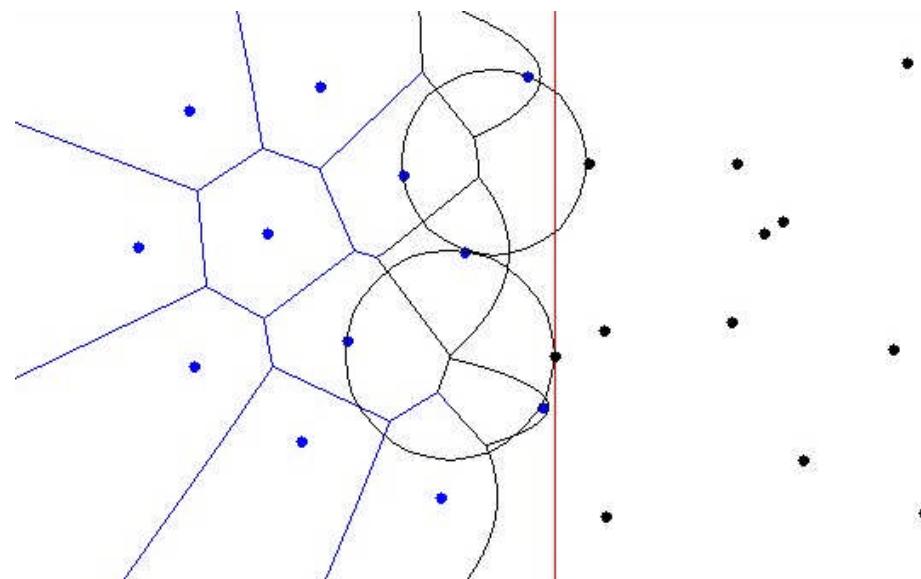
Computing Voronoi diagrams

- Events: changes to the beach line = discovery of Voronoi diagram features
 - Point events



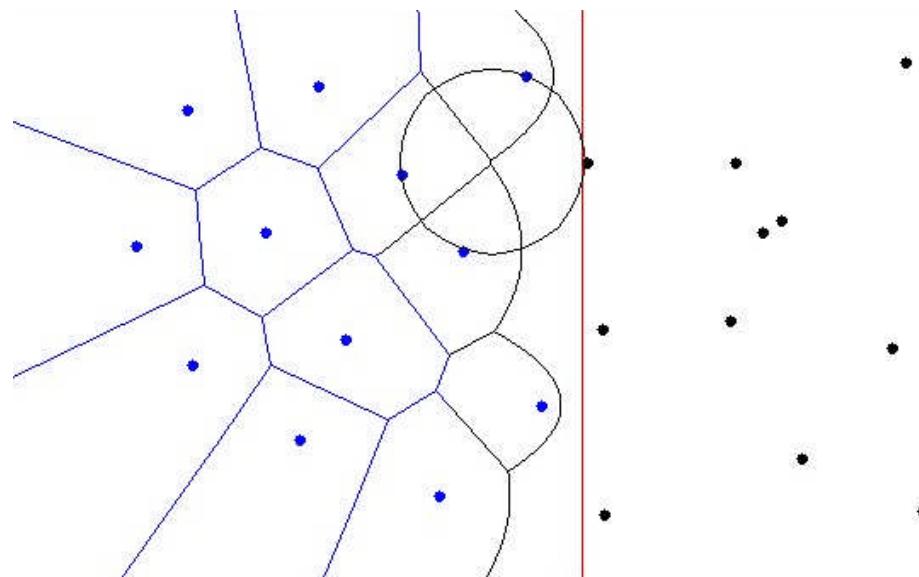
Computing Voronoi diagrams

- Events: changes to the beach line = discovery of Voronoi diagram features
 - Circle events



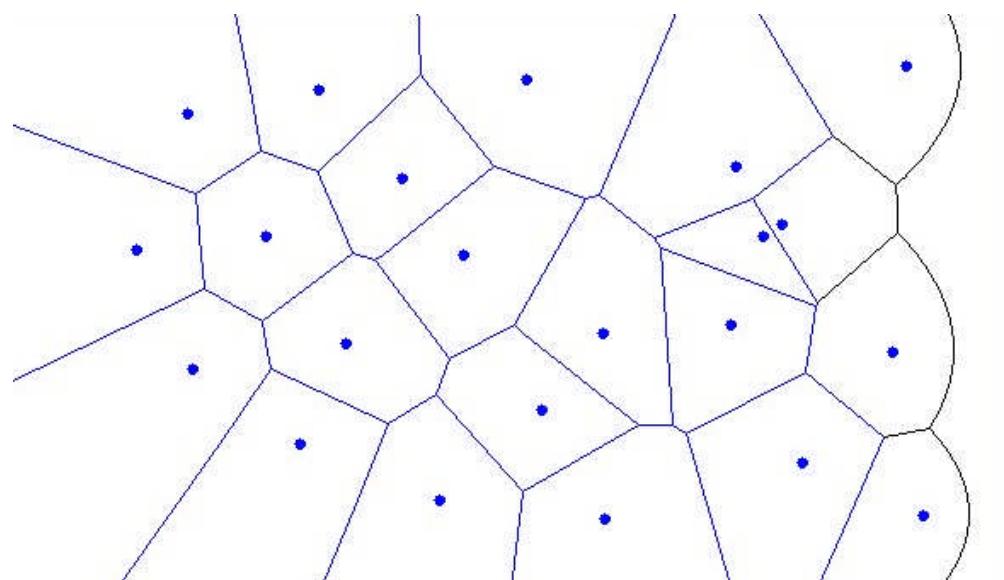
Computing Voronoi diagrams

- Events: changes to the beach line = discovery of Voronoi diagram features
 - Circle events



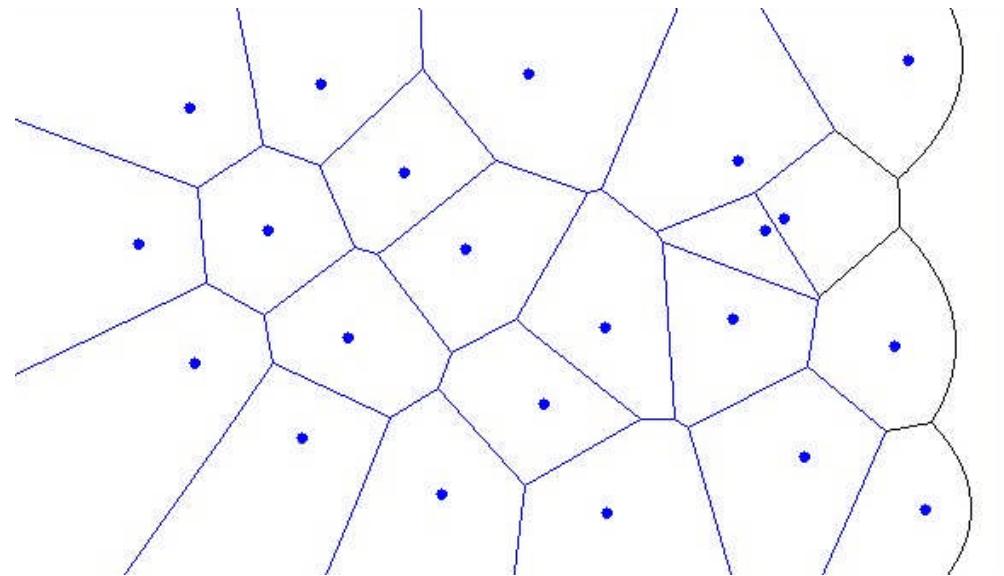
Computing Voronoi diagrams

- Events: changes to the beach line = discovery of Voronoi diagram features
 - Only point events and circle events exist



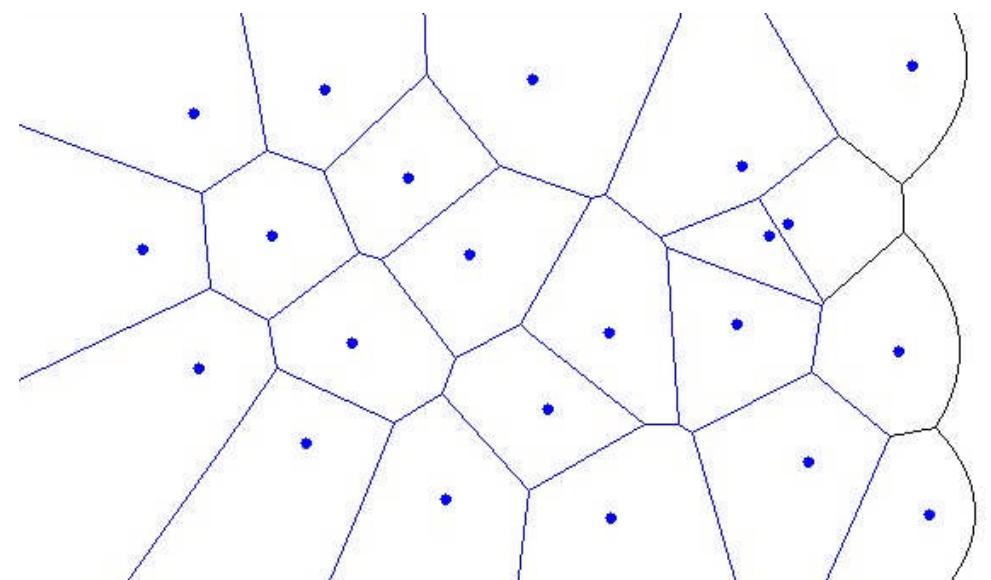
Computing Voronoi diagrams

- For n points, there are
 - n point events
 - at most $2n$ circle events



Computing Voronoi diagrams

- Handling an event takes $O(\log n)$ time due to the balanced binary tree that stores the beach line → in total $O(n \log n)$ time



Divide-and-Conquer

- Divide the points into two parts.

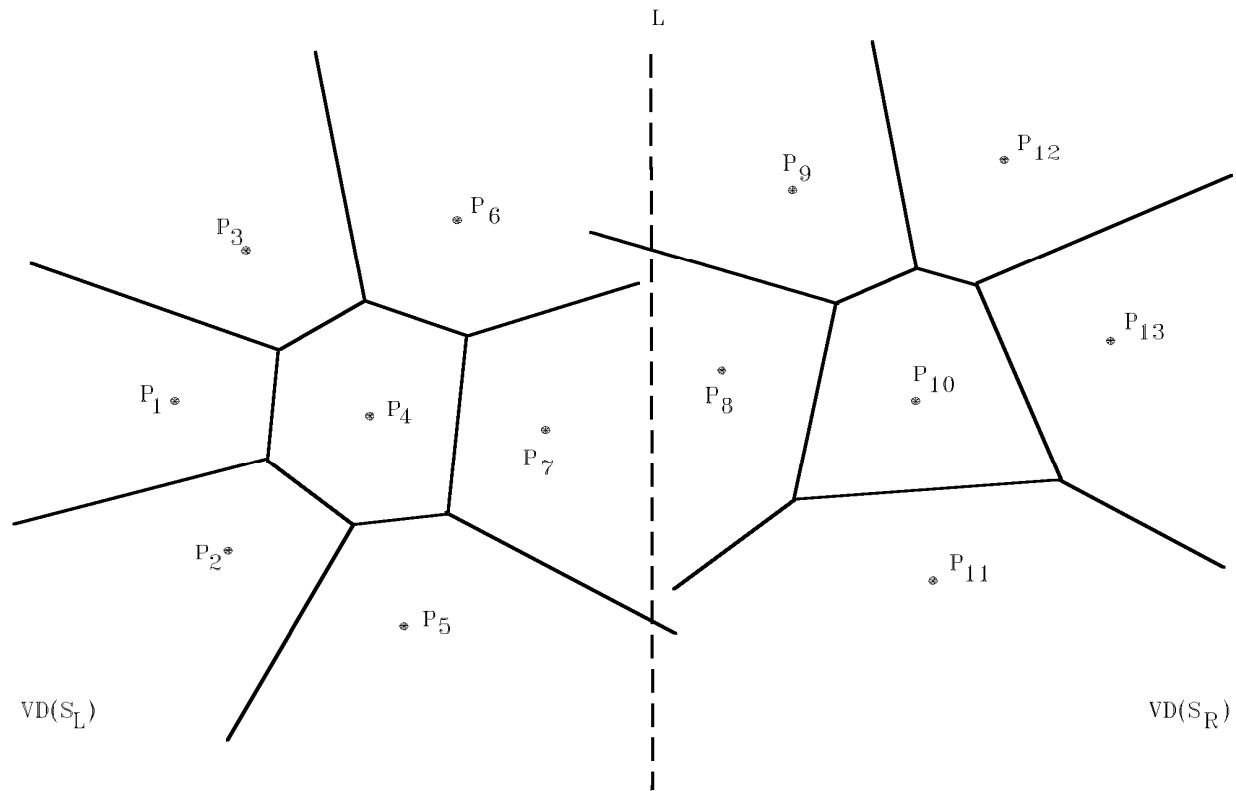


Fig. 5-17: Two Voronoi Diagrams After Step 2

Merging two Voronoi diagrams

- Merging along the piecewise linear hyperplane

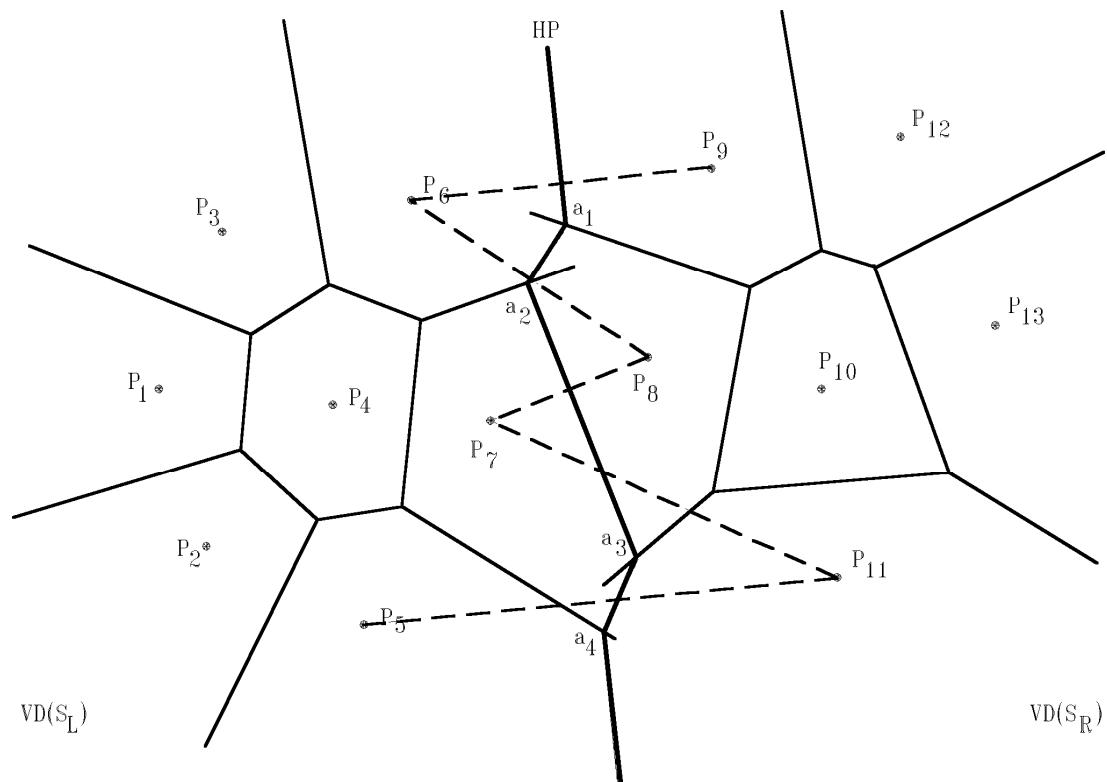


Fig. 5-18: The Piecewise Linear Hyperplane for the set of Points Shown in Fig. 5-17.

The final Voronoi diagram

- After merging

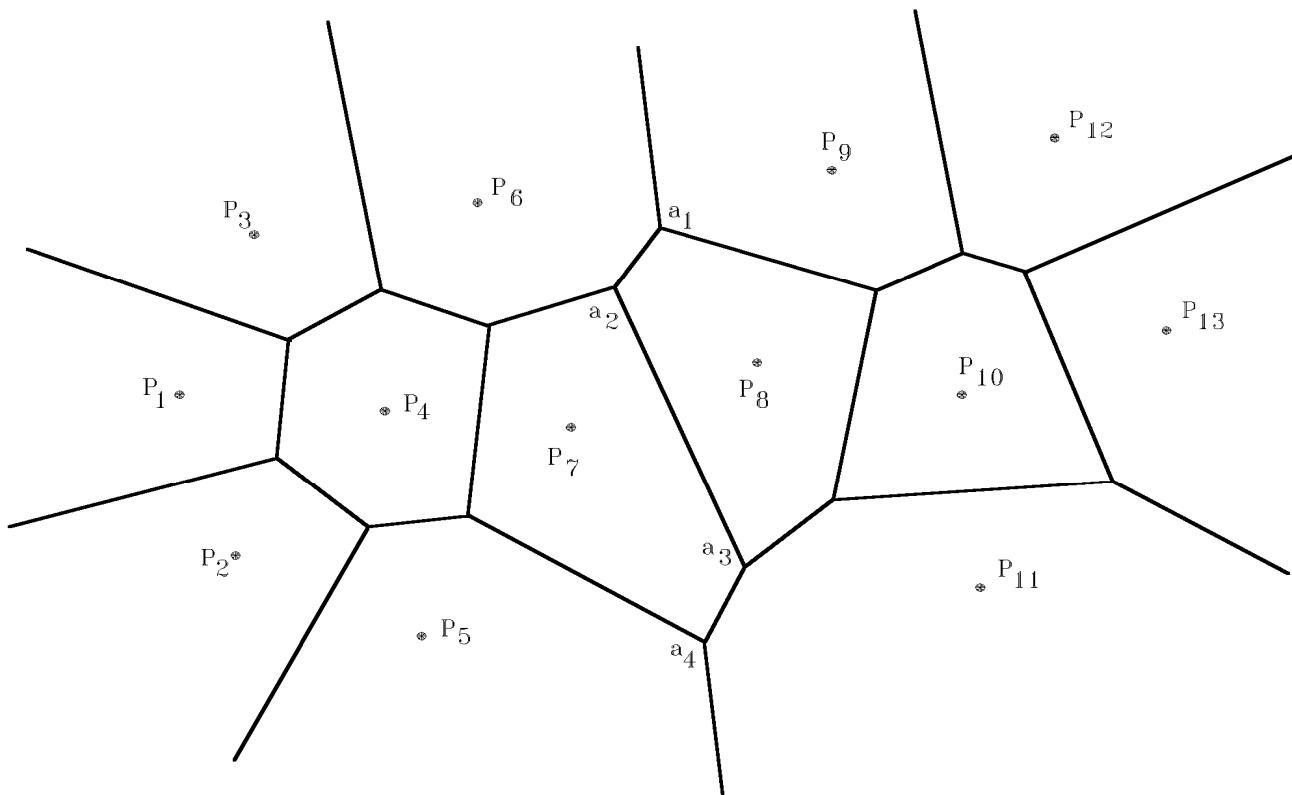


Fig. 5–19: The Voronoi Diagram of the Points in Fig. 5–17.

Time Complexity

- Merging takes $O(n)$ time (**This is the Key!**)
- $T(n) = 2 * T(n/2) + O(n)$
- $T(n) = O(n \log n)$

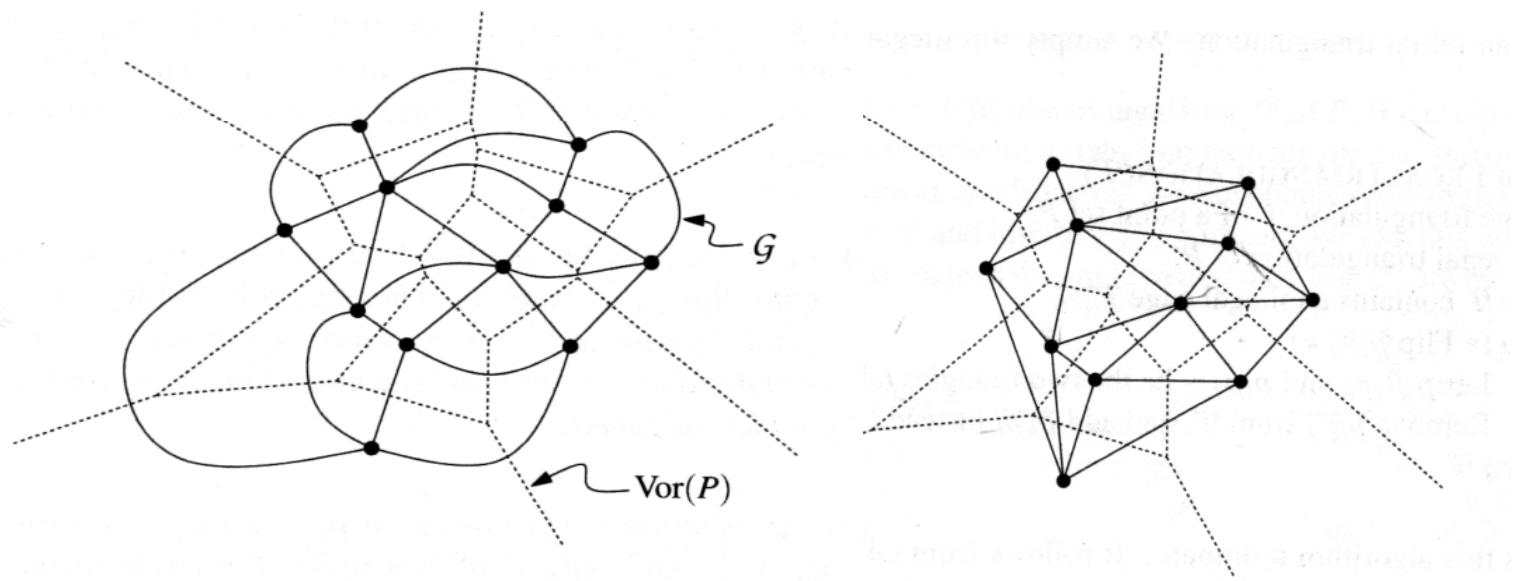
Intermediate summary

- Voronoi diagrams are useful for clustering (among many other things)
- Voronoi diagrams can be computed efficiently in the plane, in $O(n \log n)$ time
- The approach is plane sweep (by Fortune)

*Figures from the on-line animation of
Allan Odgaard & Benny Kjær Nielsen*

Dual graphs of Voronoi diagrams

- Delaunay triangulation is a dual graph to a Voronoi diagram.

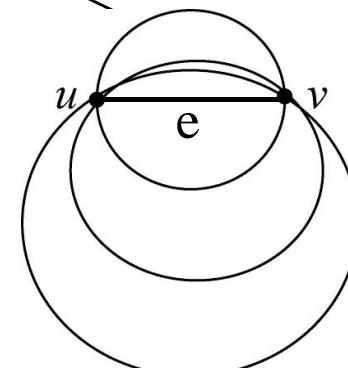
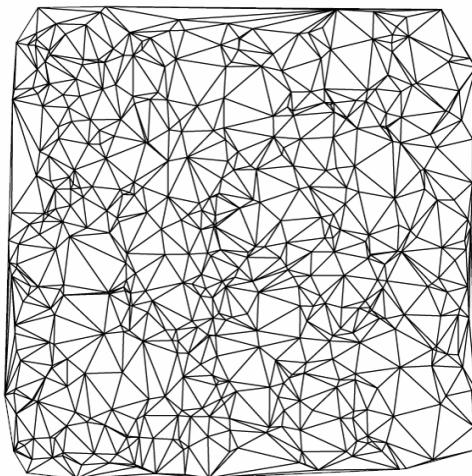


Delaunay triangulation

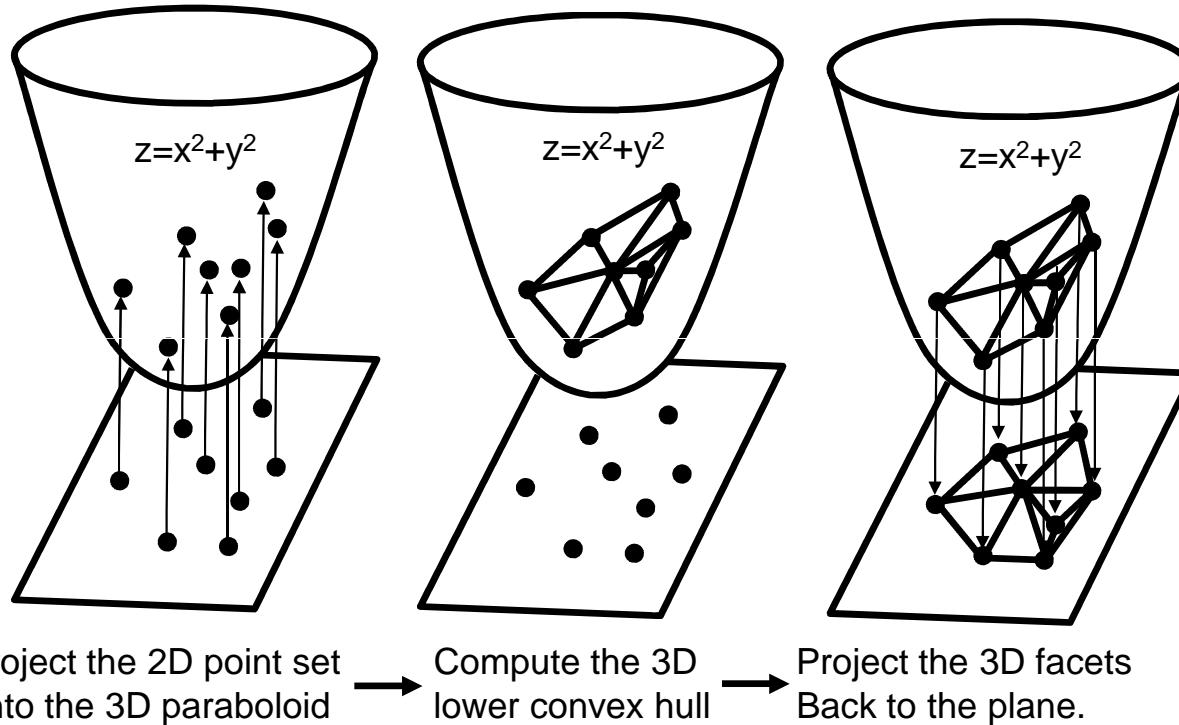
- also called tessellation

Delaunay triangulation

- An *edge* e is called *Delaunay edge* iff there exist an *empty* circle passing through e .
- *Delaunay triangulation* - a set of Delaunay edges



Delaunay triangulations and convex hulls



The 2D triangulation is Delaunay!

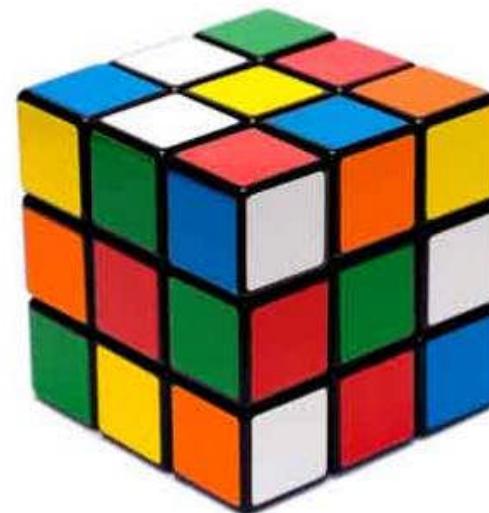
Intractable (Hard) Problems



InAndOut
Gifts.com

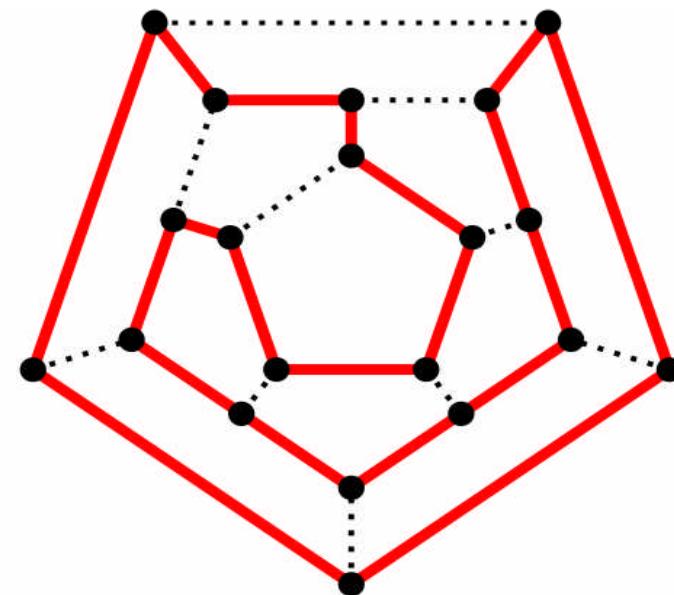
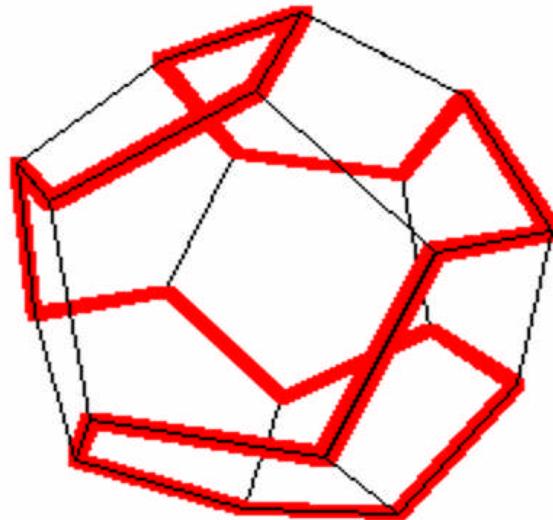
Combinatorial puzzles

- Hard to solve: **exponential** number of choices, too many to try one by one.
- But easy to check if a given solution is correct or not.



Finding a Hamiltonian cycle on a dodecahedron, old puzzle

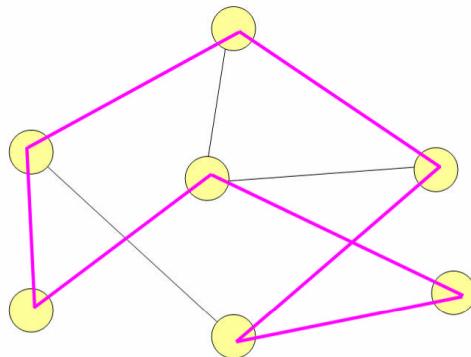
- Left: Dodecahedron with a solution
- Right: dodecahedron flattened into a planar graph



How many choices is too many?

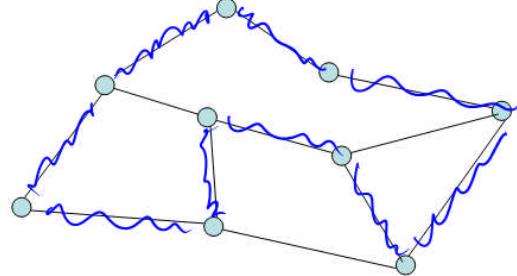
- Number of subsets of n elements is $O(2^n)$.
- Size of truth table of n boolean variables,
is $O(2^n)$
- Choose k out of N is also exponential,
 $O(n^k)$.
$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \quad \text{for } 0 \leq k \leq n.$$
- Number of paths in a graph with n nodes,
is $n!$, $O(n^n)$.
$$n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$$

Hamiltonian Cycle is NPC

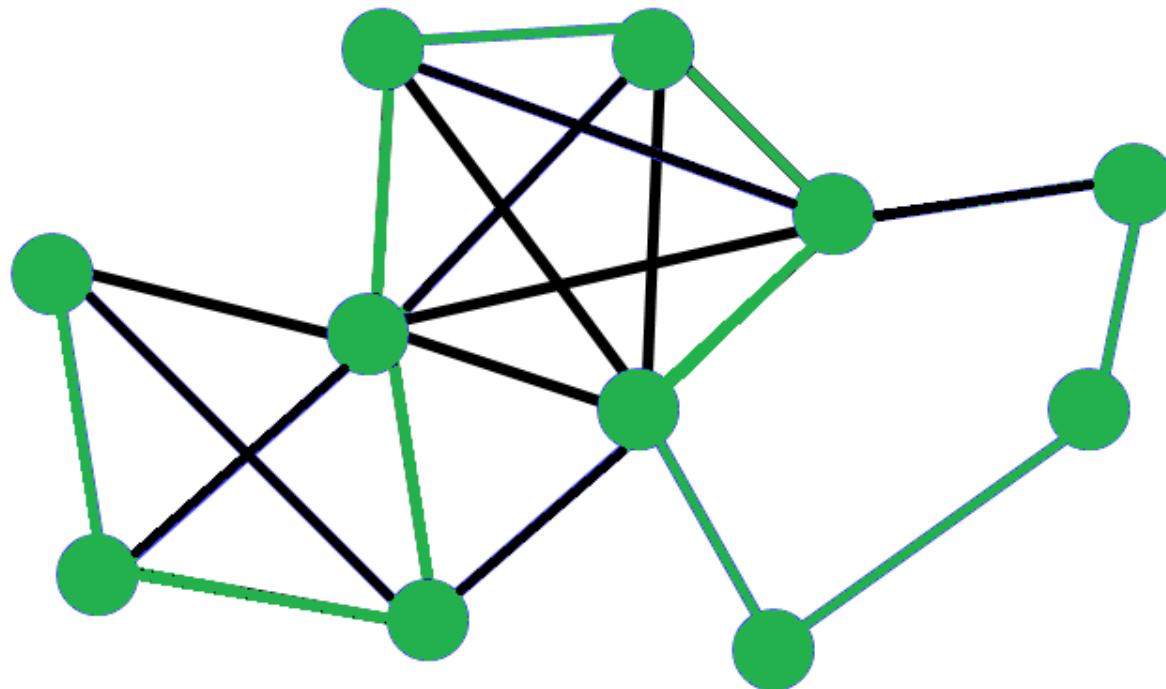


Is there a Hamilton cycle (tour that visits each vertex exactly once)?

Number of cycles to check is $n!$



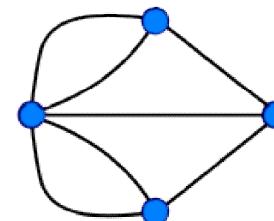
Hamiltonian Path problem is NPC
(no need to return to start).



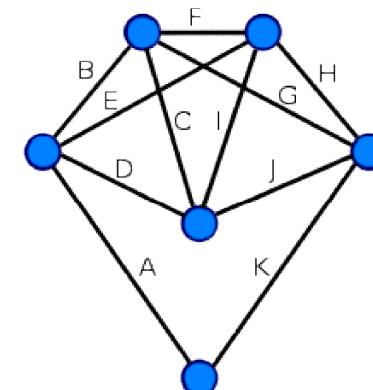
Euler Path is in P

- **Eulerian path** is a trail in a graph which visits every edge exactly once.
- Examples:

G1. This [Königsberg Bridges](#) graph does not have a Euler circuit.



G2. Every vertex of this graph has an even degree, therefore this is an Eulerian graph. Following the edges in alphabetical order gives an Eulerian circuit/cycle

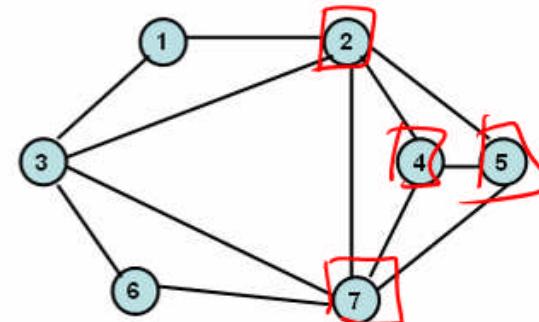
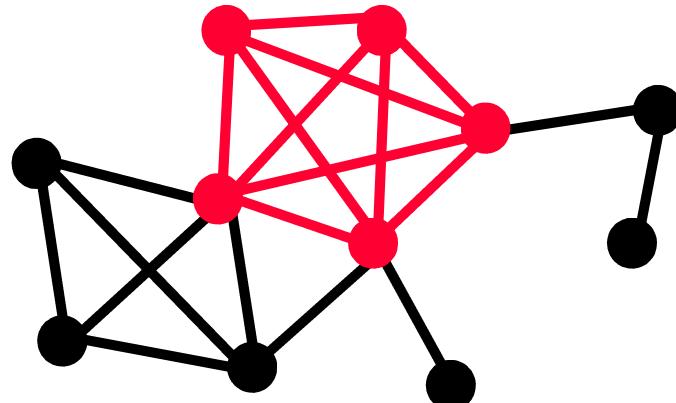


k-Clique problem is NPC

k-Clique problem: Does G have a clique of size k?

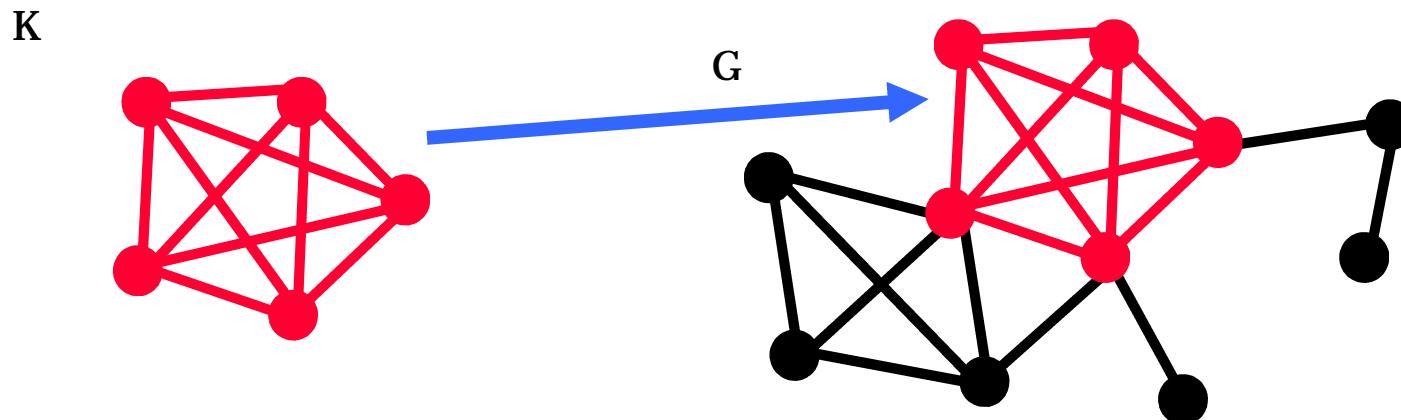
Number of k-cliques to test is $C(n,k)$.

Max-clique: What is the largest number of nodes in G forming a complete (clique) subgraph?



Subgraph isomorphism problem

- K is called a subgraph of G,
 - If K is embedded inside G, i.e. by deleting some vertices and edges of G, we get K.
- Deciding if graph K is a subgraph of G is NPC
 - It is as hard as finding a clique K in G.



Chromatic number of a Graph

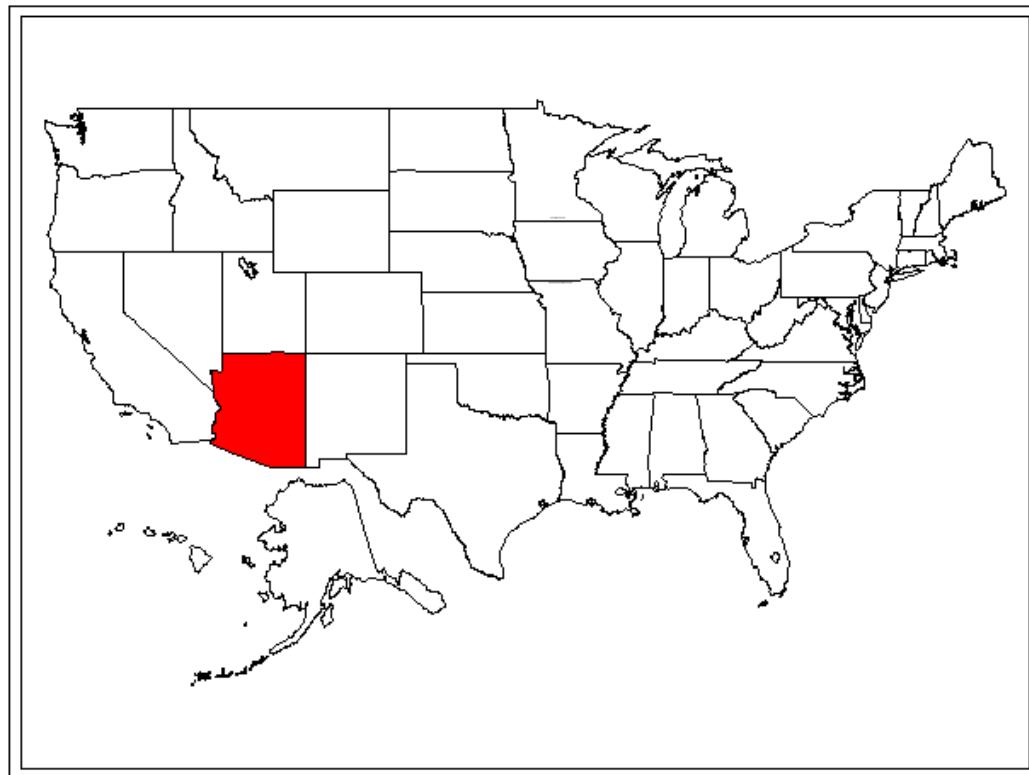
Given a graph G , how many **colors** are required to paint the vertices? such that adjacent vertices have different colors.



A torus requires 7 colors!

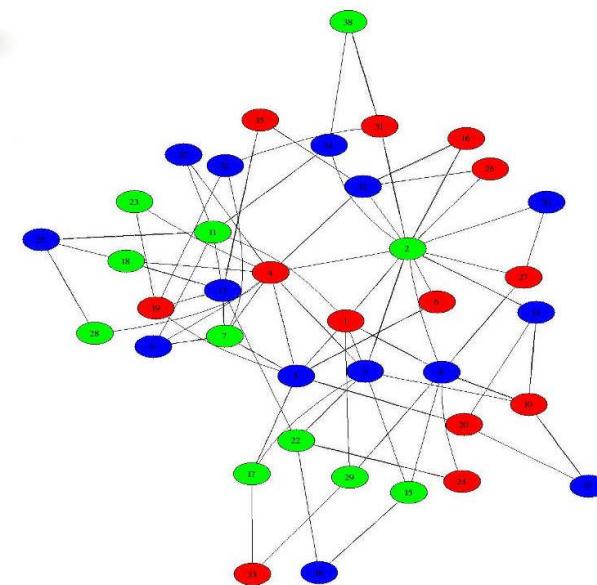
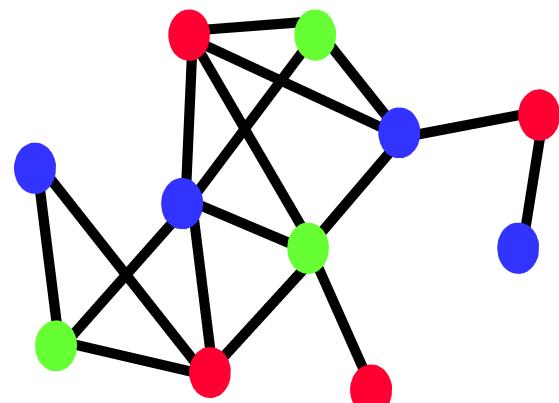
4 colors suffice for planar graphs

[Appel and Haken] A proof of the 4-color theorem. *Discrete Mathematics*, 16, 1976.



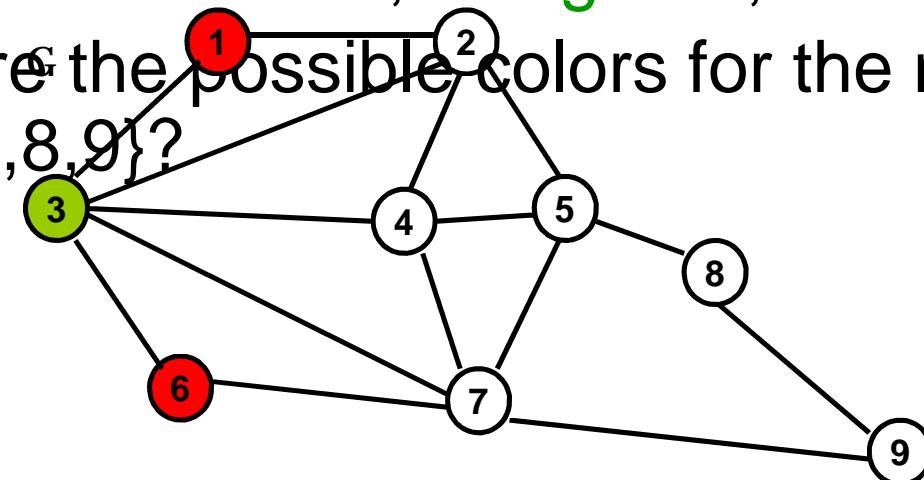
Graph 3-Coloring is NPC

Given a graph G, Can we color the vertices with 3 colors? where adjacent vertices have different colors.



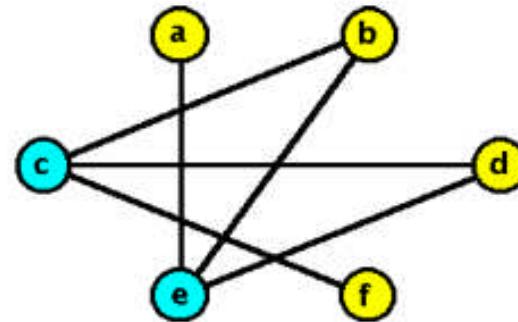
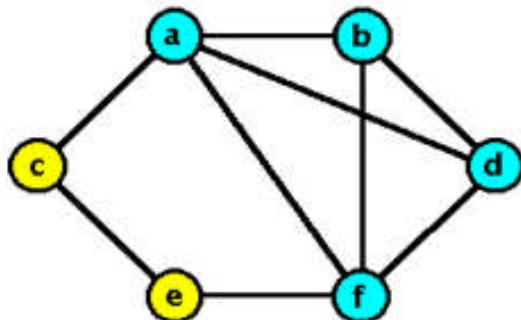
Exercise: 3 coloring a graph

- Is it possible to color G with 3 colors {Green, Red, Blue}?
- Each node must be colored differently from its neighbours.
- Use minimal colors
- In G, vertex 1 is red, 3 is green, 6 is red.
- What are the possible colors for the nodes {2,4,5,7,8,9}?



Vertex Cover (VC) is NPC

- Given a graph G and an integer k , is there a subset of k vertices (called cover) such that every edge is connected to a vertex in the cover?

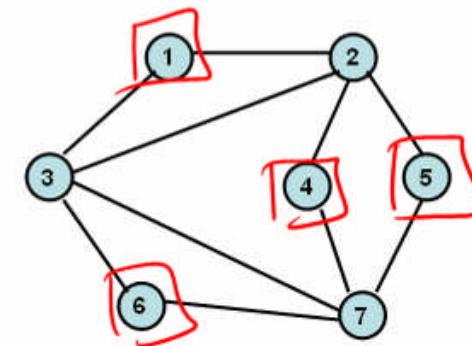
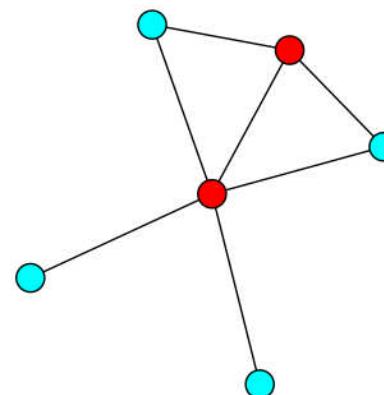
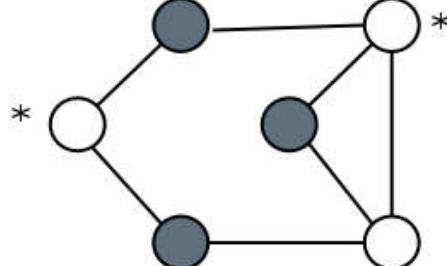


Independent set is NPC

Independent set is a subset of vertices in the graph with no edges between any pair of nodes in the independent set.

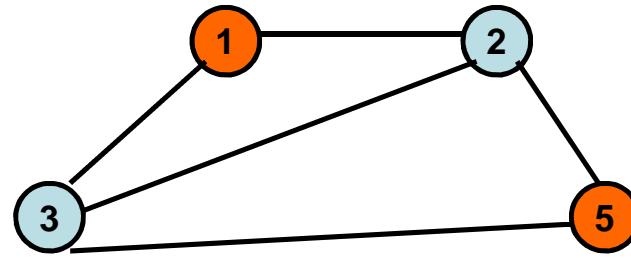
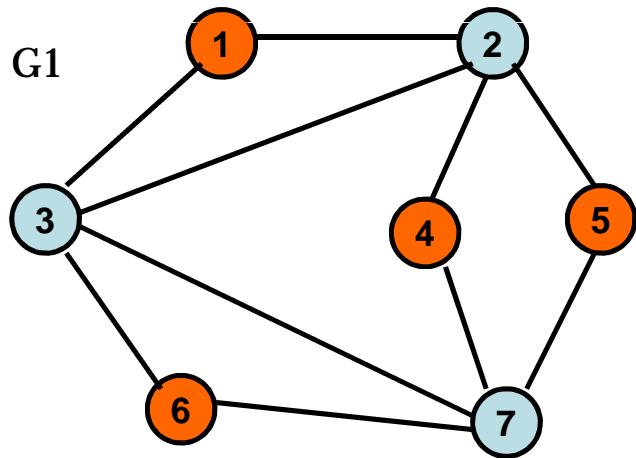
Given a graph $G=(V,E)$, find the *largest independent set*.

Examples:



Exercises

1. Find a maximum independent set S
2. Find a minimal vertex cover
3. Find a hamiltonian path
4. Find an Euler circuit
5. 3 Color the nodes



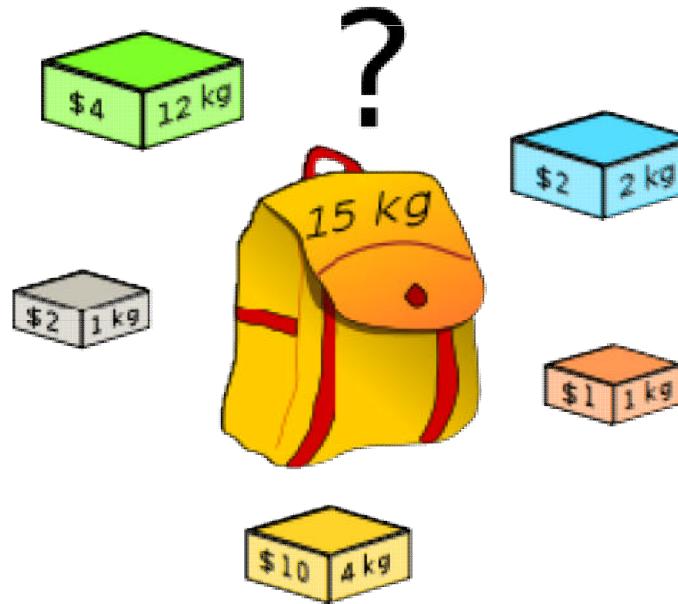
Knapsack problem is NPC

Given a set A integers $a_1..a_n$, and another integer K in binary.

Q. Is there a subset B of A, of these integers that sum exactly to K?

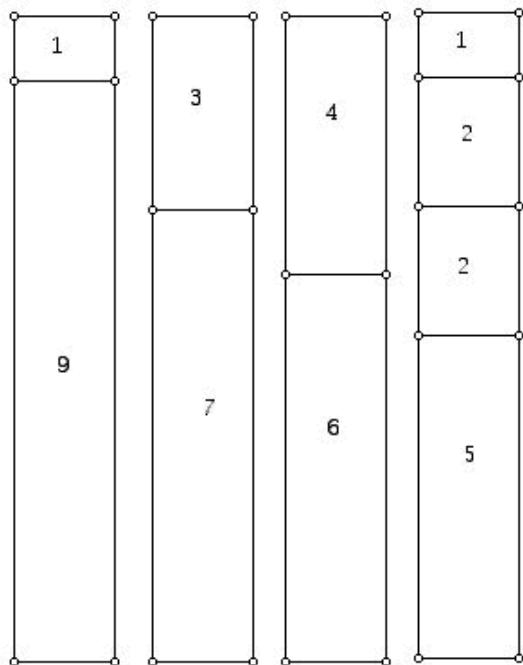


Brute force packing



Bin packing problem is NPC

Can k of bins of capacity C each, store a finite collection of weights $W = \{w_1, \dots, w_n\}$?



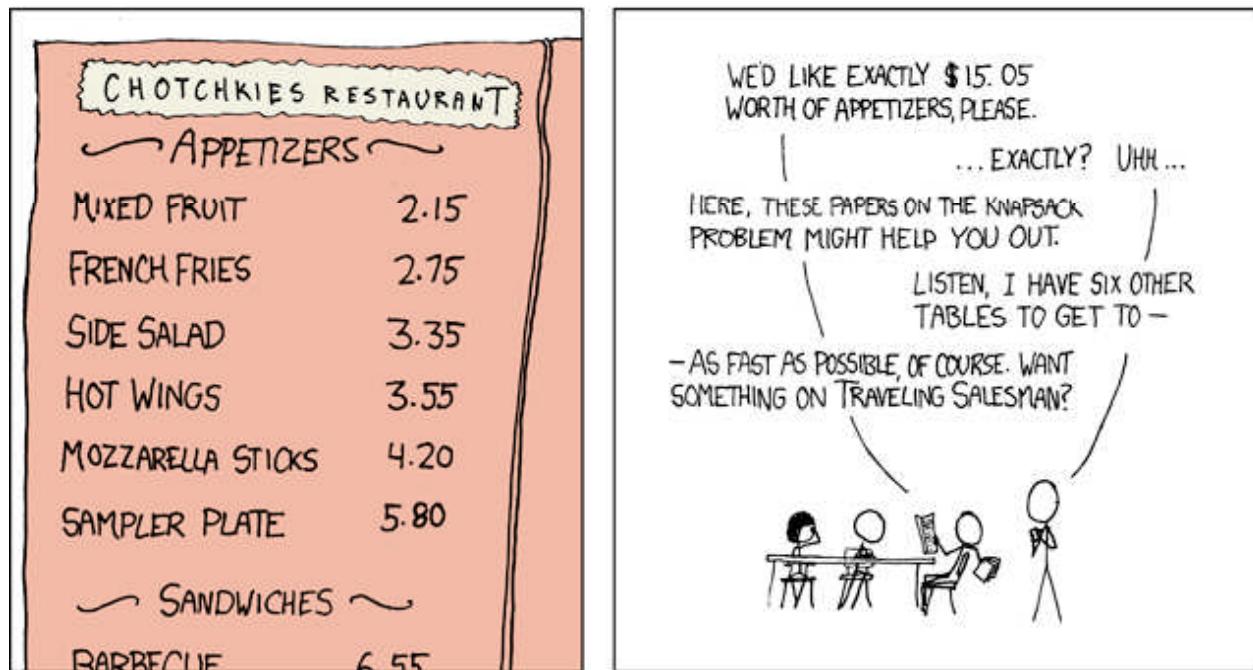
Example:

We have 4 bins with capacity 10 each.
 $k=4$ bins, $C=10$, $W = \{1, 9, 3, 7, 6, 4, 1, 2, 2, 5\}$



Integer subset sum (partition) is NPC

Given a set of numbers, can you divide it into 2 sets of equal sums?



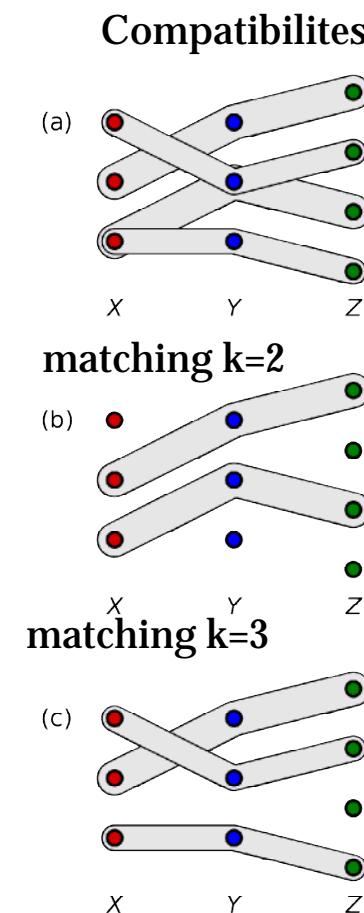
Restaurant ordering is NP Complete
(to divide the bill exactly)

3D matching is NPC

3D-Matching Problem: Is there a matching of size k ?

Given 3 sets $\{X, Y, Z\}$ and their compatibilities.

But 2D matching is easy, using bipartite graph matching.



Language of Boolean formulas

1. **Constants**: True, False (1, 0).
2. Boolean **Variables**: $\{x_1, x_2, x_3, \dots\}$
3. **Literals** are the **variables** $\{x_1, x_2, x_3, \dots\}$ or its negation $\{\neg x_1, \neg x_2, \neg x_3, \dots\}$.
4. **Negation** (NOT, !, \sim , \neg , operator).
5. **Conjunction** is (AND, \wedge , $\&$, operator) of literals are called clause.
6. **Disjunctions** is (OR, \vee , $|$, operator)

Language of Boolean formulas

1. OR(AND(literals)) is called **DNF formula**
(disjunctive normal form).
2. AND(OR(literals)) is called **CNF formula**
(conjunctive normal form).
3. CNF formula, with 3 literals in each clause is called **3CNF**.

e.g. of 3CNF $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_4)$.

e.g. of 2CNF $(x_3 \vee x_6) \wedge (x_4 \vee x_6) \wedge (x_5 \vee x_6)$.

Language of Boolean Logic

1. A formula is **satisfiable**, if it is true for some truth assignment to its variables. e.g. $S = "(x \text{ or } y \text{ or not}(z))"$.
2. A formula is **valid (tautology)**, if it is true for all (any) truth assignment to its variables. e.g. $V = "(x \text{ or not}(x))"$
3. A formula is **unsatisfiable (invalid)**, if it is false for all values of its variables, e.g. $U = "(x \text{ and not}(x))"$

Boolean Algebra

- Distributive Laws:

$$x \text{ or } (y \& z) = (x \text{ or } y) \& (x \& z)$$

$$x \& (y \text{ or } z) = (x \& y) \text{ or } (x \& z)$$

- D'Morgan's laws:

$$\!(x \& y) = \!x \text{ or } \!y$$

$$\!(x \text{ or } y) = \!x \& \!y$$

- Any boolean formula can be converted into standard form called CNF:

(...or...)&(..or..)&(..or..)...

E.g. $P = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4)$

Boolean formulas

Consider the Boolean expression:

$$\begin{aligned} P = & (x_1 \vee \neg x_2 \vee x_3) \quad \dots \text{clause 1} \\ & \wedge (x_2 \vee \neg x_3 \vee x_4) \quad \dots \text{clause 2.} \end{aligned}$$

1. P is a **conjunction** (AND) of 2 **clauses**.
2. Each **clause** is a **disjunctions** (OR) of 3 literals (called **3CNF**)
3. The **literals** are the **variables** $\{x_1, x_2, x_3\}$ or its negation (**not**) $\{\neg x_1, \neg x_2, \neg x_3\}$.
4. P is true, if every clause is true, easy enough?

Satisfiability (SAT)

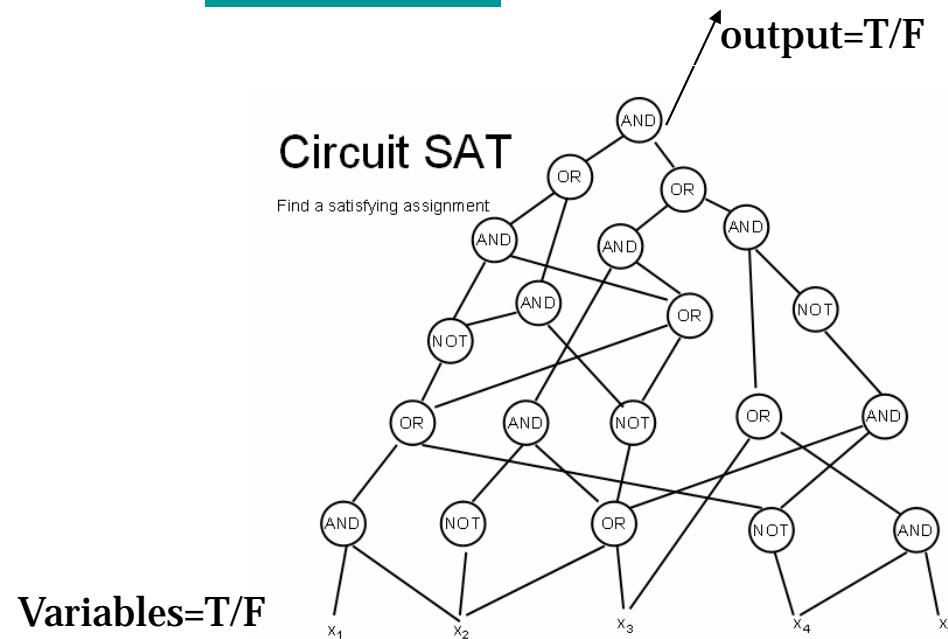
- Is this boolean expression satisfiable?

$$Q = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee x_4 \vee x_5)$$

- Can you assign some truth values to all the x_i 's such that $Q = \text{true}$?
- Easy solution: set x_1 , x_2 , and x_4 to True
- More solutions: $\{x_1, x_2, \textcolor{red}{x}_5\} = \{\text{True}\}, \dots$
- But how to solve a formula with 5000 variables and 300 clauses? Truth table will have $O(2^{5000}) = O(10^{1500})$ rows to check.

Satisfiability problem (SAT) is NPC

Given a boolean formula, determine if there exists an assignment to the variables, that satisfies the formula?

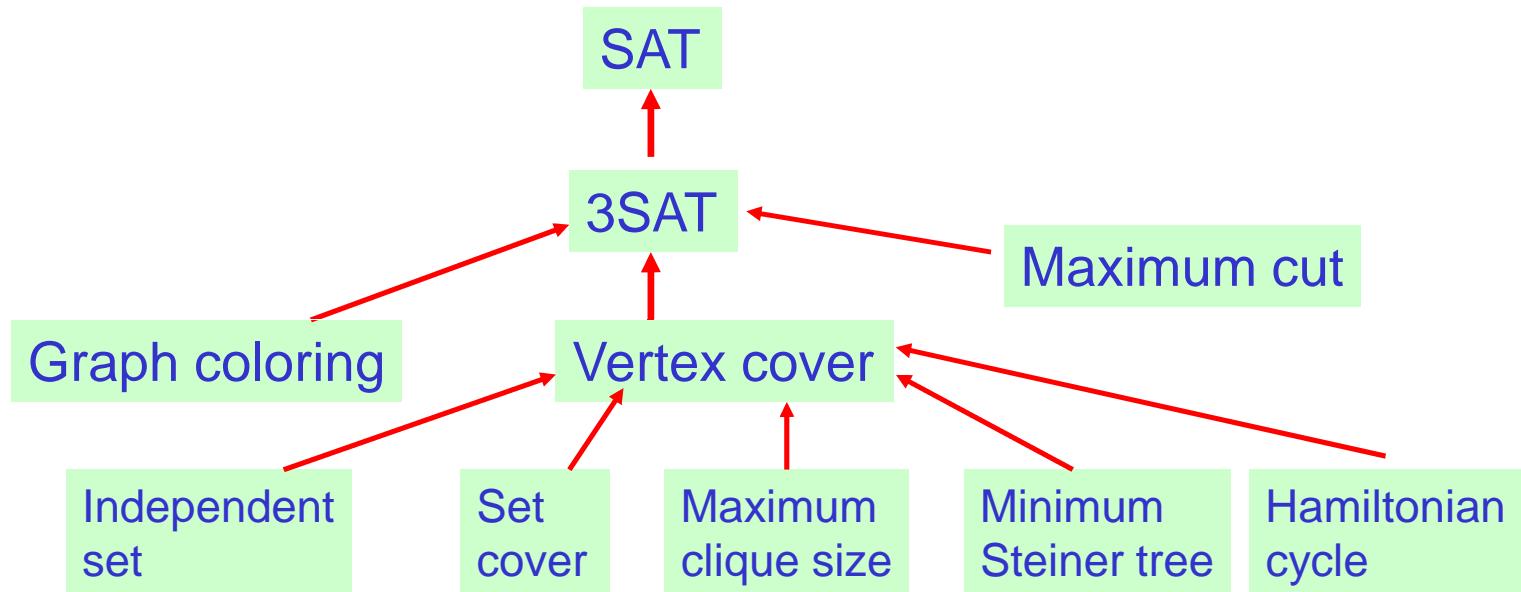


3SAT is also NPC

- 3CNF is a boolean formula with just 3 literals per clause.
- 3SAT problem: is a given 3CNF formula satisfiable? NP Complete.
- But 2-SAT is in P (polynomial algorithm).

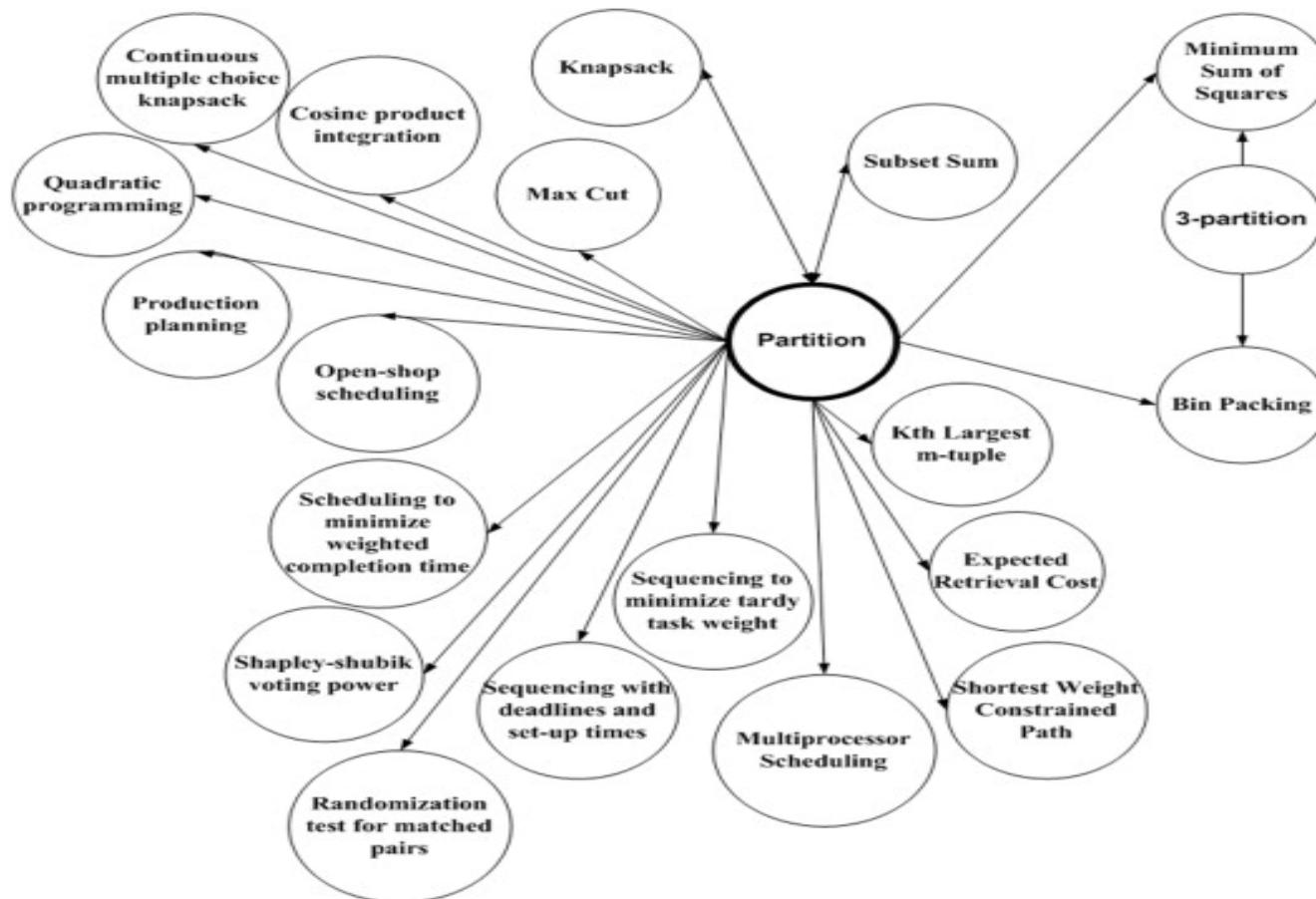
Examples of NP-complete problems

some NPC problems



find more NP-complete problems in
http://en.wikipedia.org/wiki/List_of_NP-complete_problems

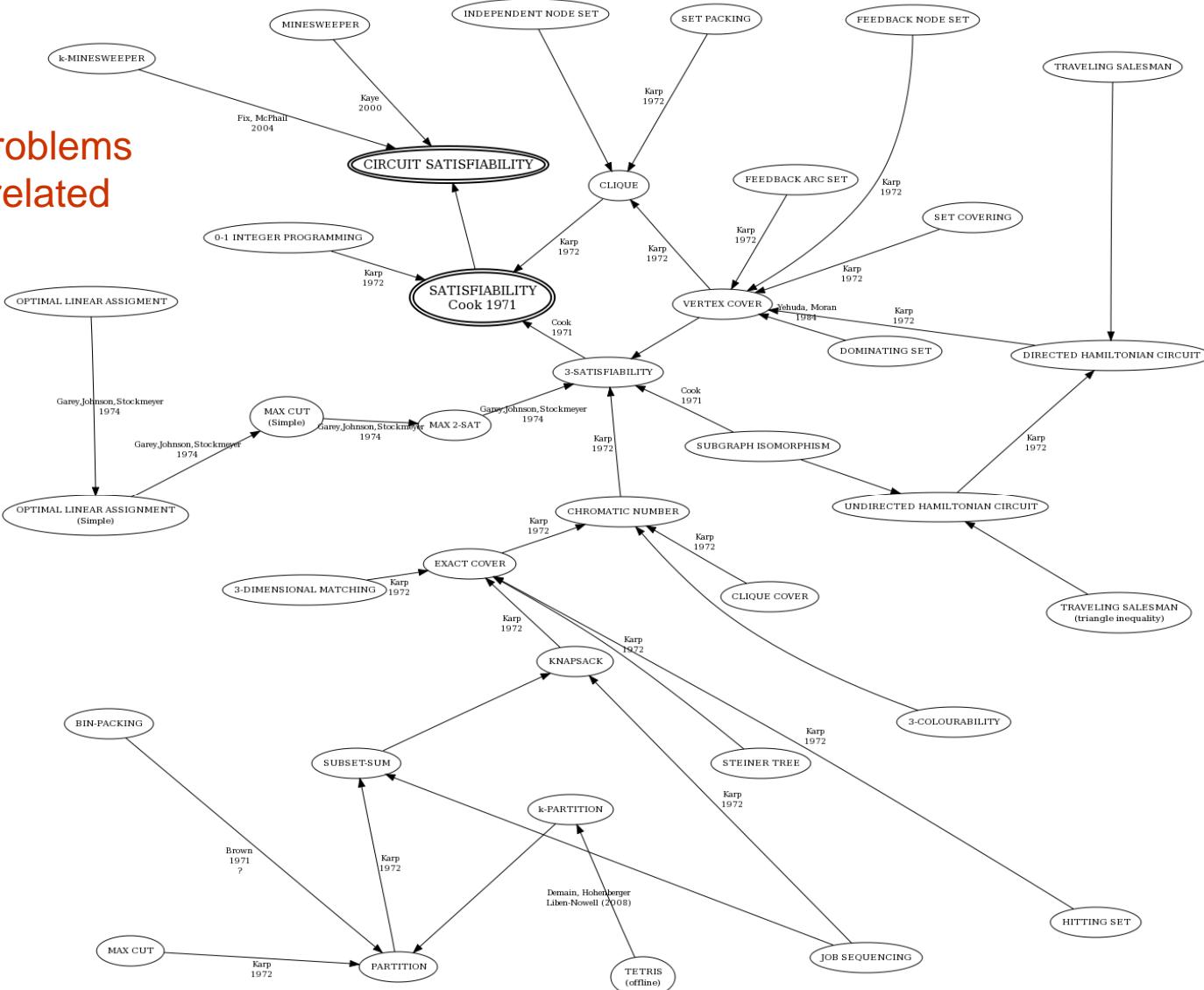
Some NPC Problems



NP Complete list?

- There are hundreds of NP complete problems
- It has been shown that if you can solve one of them efficiently, then you can solve them all efficiently (polynomial time).

NPC Problems are all related



TSP: The traveling saleperson problem

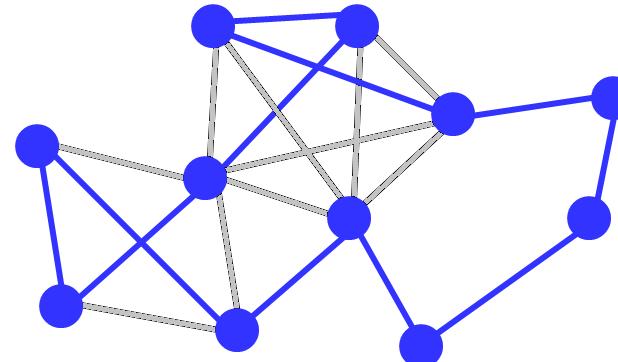
Given a graph with N cities and edges with costs (the costs of traveling from any city to any other city)

What is the cheapest round-trip route that visits each city once and then returns to the starting city?

This is an optimization problem (find the minimum costs: not a decision yes/no problem)

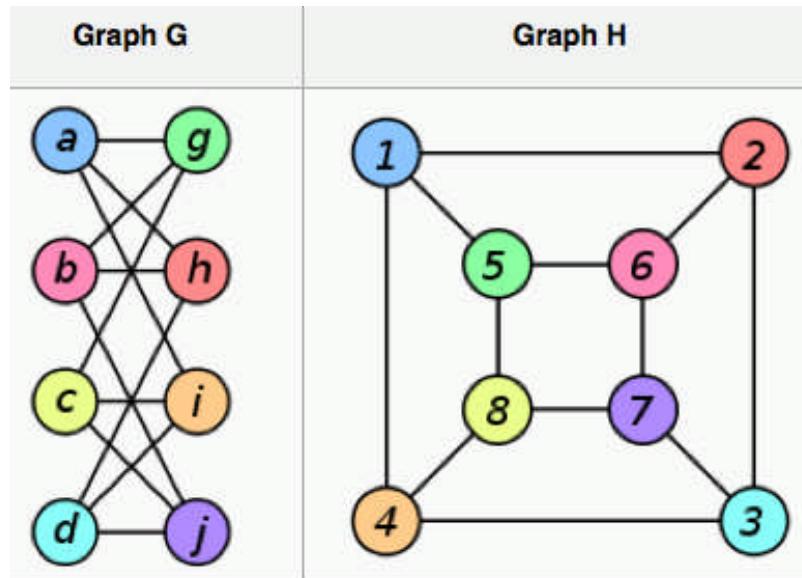
Traveling Salesman vs Hamiltonian Path

1. The Decision problem: Is there a **Hamiltonian circuit** of a given cost **NP-Complete**.
2. The Optimization problem: **Traveling Salesman Problem** is to find the cheapest circuit is **NP Hard, Not NP-Complete**.



Graph isomorphism problem is ??

- Two graphs G and H are **isomorphic**, if they have the same structure (connections) but their vertices may have different names.
- Example:



Unresolved complexity: deciding if two graphs isomorphic?

Graph G	Graph H	An isomorphism between G and H
		$\begin{aligned}f(a) &= 1 \\f(b) &= 6 \\f(c) &= 8 \\f(d) &= 3 \\f(g) &= 5 \\f(h) &= 2 \\f(i) &= 4 \\f(j) &= 7\end{aligned}$

Example

Unresolved complexity

Integer Factoring.

- Not all algorithms that are $O(2^n)$ are NP complete.
- Integer factorization (also $O(2^n)$) is not thought to be NP complete.
- Note: Size of the problem is measured by the number of digits in n (not the value of n). E.g. n=999, size of n is 3.

"Is Primes" is in P

- “Primes is in P” [Manindra Agrawal, Kayal, Saxena, IIT Kanpur, 2002]
- They give an unconditional deterministic polynomial-time algorithm
- The algorithm decides whether an input number is prime or composite (but not its factors).



P = NP? NP Completeness

General Definitions

P, NP, NP-hard, NP-easy, and NP-complete

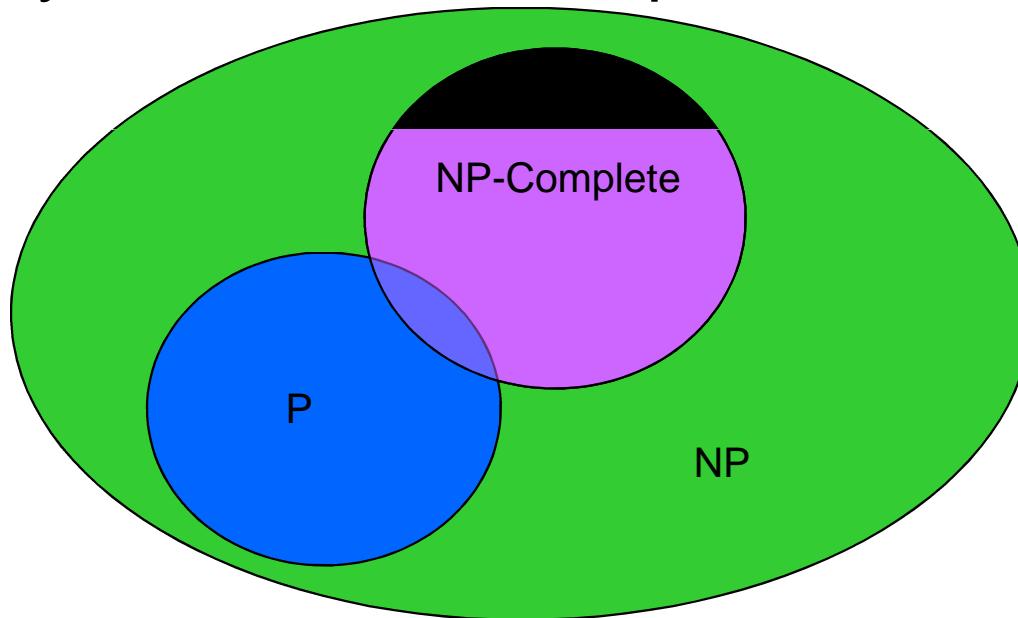
- Problems
 - Decision problems (yes/no)
 - Optimization problems (solution with best score)
- P
 - Decision problems (decision problems) that can be solved in polynomial time (efficiently).
- NP
 - Decision problems whose “YES” answer can be verified in polynomial time, if we already have the *proof* (or *witness*)
- co-NP
 - Decision problems whose “NO” answer can be verified in polynomial time, if we already have the *proof* (or *witness*)

P = NP?

- P: The complexity class of decision problems that can be solved on a deterministic Turing machine in polynomial time.
- NP: The complexity class of decision problems that can be solved on a non-deterministic Turing machine in polynomial time.
- P=NP is the biggest **unsolved problem** in CS
- Text book: **Garey and Johnson**, Computers and Intractability: A Guide to the Theory of NP-Completeness, 1979.

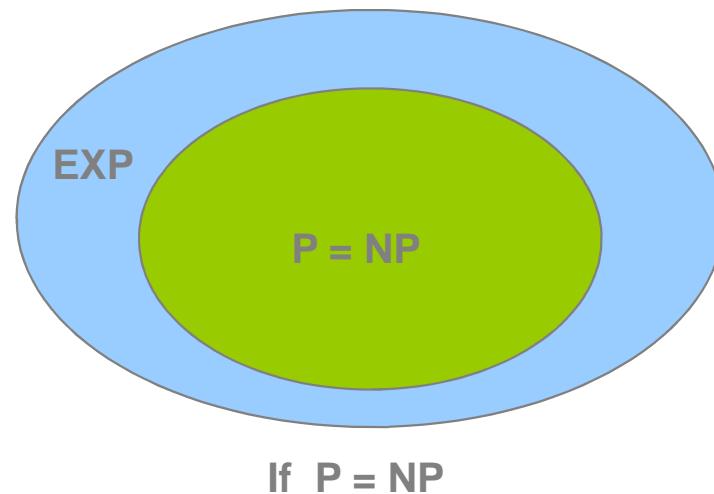
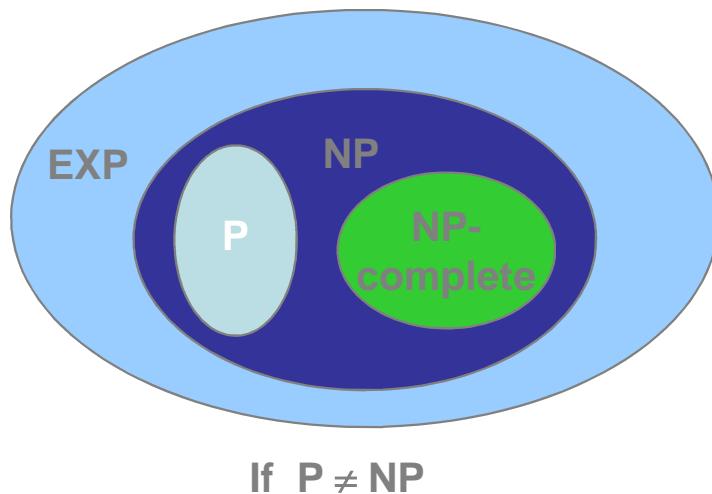
NP-Completeness

- How would you define NP-Complete?
- They are the “hardest” problems in NP



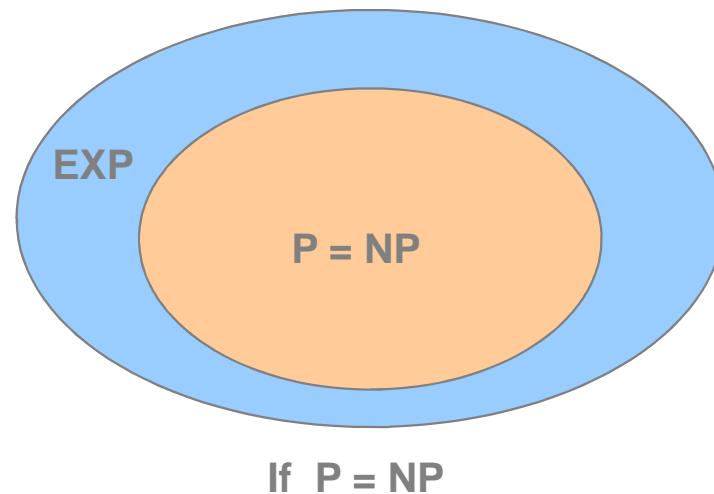
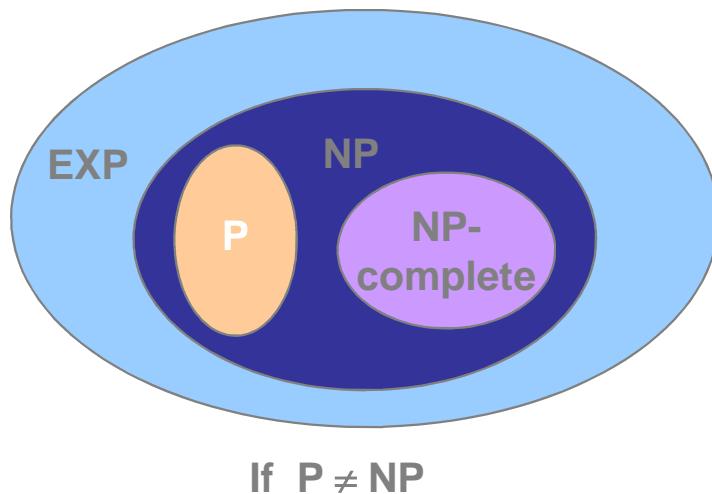
Definition: S is called **NP-Complete** if

1. Problem S in NP and
2. S is NP hard: Every other NP problem is polynomial time **reducible** to S



Definition: S is in P

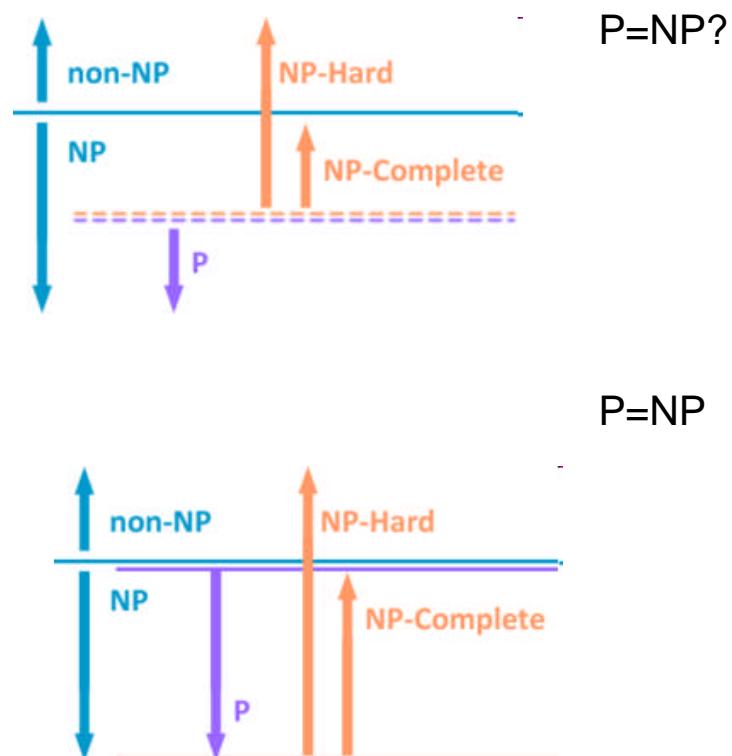
- S is in P if S can be solved in polynomial time by a deterministic turing machine.



An other P versus NP diagram

1. **P:** The class of problems that can be solved by a deterministic Turing Machine in polynomial time.
2. **NP:** The class of problems that can be solved by a non-deterministic Turing Machine in polynomial time.
3. **NP-complete:** The class of problems that are in NP and can be reducible by any problem in NP.
4. **NP-hard:** The class of problems that can be reducible by any problem in NP

from <http://henry2005.jimdo.com/2011/05/21/p-np-np-hard-and-np-complete/>



If P=NP then

- If P=NP, all NP-hard problems can also be solved in polynomial time, e.g. Satisfiability, Clique, Graph coloring, Partition numbers
- Assume, all the basic arithmetic operations (addition, subtraction, multiplication, division, and comparison) can be done in polynomial time.

Satisfiability problem (SAT) is NPC

Given a boolean formula, determine if there exists an assignment to the variables, that satisfies the formula?

- NP: Checking a given solution is easy, If given a solution to SAT, just plug in the values and evaluate to True or False, this can be done quickly, even by hand.
- NP Complete: Finding a solution is hard. Brute force algorithm: try all possible T/F values for the n Boolean variables, $O(2^n)$, exponential time.

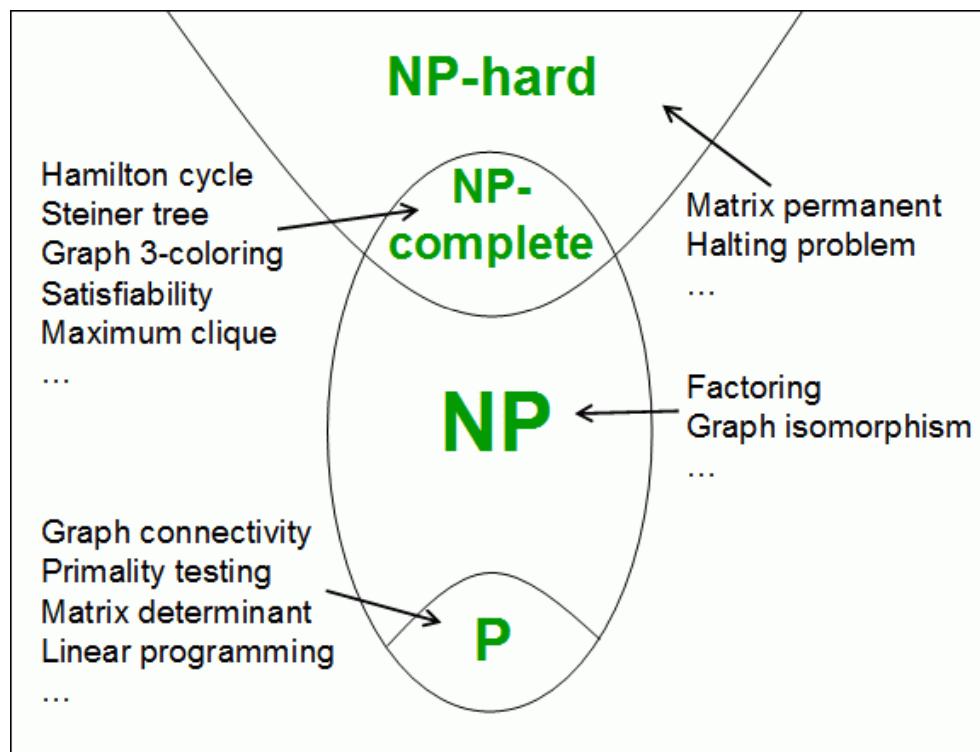
SAT is NPC

- No known efficient way to find a solution.
- There still may be an efficient way to solve it, though! We don't know.
- Cook's theorem (1971) states that SAT is NP-complete. Cook showed that a turing machine computation can be written as a boolean formula.

P = NP?

- P=NP is an Unsolved problem.
- If you could solve any NP complete problem in polynomial time, you would be showing that P = NP

NP Complete problem set



Hardness is hard to prove: nobody has found a fast algorithm for any NPC problem.



"I CAN'T SOLVE IT - BUT NEITHER CAN ALL THESE FAMOUS PEOPLE ! "

But what if P=NP, and the SAT algorithm is $O(n^{10000000})$?

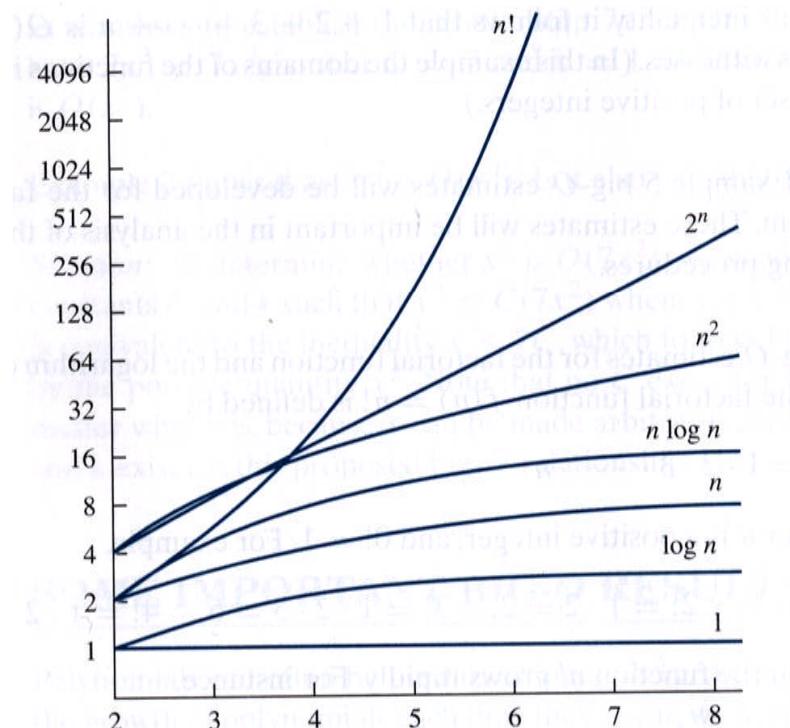


FIGURE 3 A Display of the Growth of Functions Commonly Used in Big-O Estimates.

Coping with NP-Hardness

- Approximation Algorithms
 - Run quickly (polynomial-time).
 - Obtain solutions that are guaranteed to be close to optimal

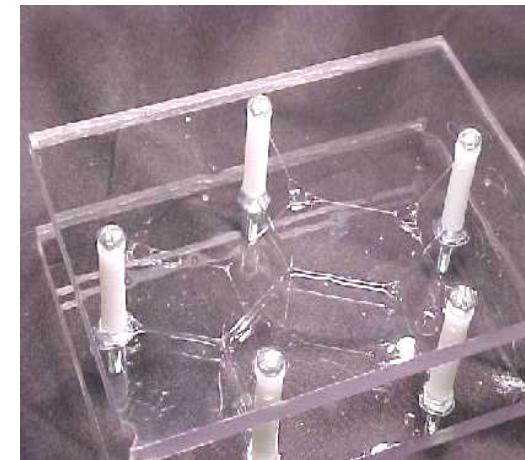
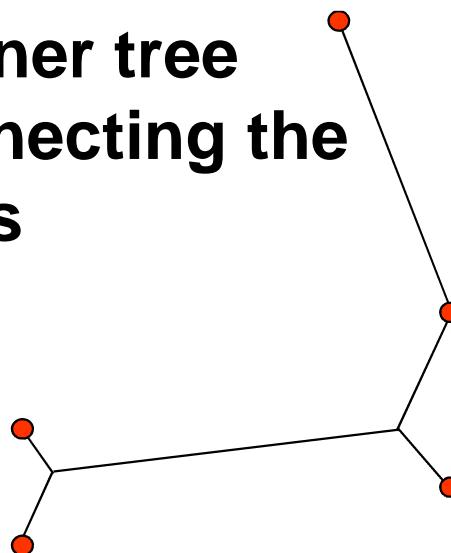
Parallel Computers

- Maybe there are **physical** system that solves an NP-complete problem just by reaching its lowest energy state?
- **DNA computers:** Just highly parallel ordinary computers
- **Quantum computers?**

Physical: soap solver

Dip two glass plates with pegs between them into soapy water

Let the soap bubbles form a minimum Steiner tree connecting the pegs



NP, NP-Hard and NPC Reductions

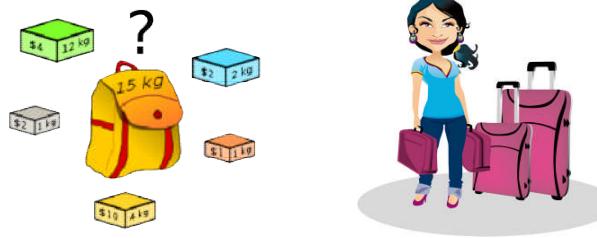
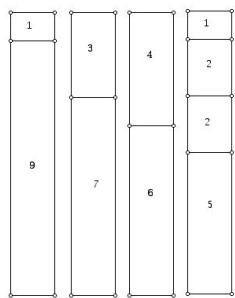
- **NP**: to show B is in NP, it has non-deterministic polynomial-time guess-and-verify algorithm.
- **NP Hard**: A decision problem B is said to be NP-hard if every decision problem in NP reduces to B; that is, for every decision problem S in NP, $S \leq_p B$. Thus B is as hard as any problem in NP.
- **NP Complete**: A decision problem B is called NP-complete if it is NP-hard and in NP itself.
- **Reductions** is used to show that a problem B is at least as hard as another problem S. If S is NPC, then B is also NPC.

Reduction $S \leq_p B$

- S is our known hard problem, and we want to show that B is at least as hard, by showing that S reduces to B in polynomial time.

Sample Reductions

- 1) PARTITION = { n_1, n_2, \dots, n_k | we can split the integers into two sets which sum to half the total }.
 - 2) SUBSET-SUM = { $\langle n_1, n_2, \dots, n_k, m \rangle$ | there exists a subset which sums to m }
- 1) If I can solve SUBSET-SUM, how can I use that to solve an instance of PARTITION?
 - 2) If I can solve PARTITION, how can I use that to solve an instance of SUBSET-SUM?



Reduce: Partition \leq_p Subset-Sum

Given $T = \{n_1, n_2, \dots, n_k\}$ to partition into two

Let $S = n_1 + n_2 + \dots + n_k$

We want two partitions of sum= $S/2$ each.

call SUBSET-SUM($\{n_1, n_2, \dots, n_k\}$, $S/2$) will
give us one set equal to $S/2$, and
remaining also will sum to $S/2$.

Reduce: Subset-Sum \leq_p Partition

Given $T = \{n_1, n_2, \dots, n_k\}$ with sum $S = n_1 + n_2 + \dots + n_k$

To find a subsets of T with sum m (and $S-m$ remaining).

Let $w=S-2m$, add w to T , and

Call $\text{Partition}(\{n_1, n_2, \dots, n_k, w\})$

To get two equal partitions: $\{m + w, S-m\}$

Remove w from one partition to get m .

Example.

To solve subset-sum: $S=100$, $m=30$,

Let $S-m=70$, $w=40$,

call $\text{Partition}(100+40)$ to get $40+30$, and 70 .

Polynomial Reductions

- 1) Partition REDUCES to Subset-Sum
 - Partition $<_p$ Subset-Sum
 - 2) Subset-Sum REDUCES to Partition
 - Subset-Sum $<_p$ Partition
- Therefore they are equivalently hard

NP-Completeness Proof Method

To show that Q is NP-Complete:

- 1) Show that Q is in NP
- 2) Pick an instance, S of your favorite NP-Complete problem (ex: Φ in 3-SAT)
- 3) Show a polynomial algorithm to transform S into an instance of Q

Starting with SAT

- Cook and Levin independently showed that SAT is **NP-complete**.
- Since the reducibility relation \leq_p is transitive, one can show that another problem B is **NP-hard** by reducing SAT, or any of the other known **NP-complete** problems, to B.

Showing NP

- To show that B is in **NP**, you should give a nondeterministic polynomial-time guess-and-verify algorithm for it.
- Give a (small) solution, and
- Give a deterministic polynomial-time algorithm to check the solution.

Showing NPC

To show that B is **NP-complete**,

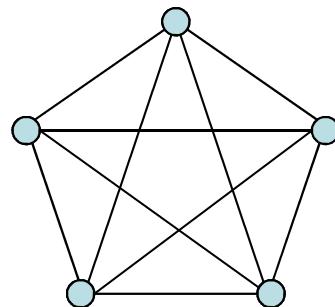
1. Show that B is in **NP-hard** by giving a polynomial-time **reduction** from one of the known **NP-complete** problems (SAT, CNFSAT, CLIQUE, etc.) to B.
2. Show that B is in **NP** by giving a nondeterministic polynomial-time guess-and-verify algorithm for it.

Showing NP Hardness:

- If you are only asked to show **NP-hardness**, you only need to do 1. But if you are asked to show **NP-completeness**, you need to do both 1 and 2

Example: Clique is NPC

- CLIQUE = { $\langle G, k \rangle$ | G is a graph with a clique of size k }
- A clique is a subset of vertices that are all connected
- Why is CLIQUE in NP?

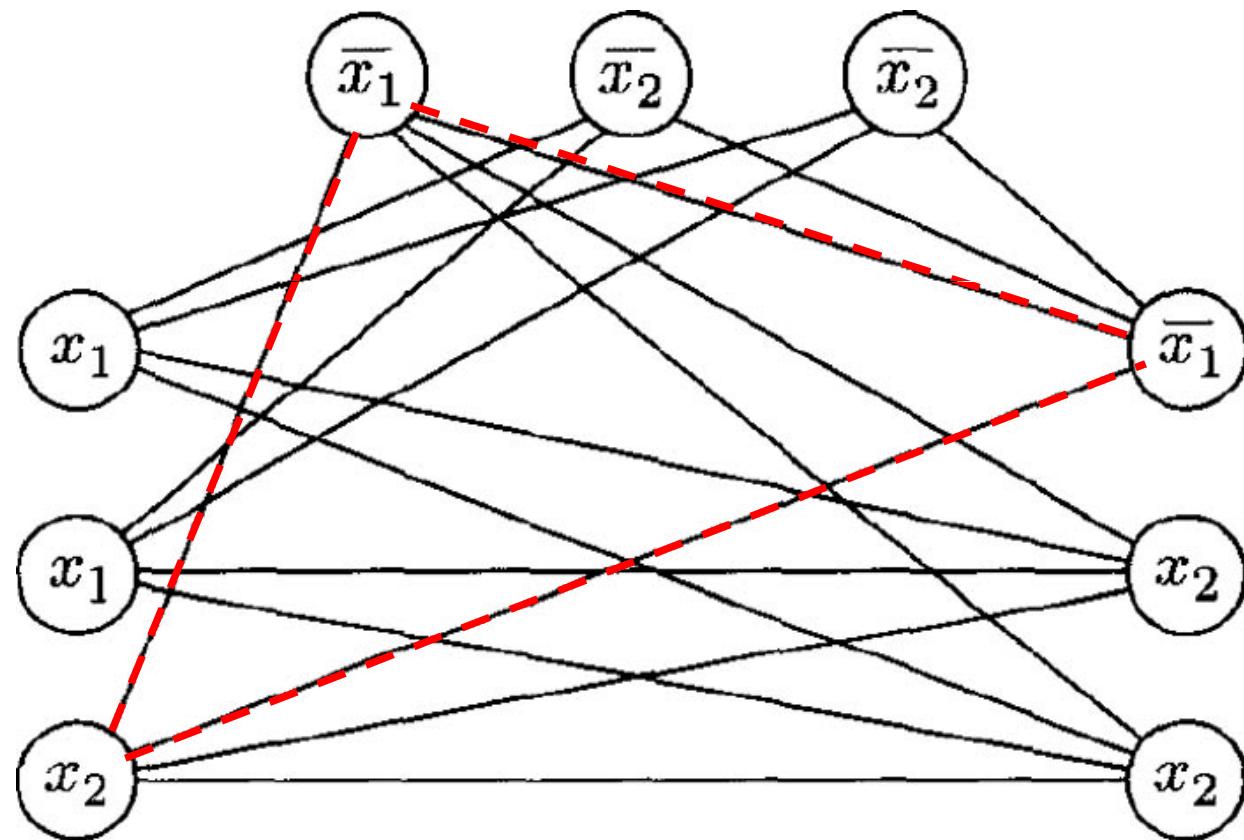


Reduce 3SAT to Clique

- Given a 3SAT problem Φ , with k clauses
- Make a vertex for each literal.
- Connect each vertex to the literals in other clauses that are not its negations.
- Any k -clique in this graph corresponds to a satisfying assignment (see example in next slide)

3SAT as a clique problem

$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\underline{\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_2}) \wedge (\underline{\overline{x}_1 \vee x_2 \vee x_2})$$

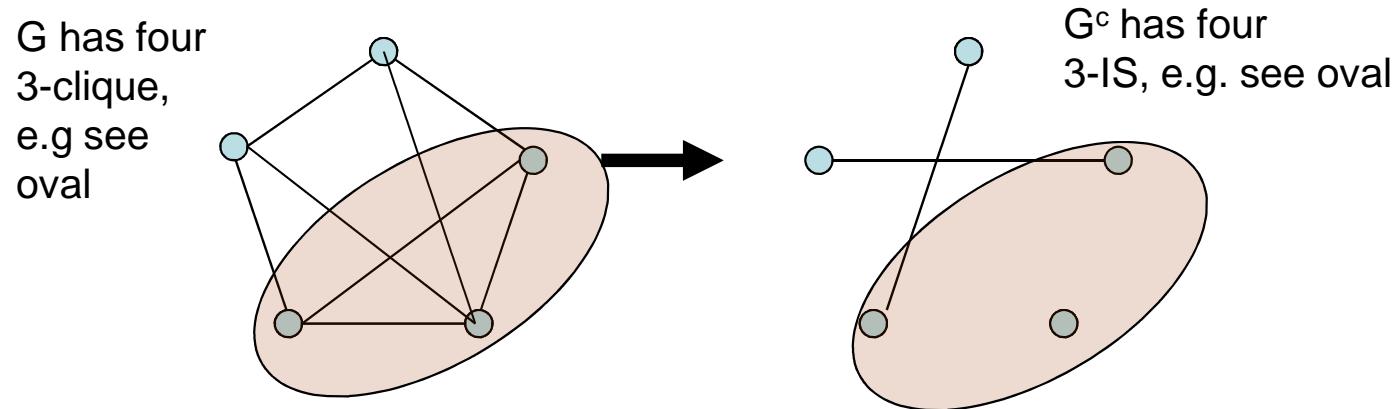


Example: Independent Set is NPC

- Definition: An k -IS, is a subset of k vertices of G with no edges between them.
- Decision problem: Does G have a k -IS?
- To Show k -IS is NPC
- Method: Reduce k -clique to k -IS

Reduce Clique to IS

- Independent set is the *dual problem* of Clique.
- Convert G to G^c (complement graph).
- G has a k -clique iff G^c has k -IS



Theorem: HAMILTONIAN-PATH
is NP-complete

Proof:

1. HAMILTONIAN-PATH is in NP
Can be easily proven
2. We will reduce in polynomial time
3CNF-SAT to HAMILTONIAN-PATH
(NP-complete)

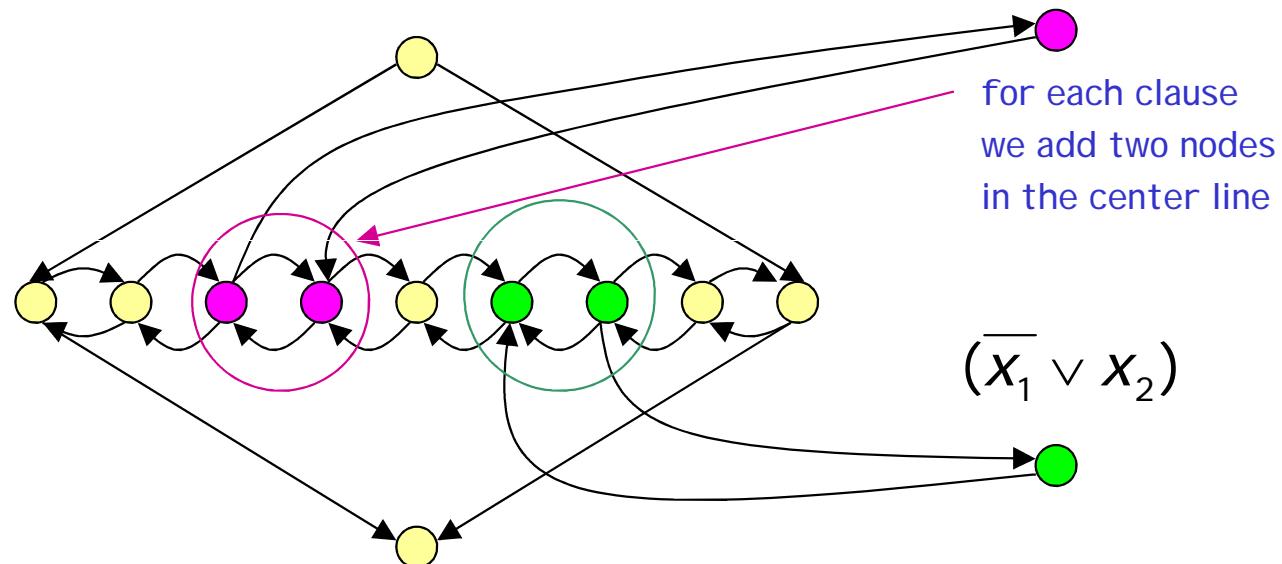
3SAT to H-Path: Reduction explained

- For each variable we will build a small graph called a "gadget".
- We will stack the gadgets for each variable
- The h-path will pass through each gadget,
 - Starting at the top,
 - Passing through the center (left to right OR right to left), and
 - Ending at the bottom.

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

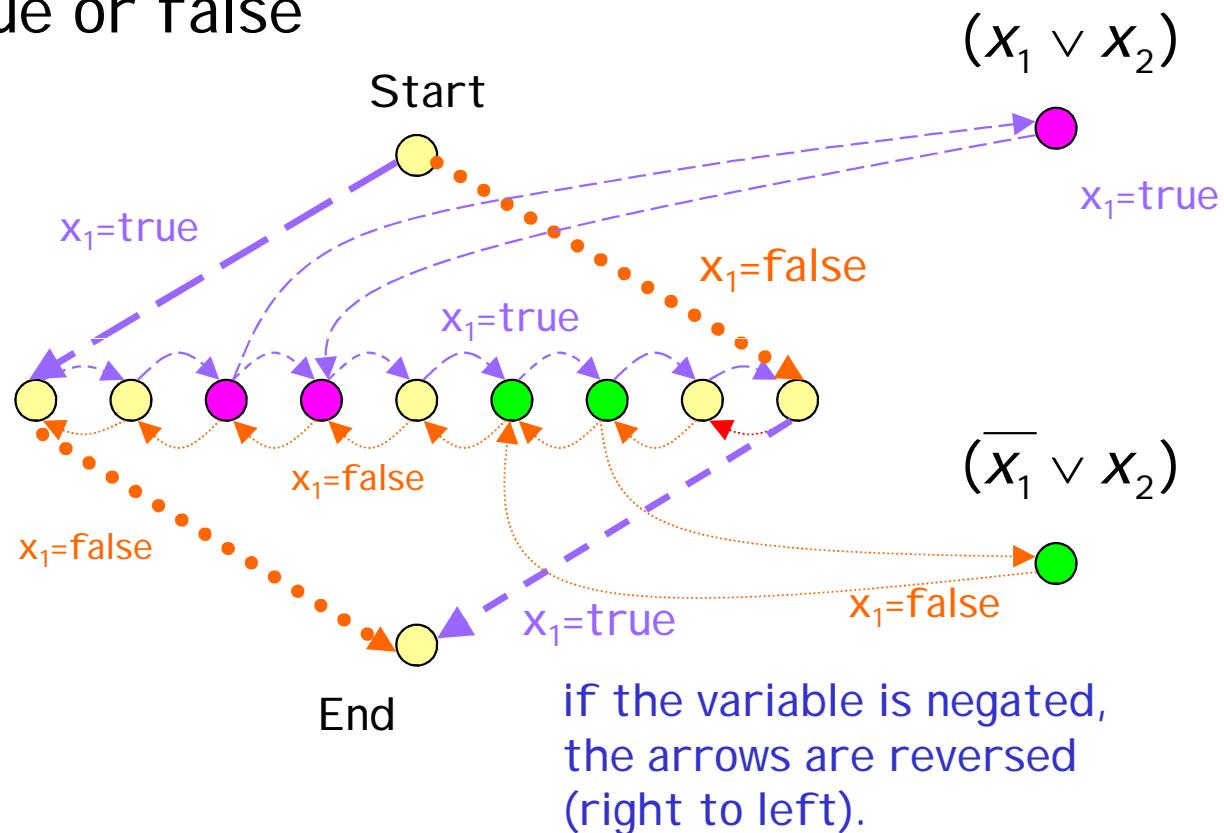
clause 1 clause 2

Gadget for variable x_1



if the variable is negated,
the arrows are reversed
(right to left).

There are only 2 possible h-paths in gadget for x_1 , depending on whether x_1 is true or false

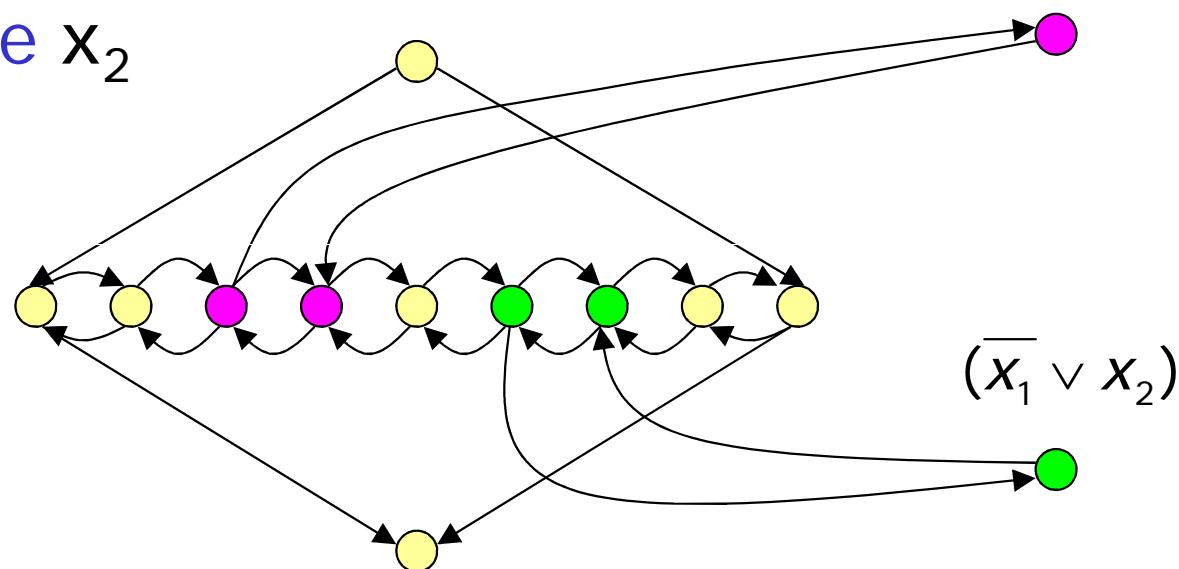


3SAT to H-Path: Reduction explained

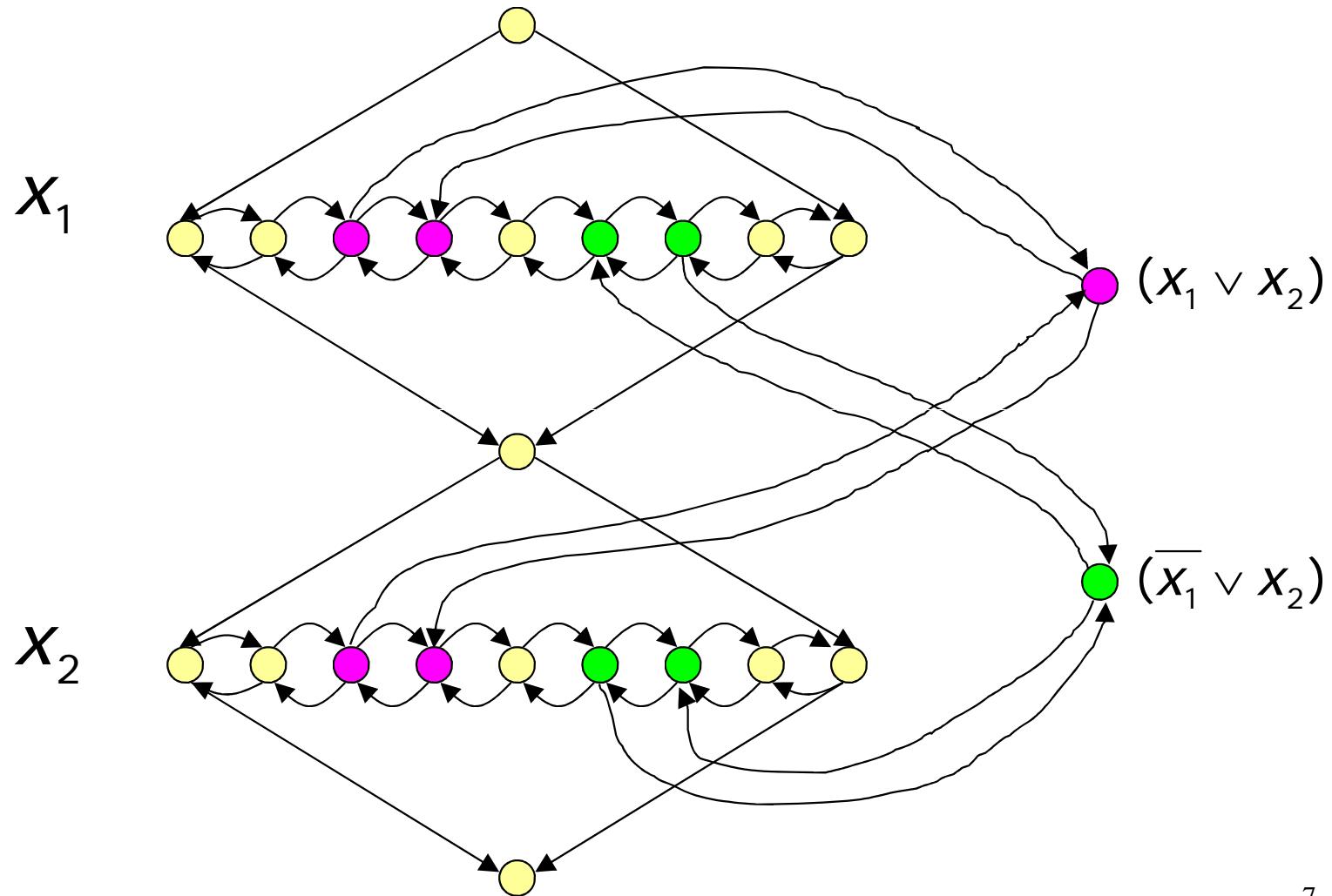
- The h-path can only go in one direction:
 - Left to Right (meaning $X=1$), visiting clauses where x is used
 - Right to Left (meaning $X=0$), visiting clauses where $\neg x$ is used
- h-path goes thru a clause iff the clause is true.
- h-path can go thru all clauses iff
 - 3SAT is true.
 - Path is Hamiltonian.

$$(x_1 \vee x_2) \wedge (\overline{x}_1 \vee x_2)$$

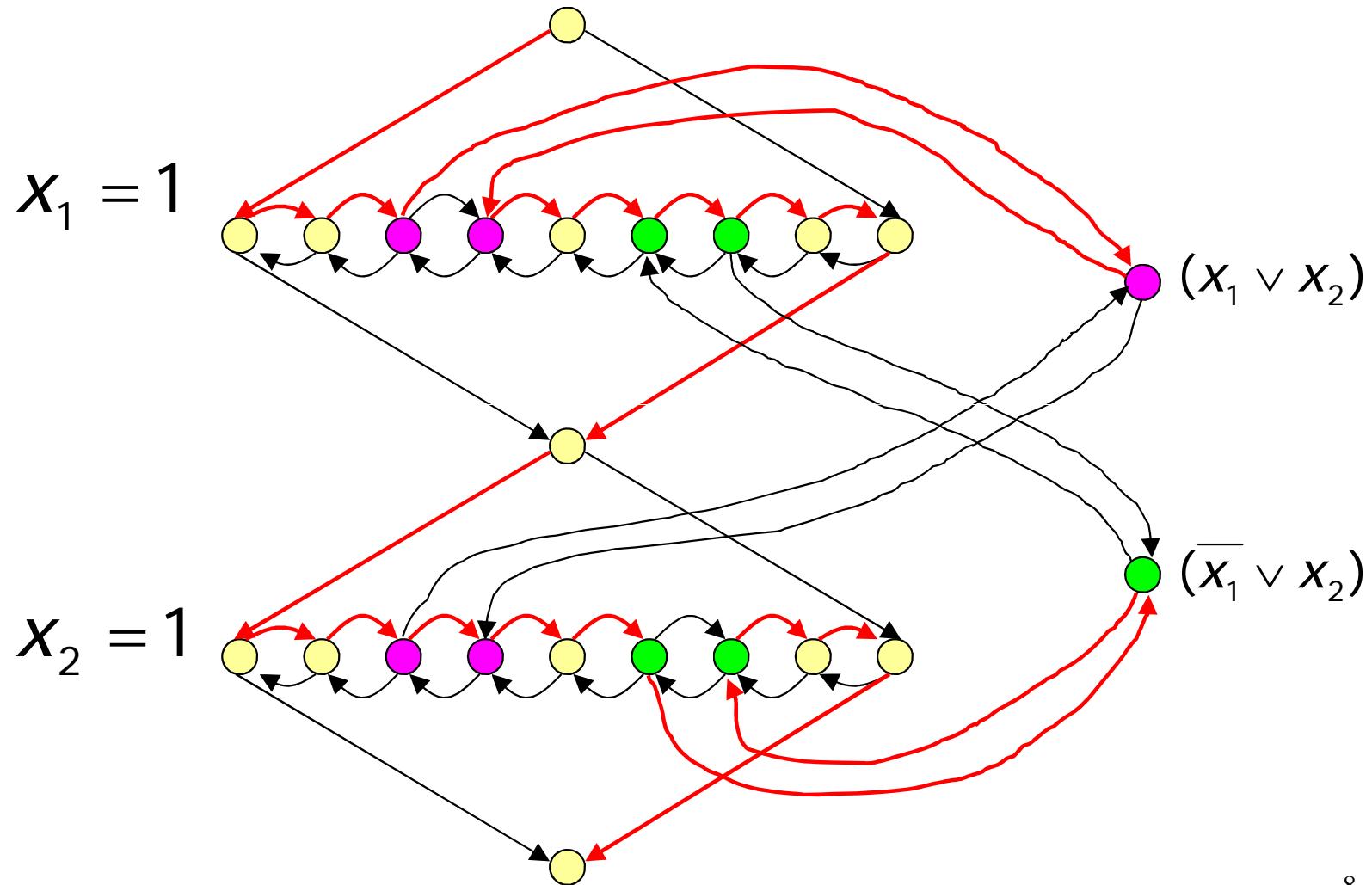
Gadget for
variable x_2



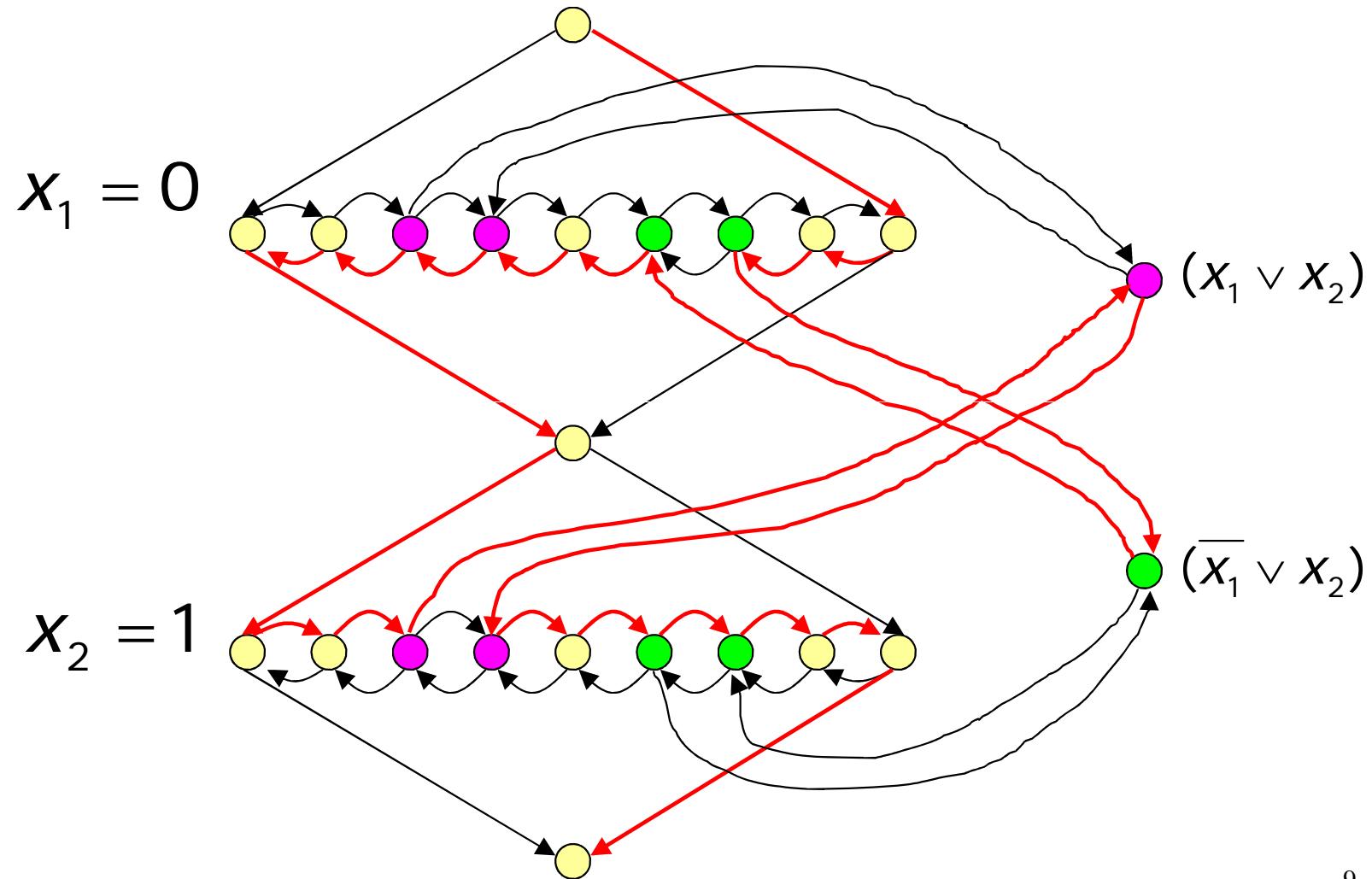
$$(x_1 \vee x_2) \wedge (\overline{x}_1 \vee x_2)$$



$$(x_1 \vee x_2) \wedge (\overline{x}_1 \vee x_2) = 1$$



$$(x_1 \vee x_2) \wedge (\overline{x}_1 \vee x_2) = 1$$



Vertex Cover is NPC

Reductions

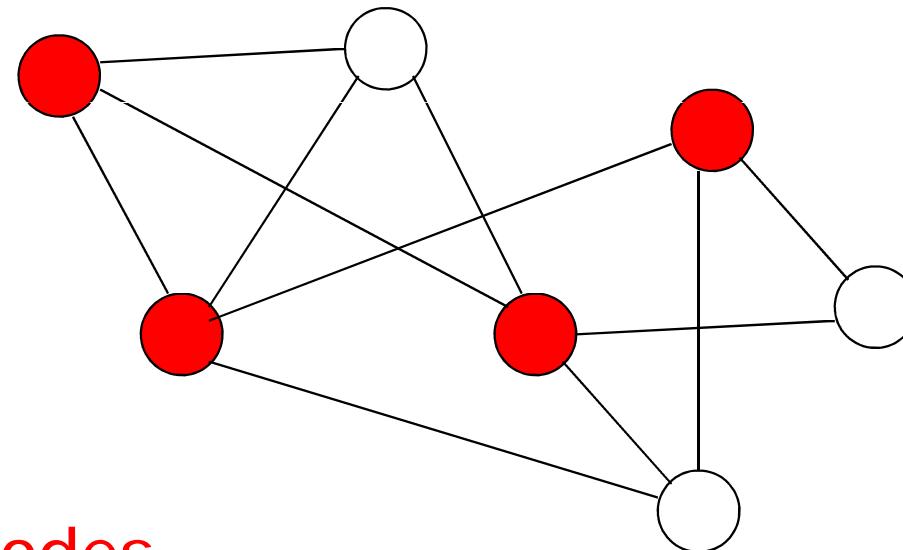
If: Language A is NP-complete
Language B is in NP
 A is polynomial time reducible to B

Then: B is NP-complete

Vertex Cover

VC of a graph is a subset S of nodes such that every edge in the graph touches one node in S

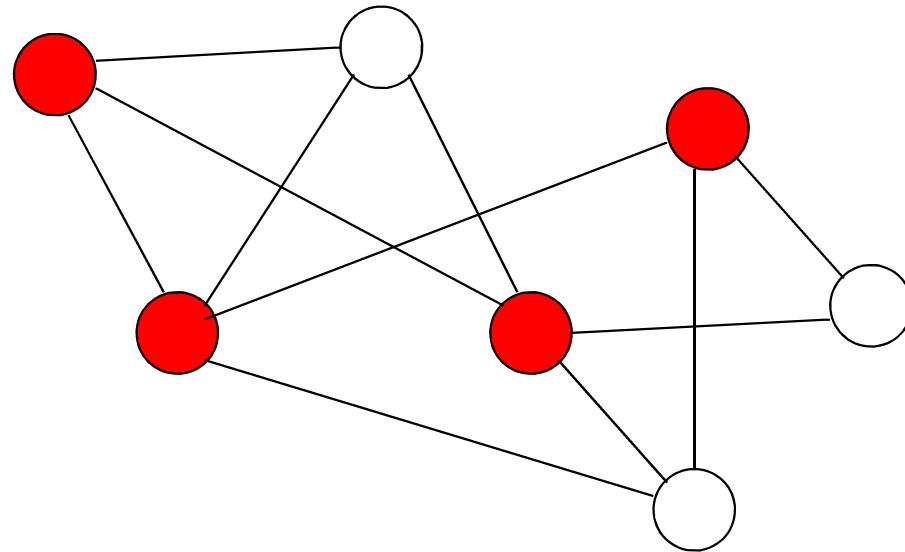
Example:



S = red nodes

Size of vertex-cover is the number
of nodes in the cover

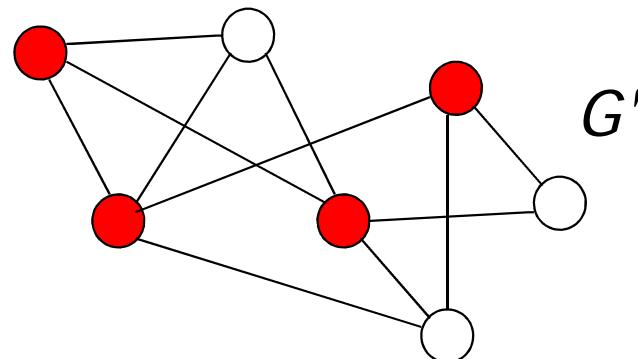
Example: $|S|=4$



Corresponding language:

VERTEX-COVER = { $\langle G, k \rangle$:
graph G contains a vertex cover
of size k }

Example:



$\langle G', 4 \rangle \in \text{VERTEX - COVER}$

Theorem: VERTEX-COVER is NP-complete

Proof:

1. VERTEX-COVER is in NP
Can be easily proven
2. We will reduce in polynomial time
3CNF-SAT to VERTEX-COVER
(NP-complete)

Let P be a 3CNF formula
with n variables and c clauses

Example:

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_4) \wedge (\overline{x}_1 \vee x_3 \vee x_4)$$

Clause 1 Clause 2 Clause 3

$n=4, c=3$

Formula P can be converted
to a graph G such that:

P is satisfiable if and only if

G Contains a vertex cover of size $k=n+2c$

3SAT to VC reduction explained

- Given a 3CNF formula P.
- We will build gadgets (small-graph) for each literal (variables x and $\neg x$).
- We will build gadgets for each clause.
- We connect the literals to the clause they appear in.

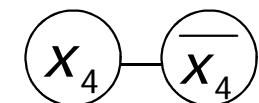
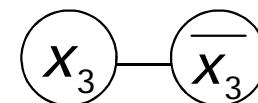
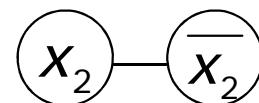
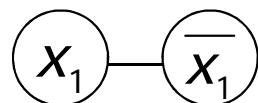
$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee x_4) \wedge (\overline{x}_1 \vee \overline{x}_3 \vee x_4)$$

Clause 1

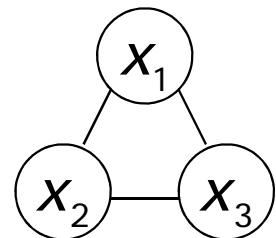
Clause 2

Clause 3

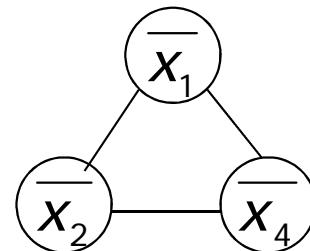
Variable Gadgets, with 2n nodes



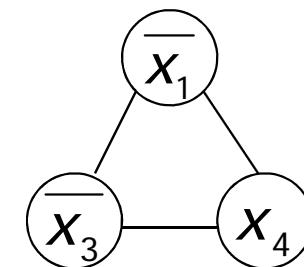
Clause Gadgets, with 3c nodes



Clause 1



Clause 2



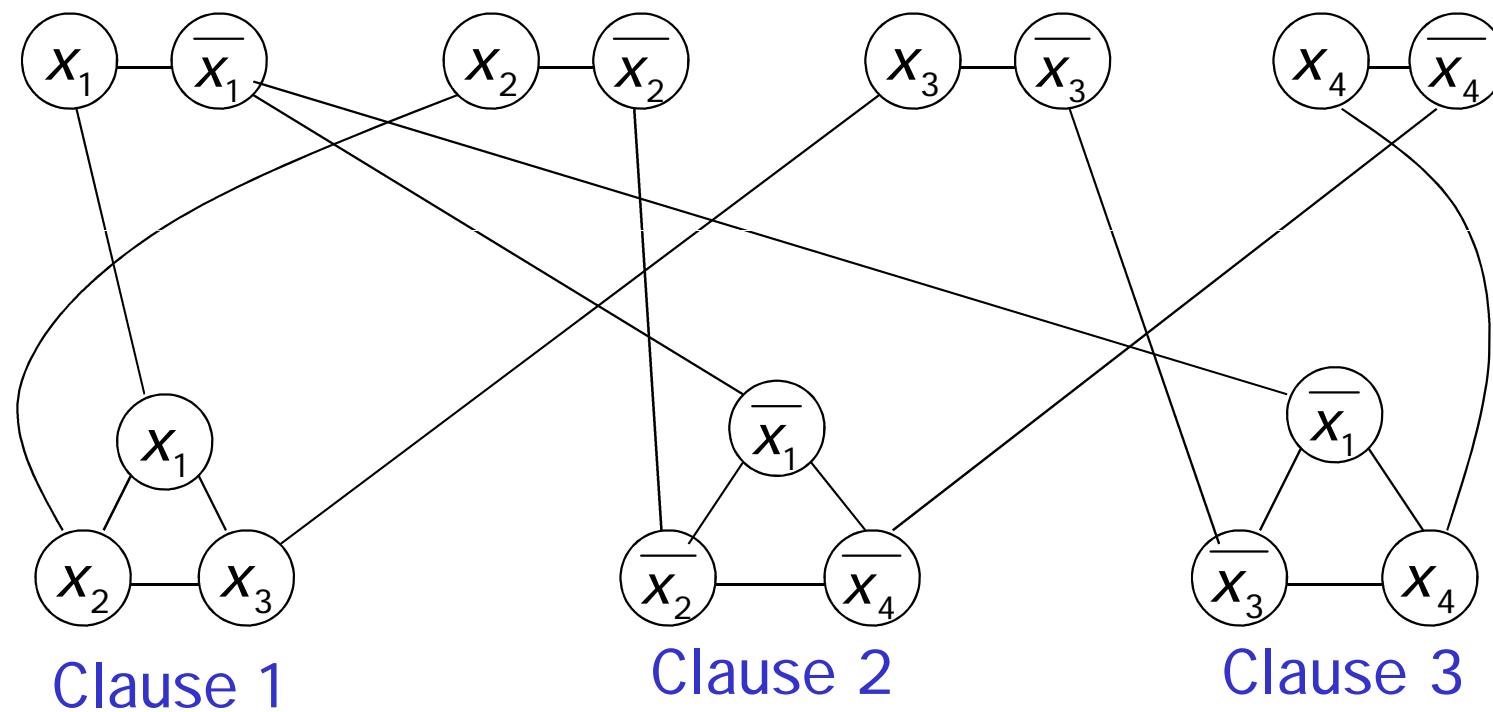
Clause 3

$$\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_4) \wedge (\overline{x}_1 \vee \overline{x}_3 \vee x_4)$$

Clause 1

Clause 2

Clause 3



First direction in proof:

If P is satisfiable,
then G contains a vertex cover of size
 $k = n + 2c$

Example:

$$P = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_4) \wedge (\overline{x}_1 \vee \overline{x}_3 \vee x_4)$$

Satisfying assignment:

$$x_1 = 1 \quad x_2 = 0 \quad x_3 = 0 \quad x_4 = 1$$

We will show that G contains
a VC of size $k=n+2c = 4+2\times 3 = 10$

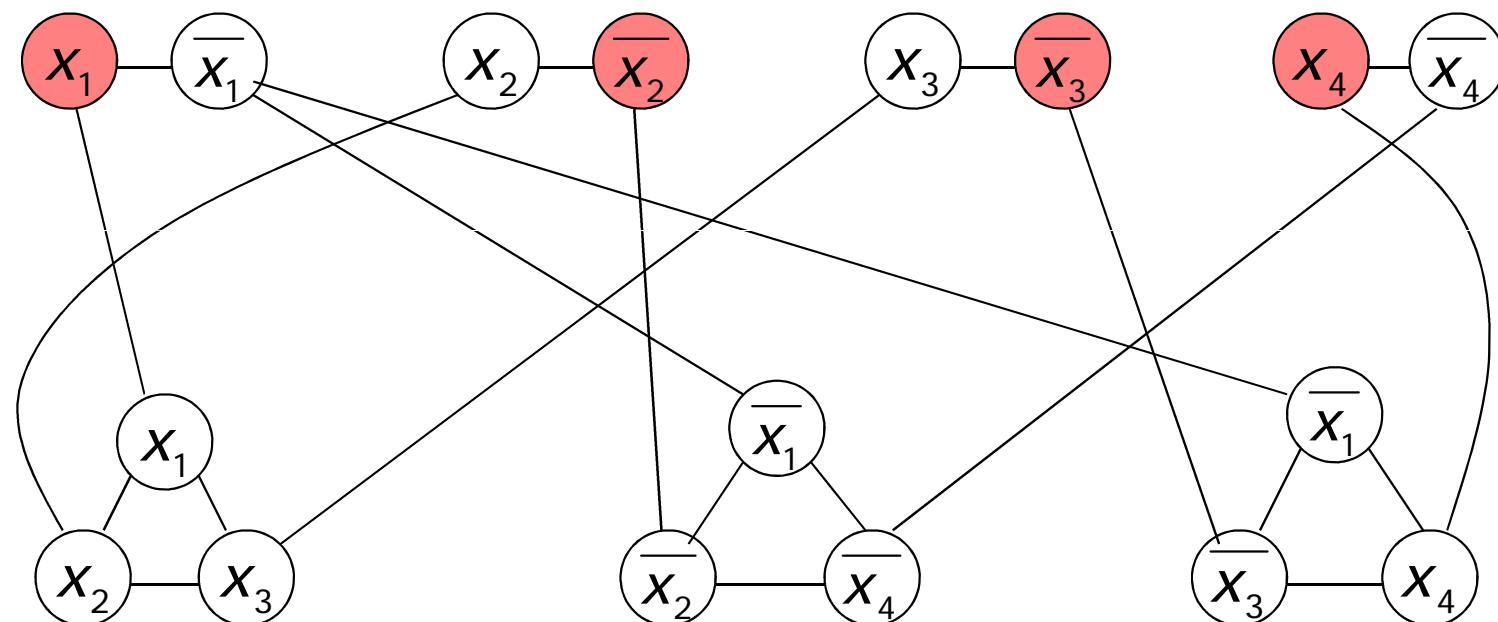
$$P = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_4) \wedge (\overline{x}_1 \vee \overline{x}_3 \vee x_4)$$

$$x_1 = 1$$

$$x_2 = 0$$

$$x_3 = 0$$

$$x_4 = 1$$



Put every satisfying literal in the cover

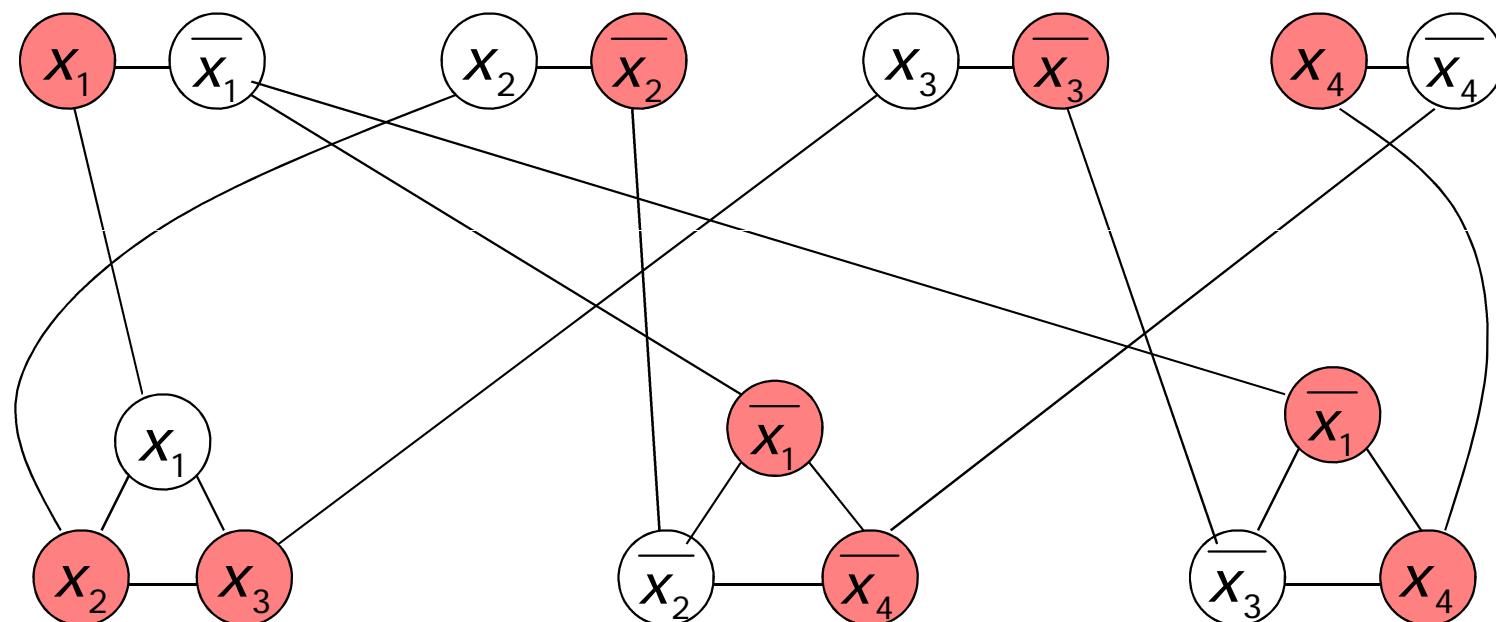
$$P = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_3} \vee x_4)$$

$$x_1 = 1$$

$$x_2 = 0$$

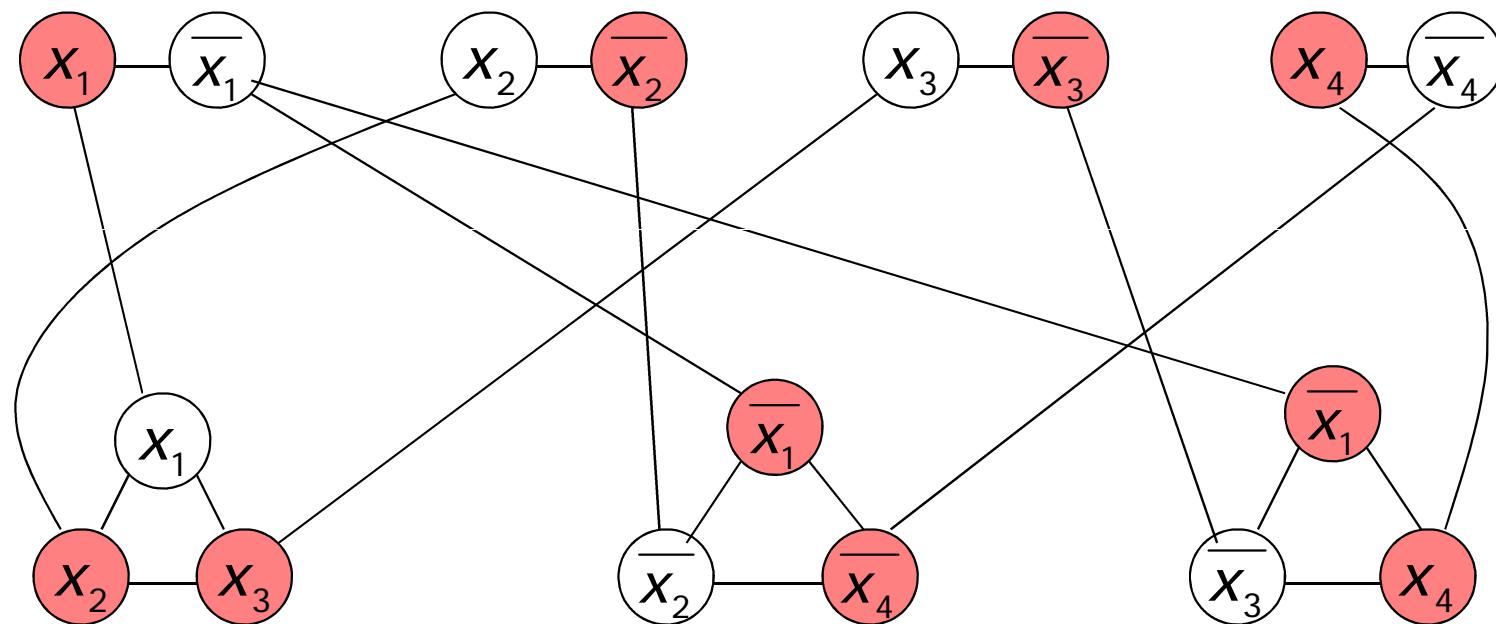
$$x_3 = 0$$

$$x_4 = 1$$

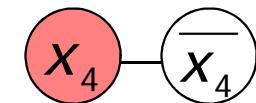
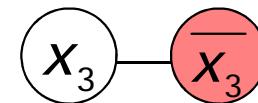
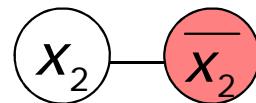
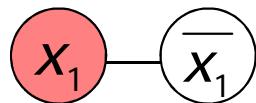


Select one satisfying literal in each clause gadget
and include the remaining literals in the cover

This is a vertex cover since every edge is adjacent to a chosen node

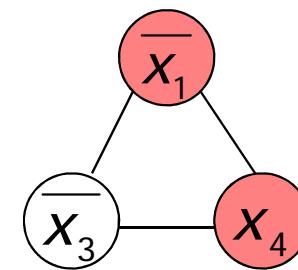
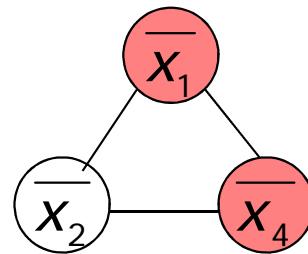
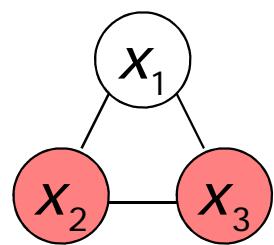


Explanation for general case:

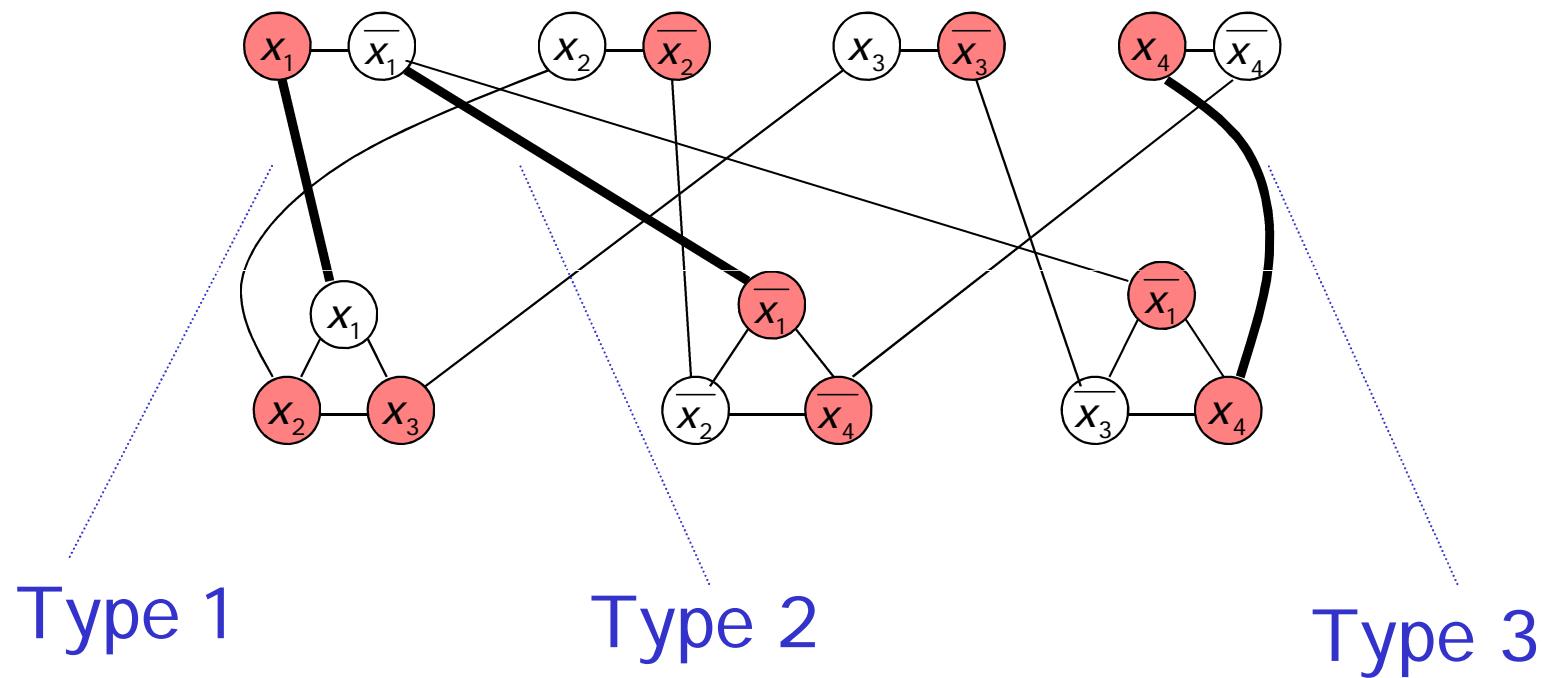


Edges in variable gadgets
are incident to at least one node in cover

Edges in clause gadgets
are incident to at least one node in cover,
since two nodes are chosen in a clause
gadget



Every edge connecting variable gadgets and clause gadgets is one of three types:

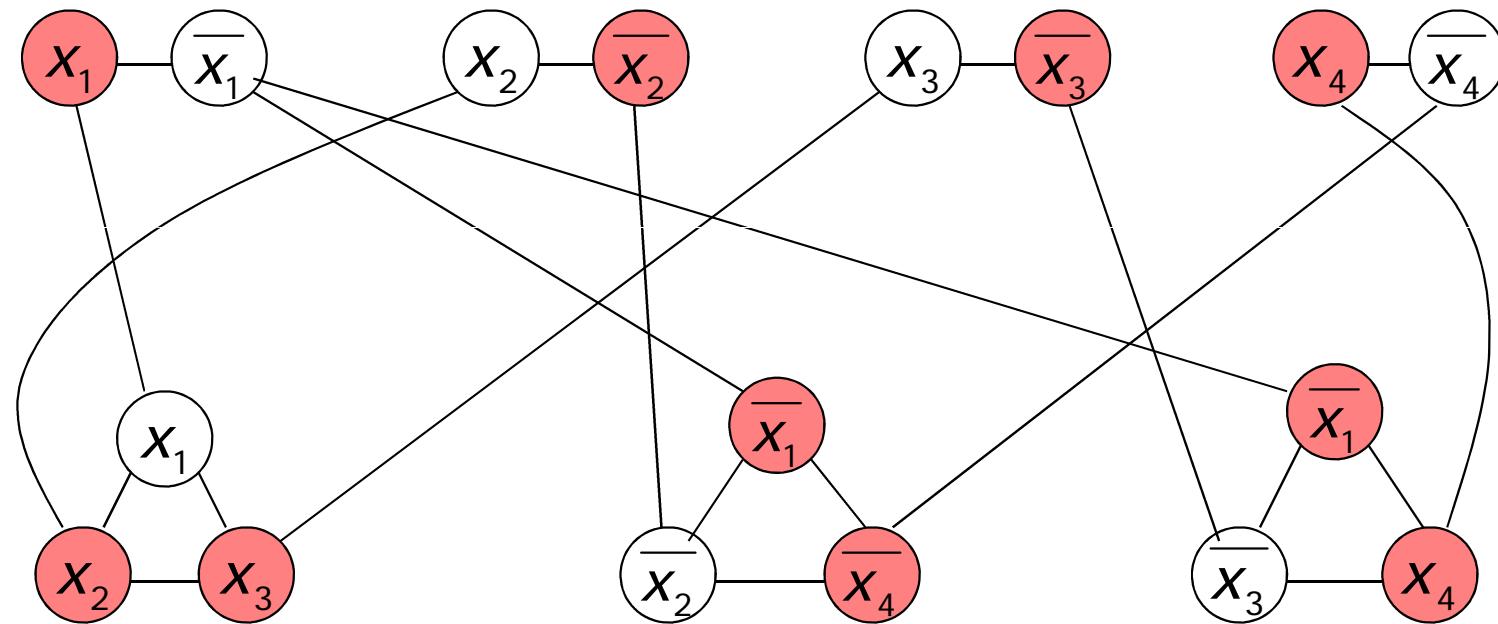


All adjacent to nodes in cover

Second direction of proof:

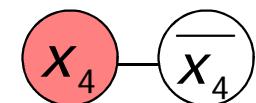
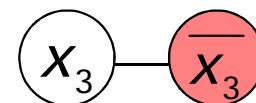
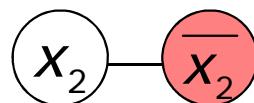
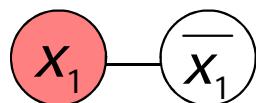
If graph G contains a vertex-cover
of size $n+2c$ then formula P is satisfiable

Example:



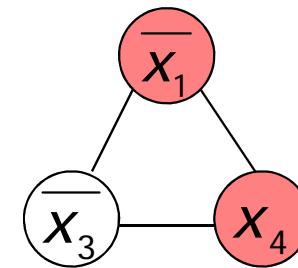
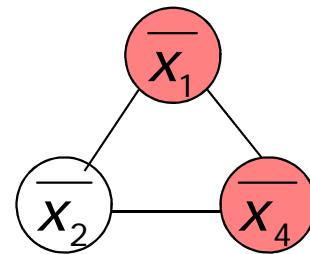
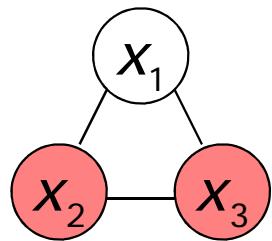
To include “internal” edges to gadgets,
and satisfy $k = n + 2c$

Exactly one literal in each variable gadget is chosen



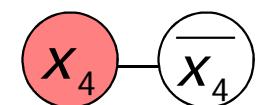
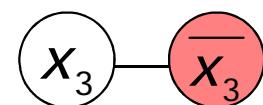
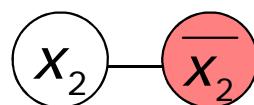
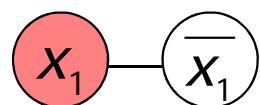
n chosen out of $2n$

Exactly two nodes in each clause gadget is chosen



$2c$ chosen out of $3c$

For the variable assignment choose the literals in the cover from variable gadgets



$$x_1 = 1$$

$$x_2 = 0$$

$$x_3 = 0$$

$$x_4 = 1$$

$$P = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_4) \wedge (\overline{x}_1 \vee \overline{x}_3 \vee x_4)$$

$$P = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_4) \wedge (\overline{x}_1 \vee \overline{x}_3 \vee x_4)$$

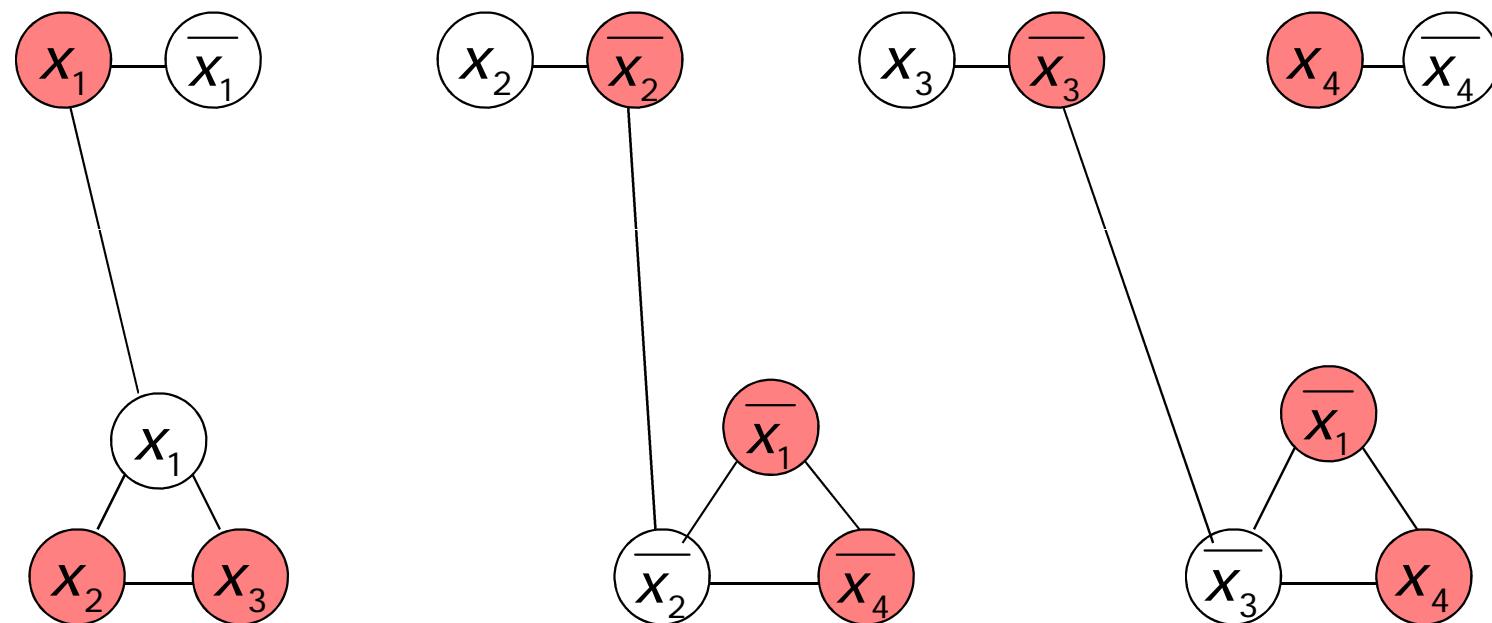
is satisfied with

$$x_1 = 1$$

$$x_2 = 0$$

$$x_3 = 0$$

$$x_4 = 1$$



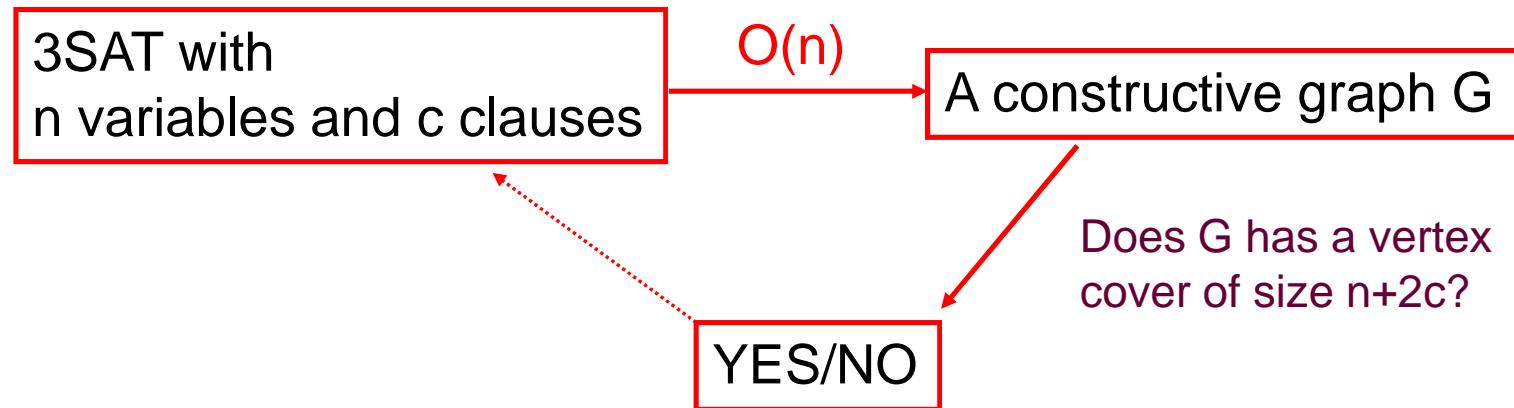
since the respective literals satisfy the clauses

Vertex cover is NPC, part 2.

VC Problem: Given a graph $G=(V,E)$, find the *smallest* number of vertexes that cover each edge

Decision problem: Does a graph G have a vertex cover of size k ?

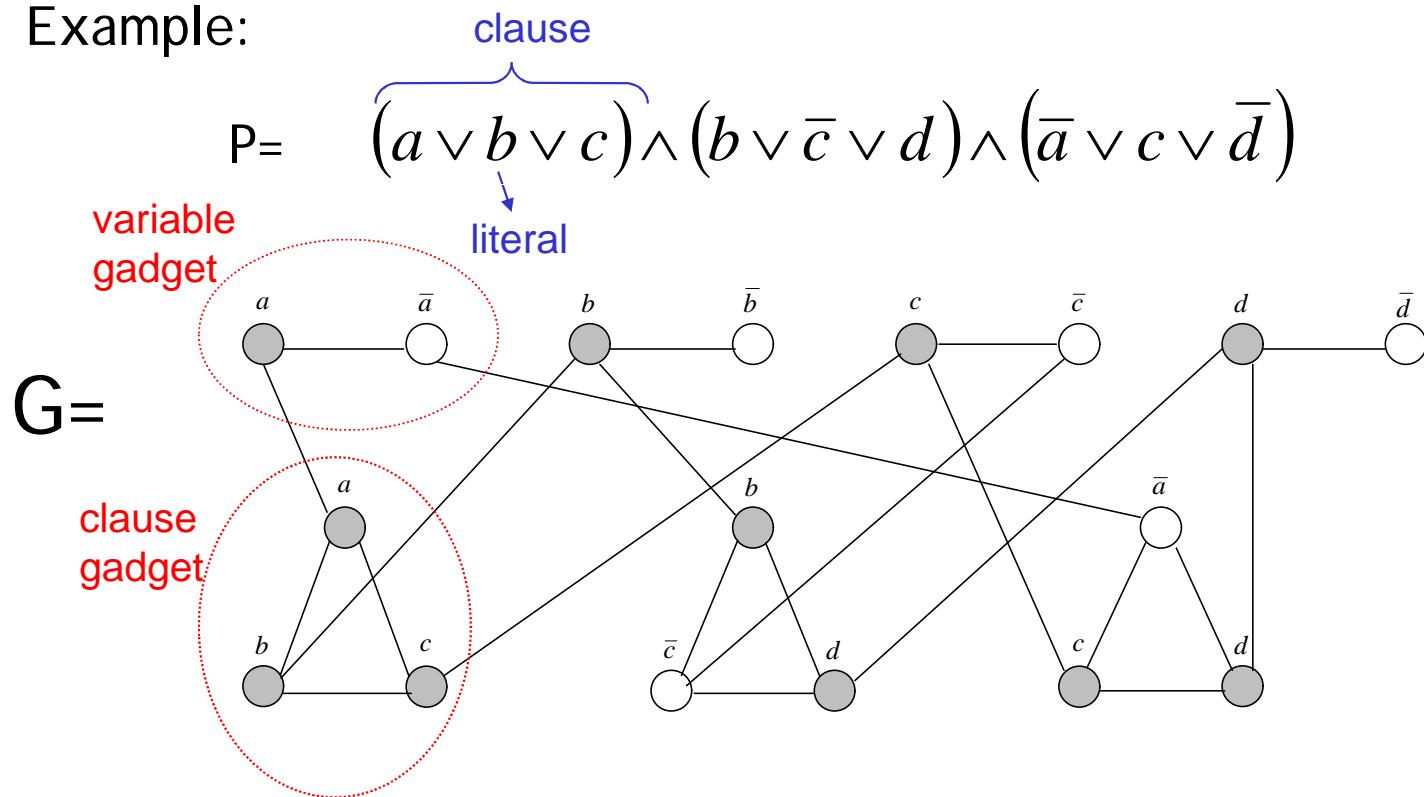
Proof of NPC, Reduce: 3SAT to VC.



Reduction: 3SAT to Vertex cover

- Given 3CNF formula P, create a gadget graph G.
- G has Vertex Cover iff P is SAT.
- P has n variables and c clauses, VC has size $n + 2c$.

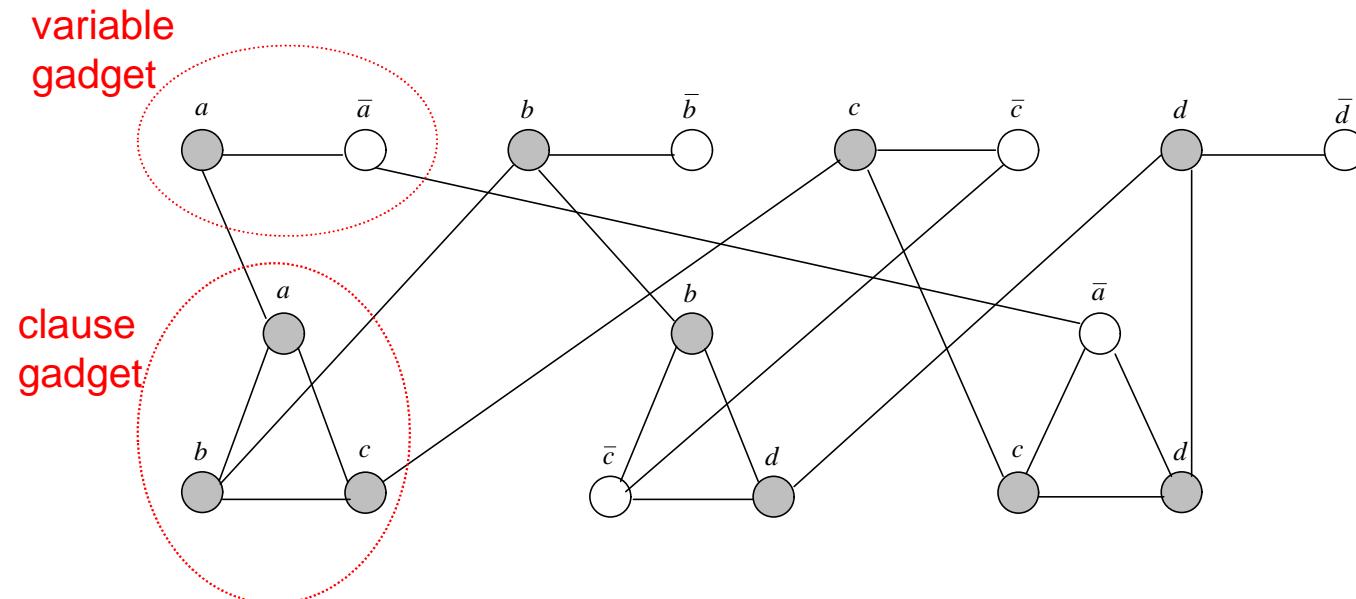
Example:



3SAT solution to VC solution

Given a solution for P, create a VC for G:

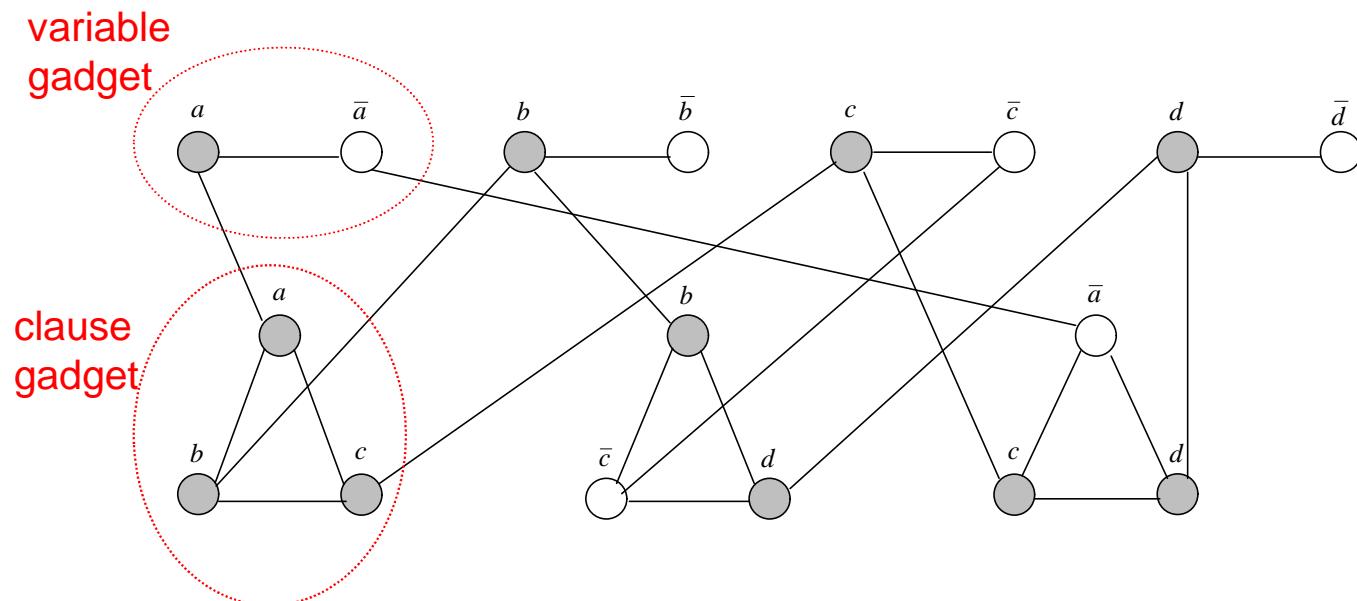
1. Put the TRUE literal in each **variable-gadget** into VC
2. No need to cover this literal again in **clause-gadgets**, as it is already covered above.
3. Put the remaining two unsatisfied literals in each **clause-gadget** in VC.
4. So we have a VC of size $n+2c$.



VC solution to 3SAT solution

Given a VC solution, create a 3SAT solution for P:

1. G has a VC of size $n+2C$, covering
 - A. One variable in each variable-gadget: Make this TRUE.
 - B. Two literals in each clause-gadget: Ignore them.
2. In each clause-gadget, the 3rd literal covered by variable-gadget is TRUE, hence all clauses are TRUE.
3. Hence P is True.



CLIQUE is NPC

3SAT to Clique polynomial-time reduction

We will reduce the 3SAT
(3CNF-satisfiability) problem
to the CLIQUE problem

Given a 3CNF formula:

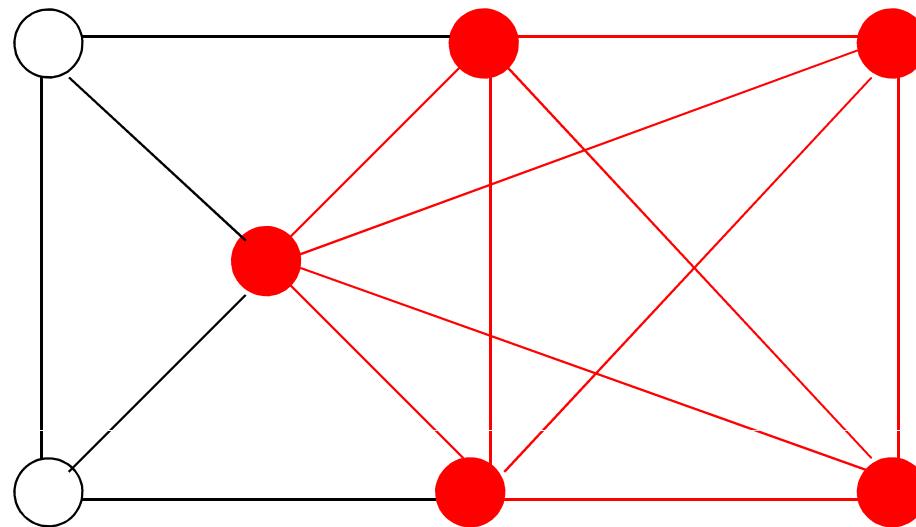
$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\underbrace{x_3 \vee \overline{x_5} \vee x_6}_{\text{clause}}) \wedge (x_3 \vee \overline{x_6} \vee x_4) \wedge (x_4 \vee x_5 \vee x_6)$$

literal

Each clause has three literals

3CNF-SAT is set of { w: w is a satisfiable 3CNF formula}

A 5-clique in graph G



CLIQUE = { $\langle G, k \rangle$: graph G
contains a k -clique}

Theorem: 3SAT is polynomial time
reducible to CLIQUE

Proof: Give a polynomial time reduction
of one problem to the other.
Transform formula to graph

3SAT to Clique reduction explained

- Given a 3CNF formula
- We will build a gadget (small graph) for each clause.
- Connect each literal in each clause to every other literal in other clauses
- Don't connect literal to its negation: x to $\neg x$
- All the true literals will form a clique.
- There is clique of size equal number of variables iff the formula is Satisfiable.

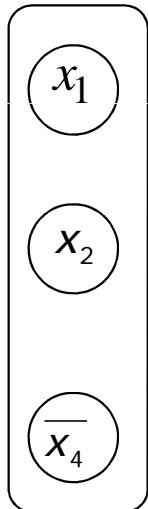
Transform formula to graph.

Example:

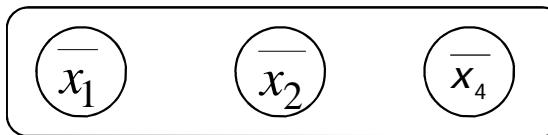
$$(x_1 \vee x_2 \vee \overline{x}_4) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_4) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x}_3 \vee \overline{x}_4)$$

Create Nodes:

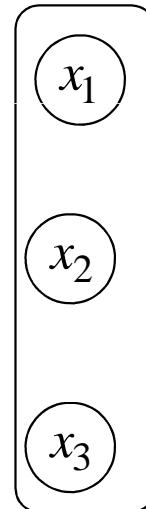
Clause 1



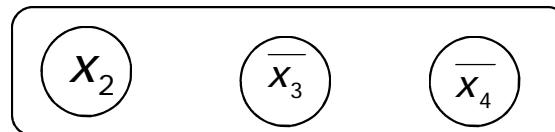
Clause 2



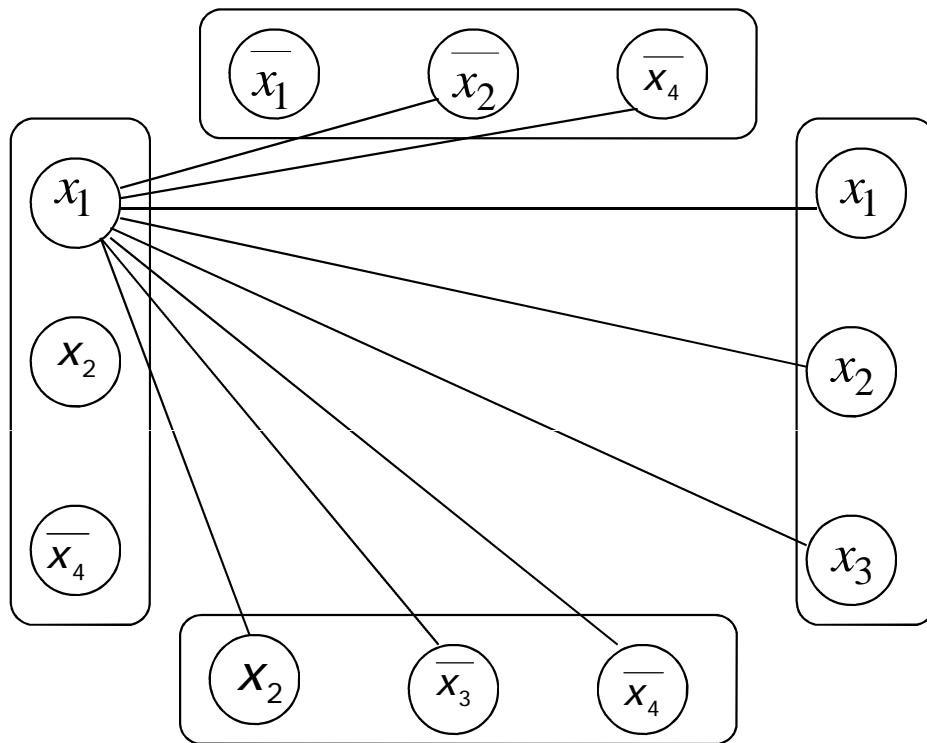
Clause 3



Clause 4

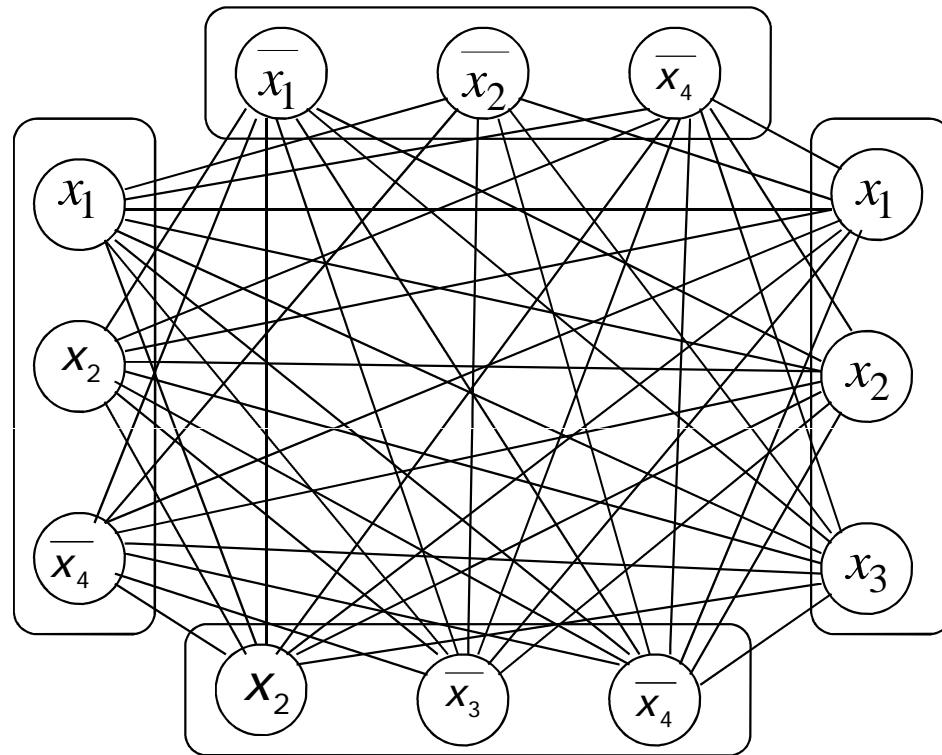


$$(x_1 \vee x_2 \vee \overline{x}_4) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_4) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x}_3 \vee \overline{x}_4)$$



Add link from a literal ξ to a literal in every other clause, except the complement $\bar{\xi}$

$$(x_1 \vee x_2 \vee \overline{x}_4) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_4) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x}_3 \vee \overline{x}_4)$$



Resulting Graph

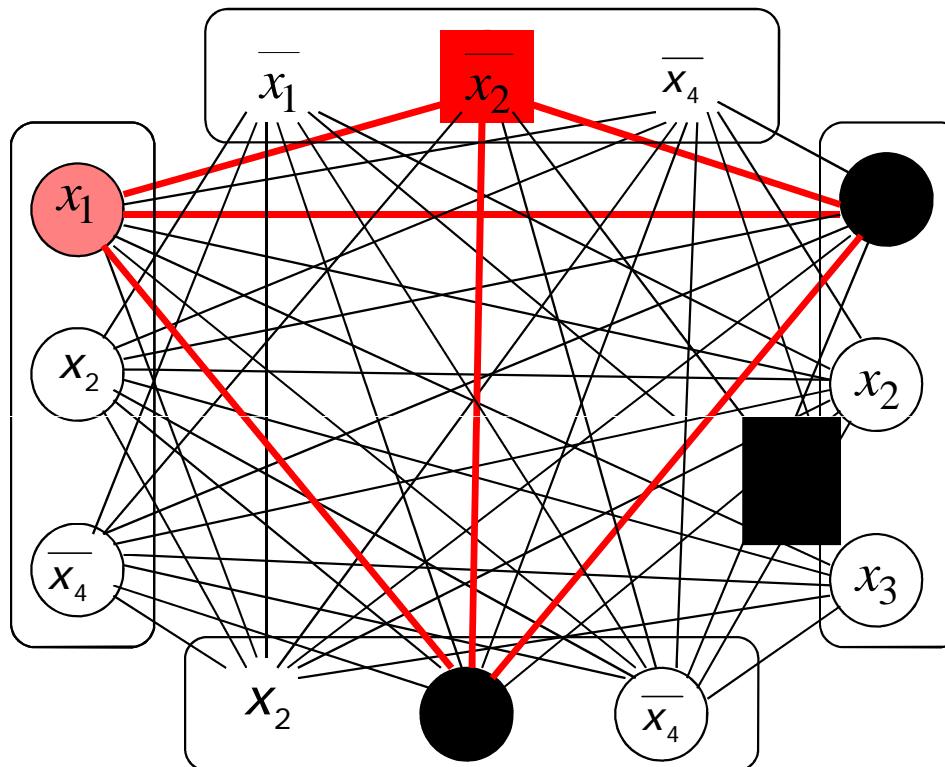
$$(x_1 \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4}) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4}) = 1$$

$$x_1 = 1$$

$$x_2 = 0$$

$$x_3 = 0$$

$$x_4 = 1$$



The formula is satisfied if and only if

the Graph has a 4-clique

End of Proof

Corollary: CLIQUE is NP-complete

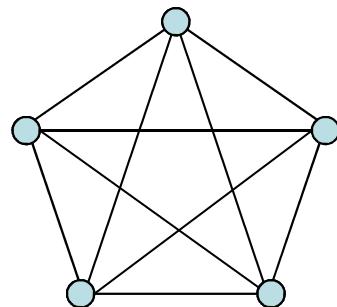
Proof:

- a. CLIQUE is in NP
- b. 3SAT is NP-complete
- c. 3SAT is polynomial reducible to CLIQUE

Clique and
Independent Set
are NPC

Example: Clique is NPC

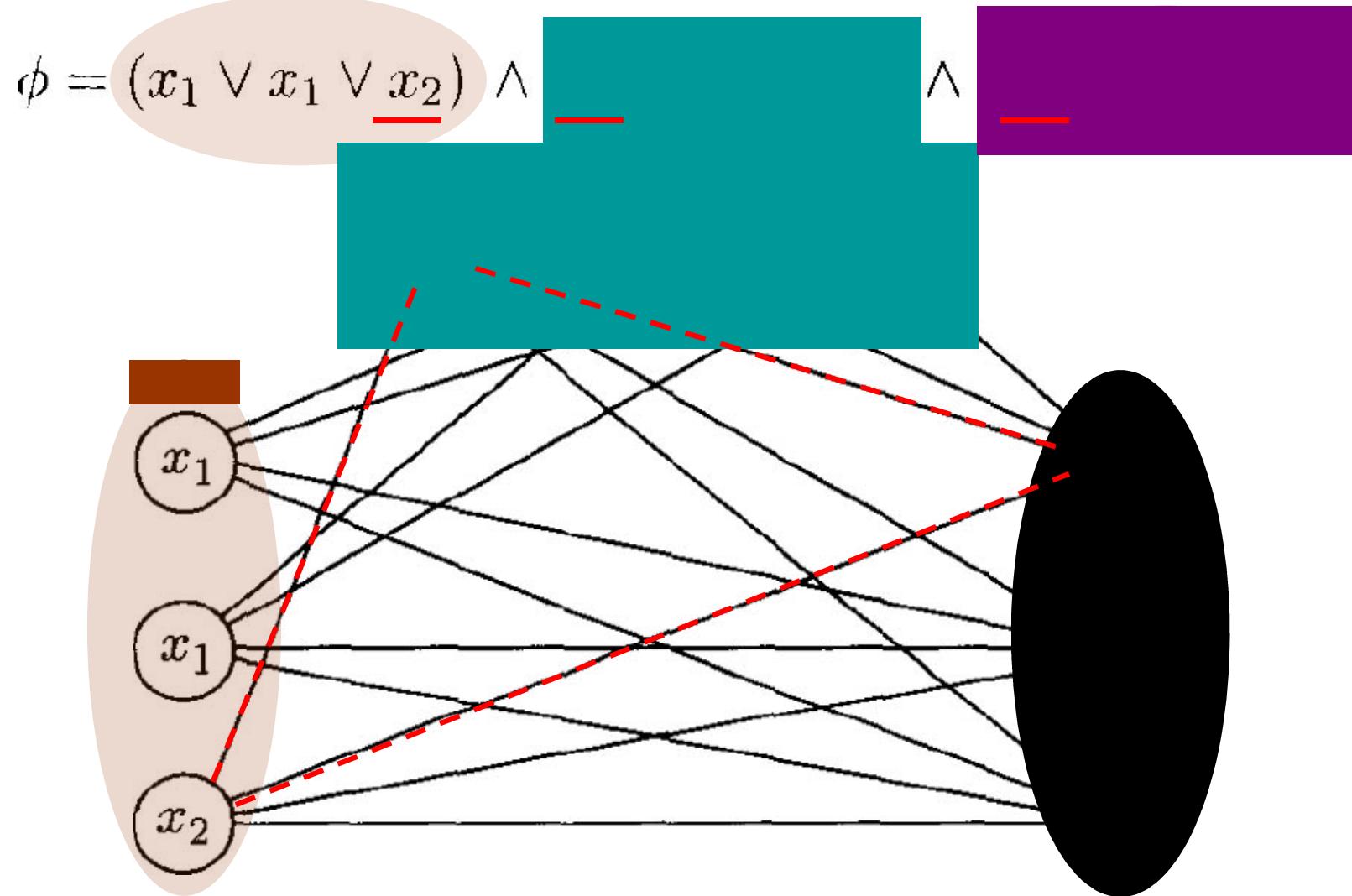
- CLIQUE = { $\langle G, k \rangle$ | G is a graph with a clique of size k }
- A clique is a subset of vertices that are all connected
- Why is CLIQUE in NP?



Reduce 3SAT to Clique

- Given a 3SAT problem Φ , with k clauses
- Make a vertex for each literal.
- Connect each vertex to the literals in other clauses that are not its negations.
- Any k -clique in this graph corresponds to a satisfying assignment (see example in next slide)

3SAT as a clique problem

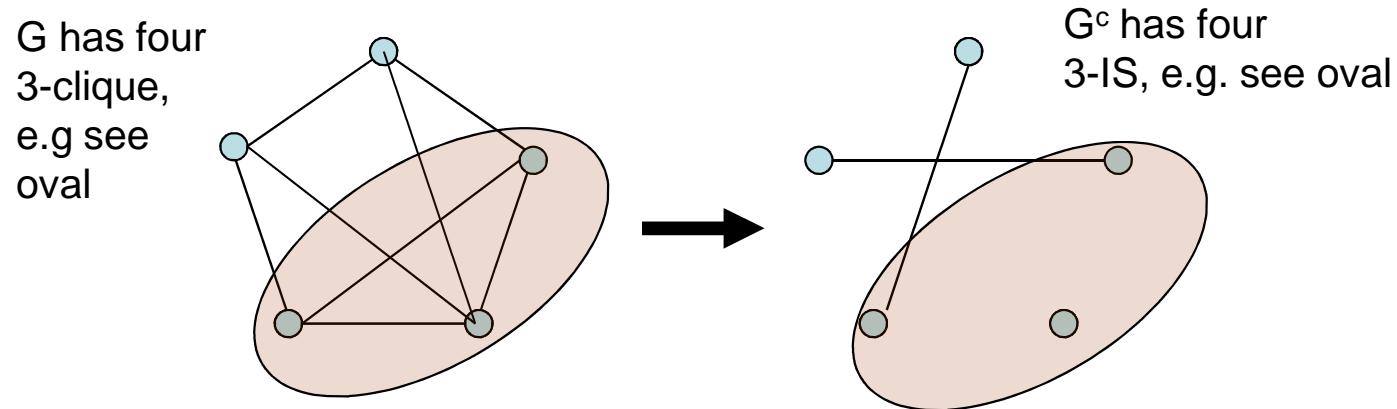


Example: Independent Set is NPC

- Definition: An k -IS, is a k subset of k vertices of G with no edges between them.
- Decision problem: Does G have a k -IS?
- To Show k -IS is NPC
- Method: Reduce k -clique to k -IS

Reduce Clique to IS

- Independent set is the *dual problem* of Clique.
- Convert G to G^c (complement graph).
- G has a k -clique iff G^c has k -IS



3SAT is NPC

SAT Reduces to 3SAT

Recall: 3CNF-SAT is a formula where each clause has 3 distinct literals.

Claim. CNF-SAT \leq_p 3CNF-SAT.

Case 1: clause C_j contains only 1 term,
add 4 new terms, and replace C_j with 4 clauses:

$$\begin{aligned} C_j &= \overline{x_3} \quad \Rightarrow \quad C'_{j1} = \overline{x_3} \vee y_1 \vee y_2 \\ C'_{j2} &= \overline{x_3} \vee \overline{y_1} \vee \overline{y_2} \\ C'_{j3} &= x_3 \vee \overline{y_1} \vee y_2 \\ C'_{j4} &= x_3 \vee y_1 \vee \overline{y_2} \end{aligned}$$

SAT Reduces to 3SAT

- Case 2: clause C_j contains exactly 3 literals,
 - nothing to do.
- Case 3: clause C_j contains 2 literals:
 - add 1 new literal y , and replace C_j with 2 clauses as follows:

$$\begin{aligned} C_j &= \overline{x_3} \vee x_7 \quad \Rightarrow \quad C'_{j1} = \overline{x_3} \vee x_7 \vee y \\ &\qquad\qquad\qquad C'_{j2} = \overline{x_3} \vee x_7 \vee \overline{y} \end{aligned}$$

Case 4 ?

SAT Reduces to 3SAT

Case 4: clause C_j contains $\ell \geq 4$ terms.

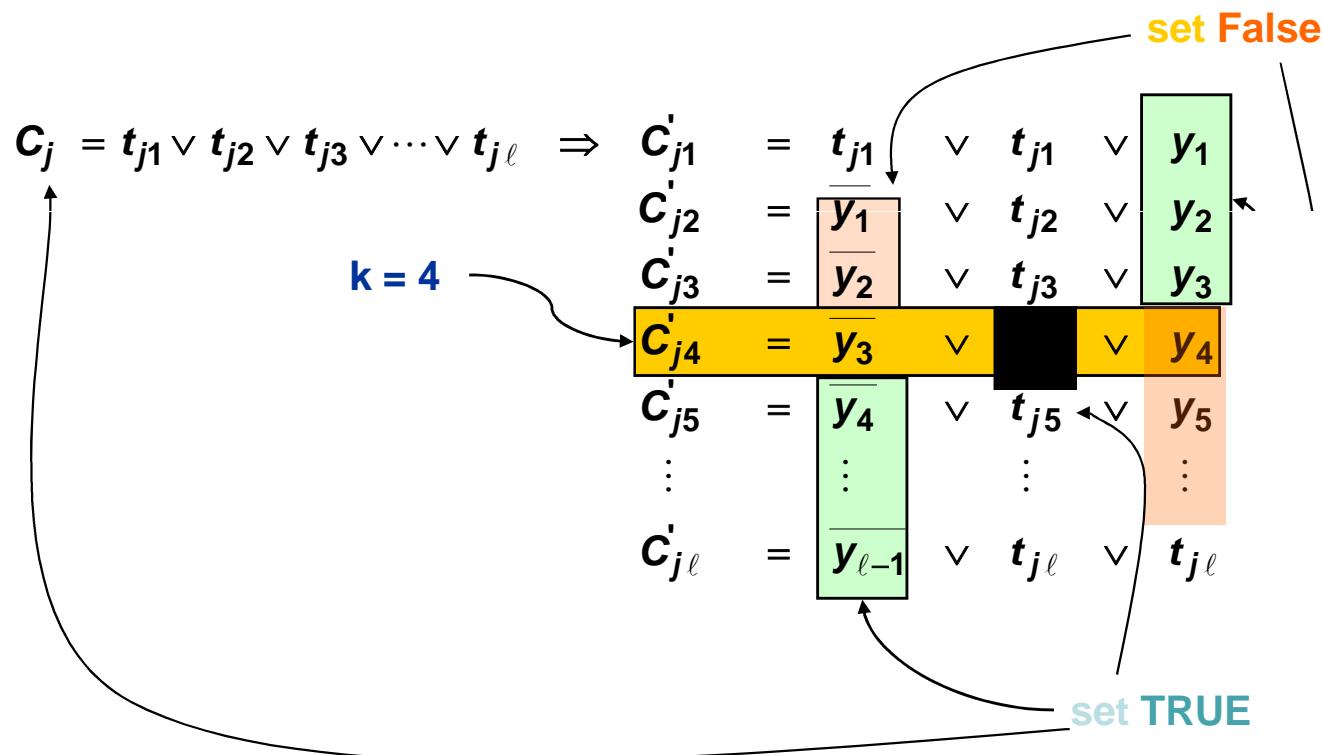
- introduce $\ell - 1$ extra Boolean variables
- replace C_j with ℓ clauses as follows:

$$C_j = x_1 \vee \overline{x_3} \vee \overline{x_4} \vee x_5 \vee x_6 \vee \overline{x_9} \Rightarrow \begin{array}{lcl} C'_{j1} & = & x_1 \vee x_1 \vee y_1 \\ C'_{j2} & = & \overline{y_1} \vee \overline{x_3} \vee y_2 \\ C'_{j3} & = & \overline{y_2} \vee \overline{x_4} \vee y_3 \\ C'_{j4} & = & \overline{y_3} \vee x_5 \vee y_4 \\ C'_{j5} & = & \overline{y_4} \vee x_6 \vee y_5 \\ C'_{j6} & = & y_5 \vee \overline{x_9} \vee \overline{x_9} \end{array}$$

SAT Reduces to 3SAT

Case 4: clause C_j contains $\ell \geq 4$ terms.

Suppose in LHS, C_j is SAT (true) because of t_{j4} is true, then in RHS, each of $C'_{j1}..C'_{j\ell}$ can be made true as follows:



SAT Reduces to 3SAT

Proof of case 4 \Rightarrow Suppose SAT instance is satisfiable.

If SAT assignment sets $t_{jk} = 1$, then

3-SAT assignment sets:

- $t_{jk} = 1$
- $y_m = 1$ for all $m < k$;
- $y_m = 0$ for all $m \geq k$

$$\begin{aligned} C_j = t_{j1} \vee t_{j2} \vee t_{j3} \vee \cdots \vee t_{j\ell} \Rightarrow \dot{C}_{j1} &= t_{j1} \quad \vee \quad t_{j1} \quad \vee \quad y_1 \\ \dot{C}_{j2} &= \overline{y_1} \quad \vee \quad t_{j2} \quad \vee \quad y_2 \\ \dot{C}_{j3} &= \overline{y_2} \quad \vee \quad t_{j3} \quad \vee \quad y_3 \\ \dot{C}_{j4} &= \overline{y_3} \quad \vee \quad t_{j4} \quad \vee \quad y_4 \\ \dot{C}_{j5} &= \overline{y_4} \quad \vee \quad t_{j5} \quad \vee \quad y_5 \\ \vdots & \quad \vdots \quad \vdots \quad \vdots \\ \dot{C}_{j\ell} &= \overline{y_{\ell-1}} \quad \vee \quad t_{j\ell} \quad \vee \quad t_{j\ell} \end{aligned}$$

SAT Reduces to 3SAT

Case 4 **Proof.** \Leftarrow Suppose 3-SAT instance is satisfiable, show SAT is true.

If 3SAT sets $t_{jk} = 1$ then

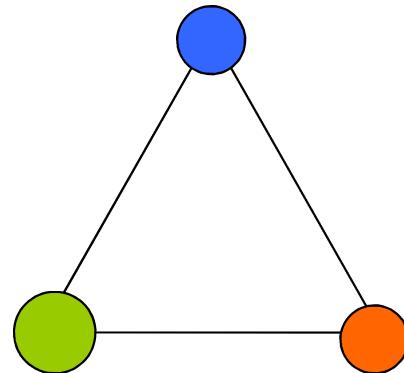
we claim $t_{jk} = 1$ for some k in C_j , because:

- each of $\ell - 1$ new Boolean variables y_j can make $\ell - 2$ of new clauses true.
- ONE remaining clause must be satisfied by an original term t_{jk}

$$\begin{aligned} C_j = t_{j1} \vee t_{j2} \vee t_{j3} \vee \cdots \vee t_{j\ell} &\Rightarrow C'_j = \overline{t_{j1}} \vee t_{j1} \vee y_1 \\ C'_{j2} &= \overline{y_1} \vee t_{j2} \vee y_2 \\ C'_{j3} &= \overline{y_2} \vee t_{j3} \vee y_3 \\ C'_{j4} &= \overline{y_3} \vee t_{j4} \vee y_4 \\ C'_{j5} &= \overline{y_4} \vee t_{j5} \vee y_5 \\ \vdots & \quad \vdots \quad \vdots \quad \vdots \\ C'_{j\ell} &= \overline{y_{\ell-1}} \vee t_{j\ell} \vee t_{j\ell} \end{aligned}$$

3Coloring is NPC

- 3Coloring a graph is NPC (hard).
- 2Coloring a graph is in P (easy).

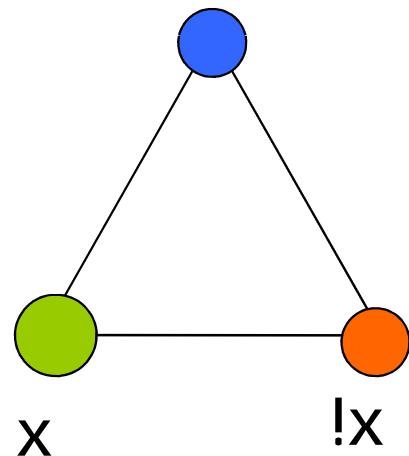


Reduce 3SAT to 3Color

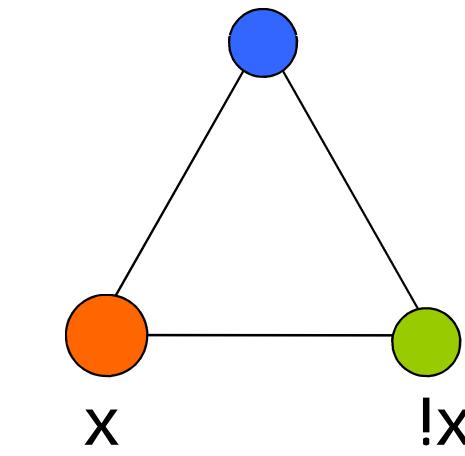
- Given a 3SAT formula S , we build a graph G , such that S is SAT iff G is 3-colorable.
- We use the colors { **True** , **False** , **Blue** } to color the graph

3-Coloring a triangle

If the top is colored blue, then there are only two possibilities for the lower nodes:



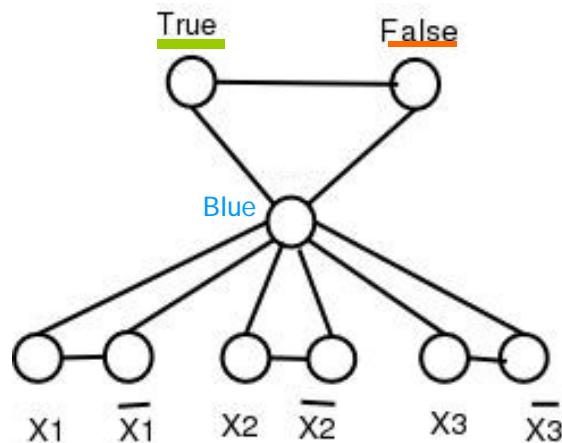
Case 1. $x=True$, $\neg x=False$



Case 2. $x=False$, $\neg x=True$

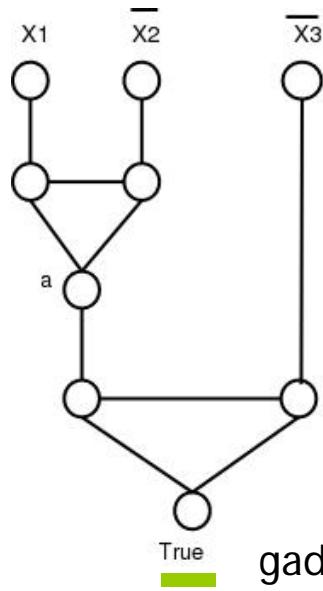
Reduce 3SAT to 3Color

- For the variables build this gadget. Color the center node **Blue**, connected to **True** and **False** above it, and the variables connected below it.
- 3 coloring this ensures that each variable x will be colored **True** or **False** only (and \bar{x} will be opposite of x).



3SAT to 3Color

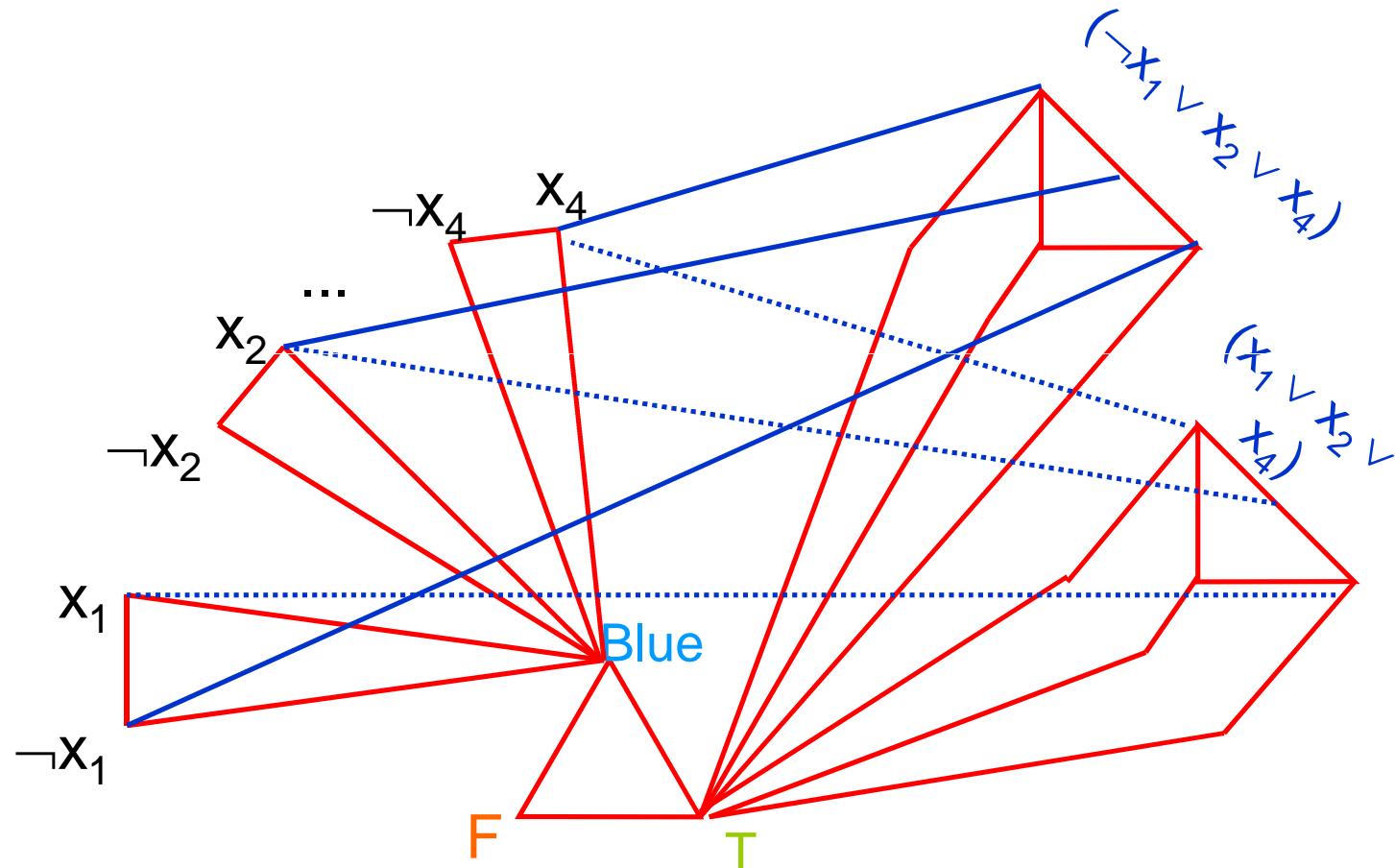
- For each clause, build this gadget, with output connected to color **True**.
- The only ways to **3-color** this is to use the **color True** for at least one of the vertices at the top (so each clause is true).
- Conversely, as long as at least one of top vertex is colored **True**, the rest of the graph can be 3-colored.



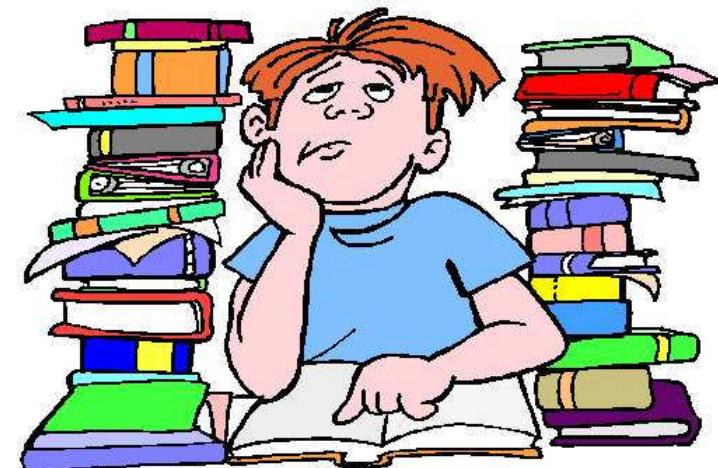
gadget for clause: $(x_1 \text{ or } \neg x_2 \text{ or } \neg x_3)$.

Example: $(\neg x_1 \vee x_2 \vee x_4) \vee (x_1 \vee x_2 \vee x_4)$

Connect all the clauses' input to the literals in it,
and join its output to **True**



NP Complete Homework



ILP is Integer Linear Programming

- Give a set of variables $\{x_1..x_n\}$ with values in integers.
- A set of m linear constraints of the form:
 - $a_{11} * x_1 + \dots + a_{1n} * x_n \text{ cmp } b_1$
 - $a_{m1} * x_1 + \dots + a_{mn} * x_n \text{ cmp } b_m$
 - Where $a[1..m, 1..n]$, $b[1..m]$ are integers,
 - cmp is { $<$, $>$, \geq , \leq , $=$ }
- Given an ILP problem, does it have an integer solution?

Q1. Reduce 3SAT to ILP

- Given 3CNF SAT with $\{b_1..b_n\}$ boolean variables, and 3CNF formulas $(b_1 \text{ or } b_2 \text{ or } b_3) \& ...$
- Write these as ILP constraints:
 1. b_1 is 0 or 1 (true or false).
 2. b_1 is opposite of $\neg b_1$.
 3. Each 3CNF $(b_1 \text{ or } b_2 \text{ or } b_3)$ is 1 (true).
- Hence ILP is NPC

Q2. Reduce 3Color to ILP

- Encode color of each vertex as an integer {c₁, c₂, c₃}.
- Add ILP constraints
 1. Each vertex is only one color.
 2. For each edge(v₁, v₂), the color(v₁) != color(v₂).
- Hence ILP is NPC.

Q3. Reduce 2Partition to 3Partition

- *2Partition*: Given a set of integer weights $A=\{w_1 \dots w_n\}$, to partition A into two sets of equal sums is NPC.
- *3Partition*: Divide A into 3 equal sets.
- Hint: To solve 2partition, add a dummy weight to A, and call 3Partition. Afterwards remove the dummy weight from the 3 partitions.

Linear Algebra

Outline

- Geometric intuition for linear algebra
- Matrices as linear transformations or as sets of constraints
- Linear systems & vector spaces
- Solving linear systems
- Eigenvalues & eigenvectors

Basic concepts

- *Vector* in \mathbb{R}^n is an ordered set of n real numbers.
 - e.g. $v = (1, 6, 3, 4)$ is in \mathbb{R}^4
 - “ $(1, 6, 3, 4)$ ” is a column vector:
 - as opposed to a row vector:
- *m-by-n matrix* is an object with m rows and n columns, each entry fill with a real number:

$$\begin{pmatrix} 1 \\ 6 \\ 3 \\ 4 \end{pmatrix} \quad \xrightarrow{\hspace{1cm}} \quad (1 \ 6 \ 3 \ 4)$$
$$\begin{pmatrix} 1 & 2 & 8 \\ 4 & 78 & 6 \\ 9 & 3 & 2 \end{pmatrix} \quad \xrightarrow{\hspace{1cm}}$$

Basic concepts

- Transpose: reflect vector/matrix on line:

$$\begin{pmatrix} a \\ b \end{pmatrix}^T = (a \quad b)$$

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^T = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$$

– Note: $(Ax)^T = x^T A^T$ (We'll define multiplication soon...)

- Vector norms:

– L_p norm of $v = (v_1, \dots, v_k)$ is $(\sum_i |v_i|^p)^{1/p}$

– Common norms: L_1, L_2

– $L_{\infty} = \max_i |v_i|$

- Length of a vector v is $L_2(v)$

Basic concepts

- Vector dot product: $u \bullet v = (u_1 \ u_2) \bullet (v_1 \ v_2) = u_1 v_1 + u_2 v_2$

- Note dot product of u with itself is the square of the length of u .

- Matrix product:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

$$AB = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}$$

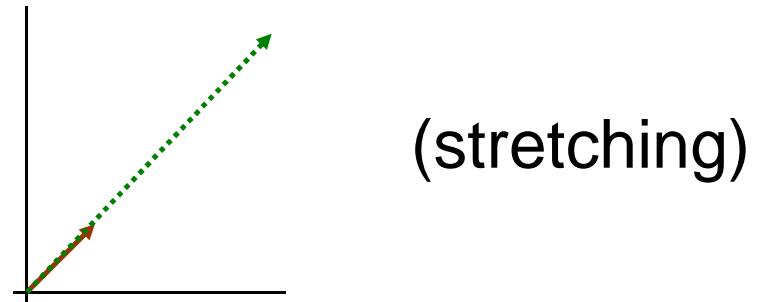
Vector products

- Dot product: $u \bullet v = u^T v = \begin{pmatrix} u_1 & u_2 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = u_1 v_1 + u_2 v_2$
- Outer product:

$$uv^T = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} (v_1 \quad v_2) = \begin{pmatrix} u_1 v_1 & u_1 v_2 \\ u_2 v_1 & u_2 v_2 \end{pmatrix}$$

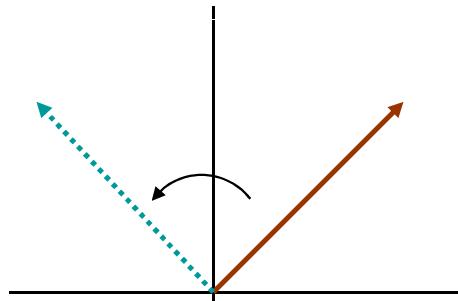
Matrices as linear transformations

$$\begin{pmatrix} 5 & 0 \\ 0 & 5 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 5 \\ 5 \end{pmatrix}$$



(stretching)

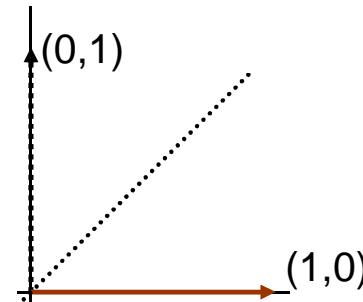
$$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$



(rotation)

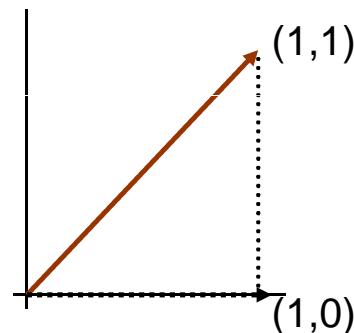
Matrices as linear transformations

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$



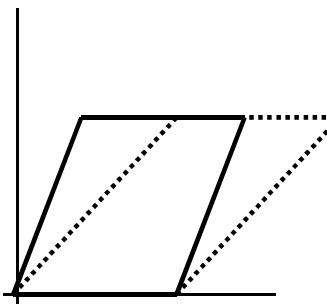
(reflection)

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$



(projection)

$$\begin{pmatrix} 1 & c \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x + cy \\ y \end{pmatrix}$$



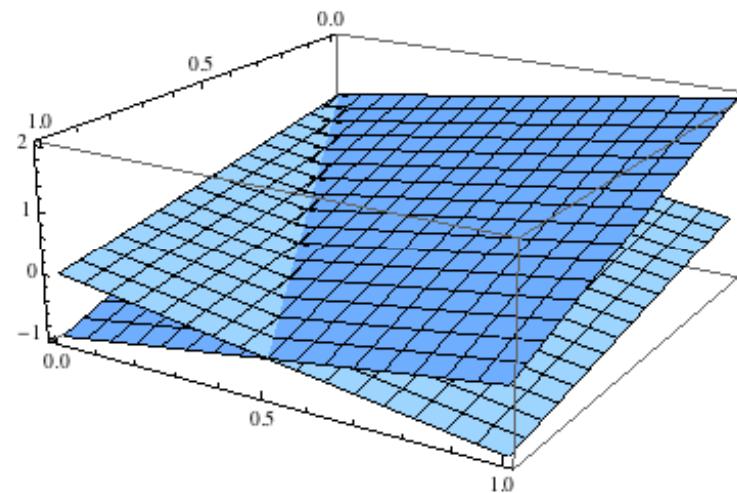
(shearing)

Matrices as sets of constraints

$$x + y + z = 1$$

$$2x - y + z = 2$$

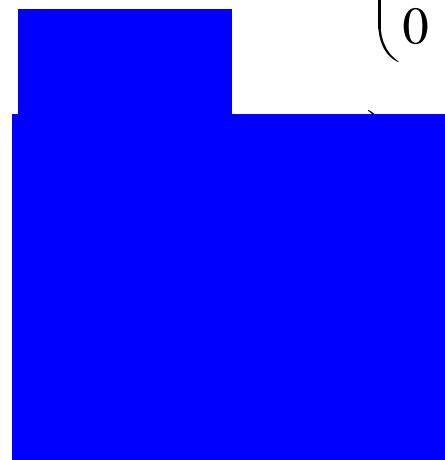
$$\begin{pmatrix} 1 & 1 & 1 \\ 2 & -1 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$



Special matrices

diagonal

$$\begin{pmatrix} & 0 & 0 \\ 0 & & 0 \\ 0 & 0 & \end{pmatrix}$$



tri-diagonal

$$\begin{pmatrix} & 0 & 0 \\ 0 & & 0 \\ 0 & 0 & \end{pmatrix}$$

$$\begin{pmatrix} a & b & c \\ 0 & d & e \\ 0 & 0 & f \end{pmatrix}$$

upper-triangular

$$\begin{pmatrix} a & 0 & 0 \\ b & c & 0 \\ d & e & f \end{pmatrix}$$

lower-triangular

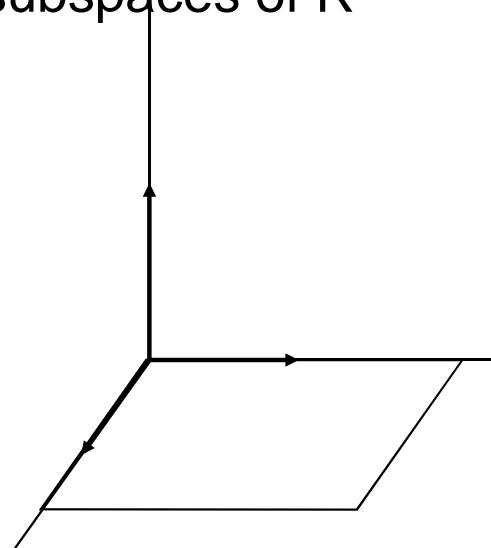
I (identity matrix)

Vector spaces

- Formally, a *vector space* is a set of vectors which is closed under addition and multiplication by real numbers.
- A *subspace* is a subset of a vector space which is a vector space itself, e.g. the plane $z=0$ is a subspace of \mathbb{R}^3 (It is essentially \mathbb{R}^2 .).
- We'll be looking at \mathbb{R}^n and subspaces of \mathbb{R}^n

Our notion of planes in \mathbb{R}^3
may be extended to
hyperplanes in \mathbb{R}^n (of
dimension $n-1$)

Note: subspaces must
include the origin (zero
vector).

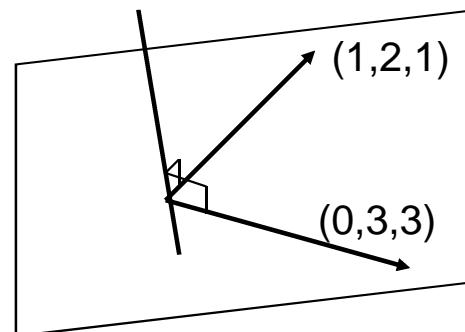


Linear system and subspaces

$$\begin{pmatrix} 1 & 0 \\ 2 & 3 \\ 1 & 3 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

$$u \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} + v \begin{pmatrix} 0 \\ 3 \\ 3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

- Linear systems define certain subspaces
- $Ax = b$ is solvable iff b may be written as a linear combination of the columns of A
- The set of possible vectors b forms a subspace called the *column space* of A



Linear system & subspaces

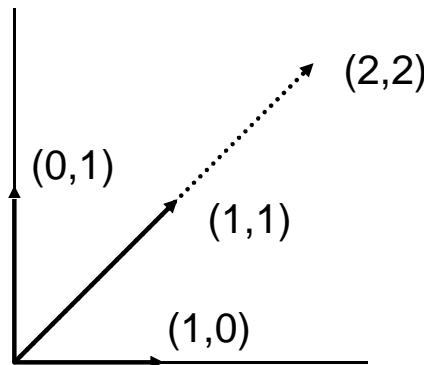
The set of solutions to "Ax = 0" forms a subspace called the *null space* of A.

$$\begin{pmatrix} 1 & 0 \\ 2 & 3 \\ 1 & 3 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \rightarrow \text{Null space: } \{(0,0)\}$$

$$\begin{pmatrix} 1 & 0 & 1 \\ 2 & 3 & 5 \\ 1 & 3 & 4 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \rightarrow \text{Null space: } \{(c,c,-c)\}$$

Linear independence and basis

- Vectors v_1, \dots, v_k are **linearly independent** if $(c_1v_1 + \dots + c_kv_k = 0)$ implies $(c_1 = \dots = c_k = 0)$



i.e. the nullspace is the origin

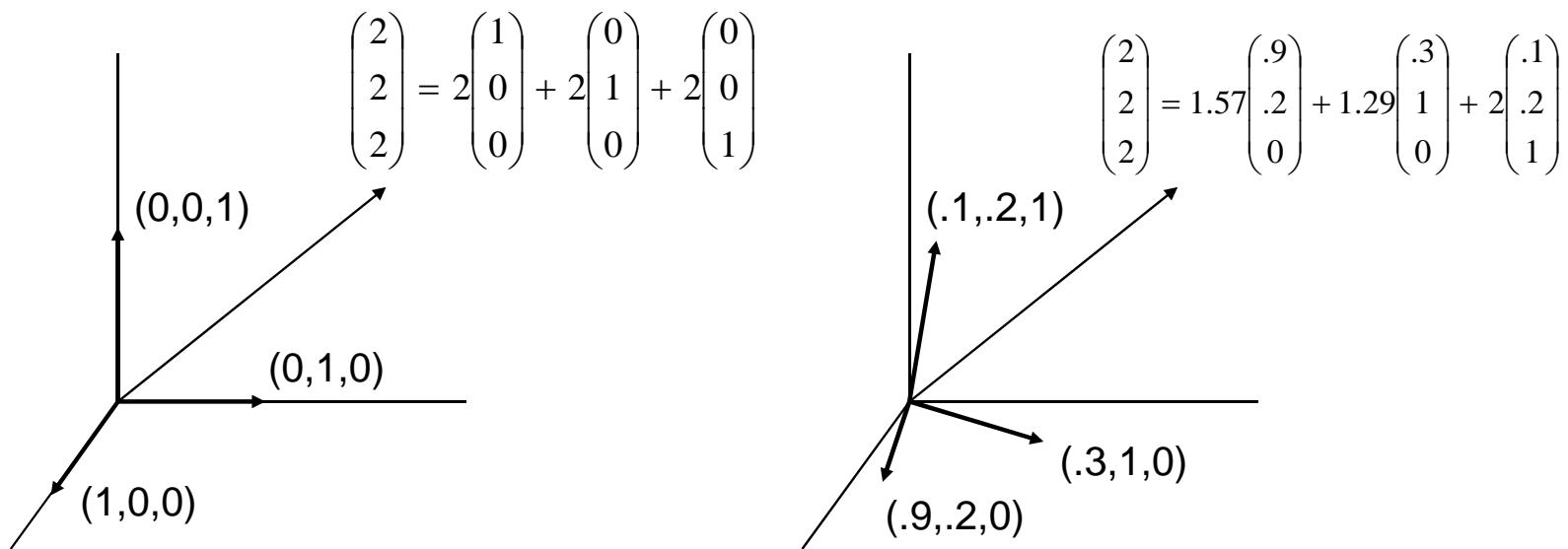
$$\begin{pmatrix} | & | & | \\ v_1 & v_2 & v_3 \\ | & | & | \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 \\ 2 & 3 \\ 1 & 3 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Recall nullspace contained only $(u,v)=(0,0)$.
i.e. the columns are linearly independent.

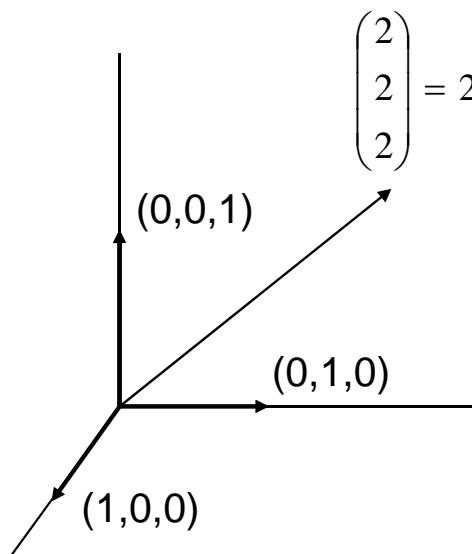
Linear independence and basis

- If all vectors in a vector space may be expressed as linear combinations of v_1, \dots, v_k , then v_1, \dots, v_k *span* the space.

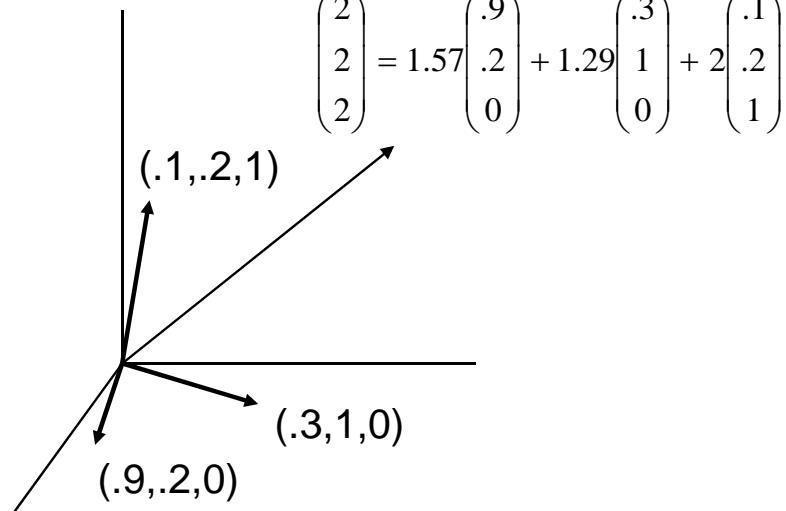


Linear independence and basis

- A *basis* is a set of linearly independent vectors which span the space.
- The *dimension* of a space is the # of “degrees of freedom” of the space; it is the number of vectors in any basis for the space.
- A basis is a maximal set of linearly independent vectors and a minimal set of spanning vectors.



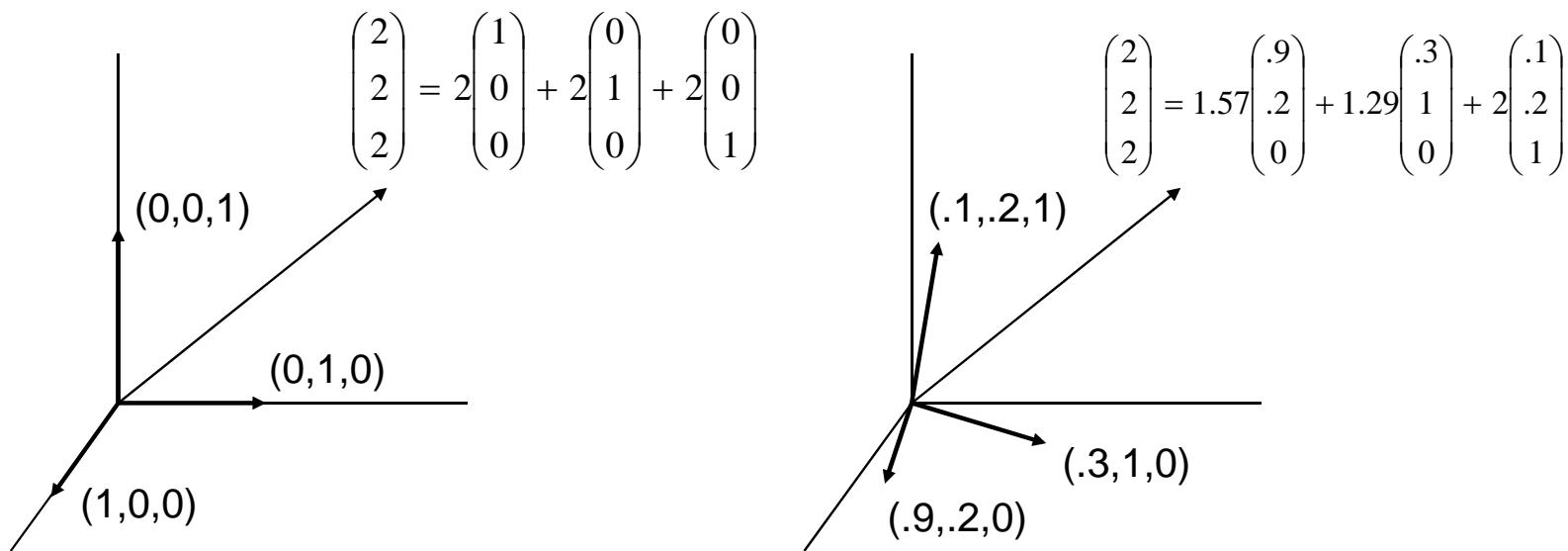
$$\begin{pmatrix} 2 \\ 2 \\ 2 \end{pmatrix} = 2 \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + 2 \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + 2 \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$



$$\begin{pmatrix} 2 \\ 2 \\ 2 \end{pmatrix} = 1.57 \begin{pmatrix} .9 \\ .2 \\ 0 \end{pmatrix} + 1.29 \begin{pmatrix} .3 \\ 1 \\ 0 \end{pmatrix} + 2 \begin{pmatrix} .1 \\ 0 \\ 1 \end{pmatrix}$$

Linear independence and basis

- Two vectors are *orthogonal* if their dot product is 0.
- An *orthogonal basis* consists of orthogonal vectors.
- An *ortho-normal basis* consists of orthogonal vectors of unit length.



About subspaces

- The *rank* of A is the dimension of the column space of A.
- It also equals the *dimension of the row space* of A (the subspace of vectors which may be written as linear combinations of the rows of A).

$$\begin{pmatrix} 1 & 0 \\ 2 & 3 \\ 1 & 3 \end{pmatrix} \quad (1,3) = (2,3) - (1,0)$$

Only 2 linearly independent rows, so rank = 2.

About subspaces

Fundamental Theorem of Linear Algebra:

If A is $(m \times n)$ matrix with rank r,

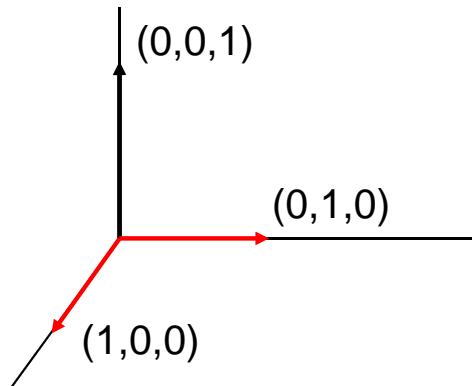
Column space(A) has dimension r

Nullspace(A) has dimension $n-r$ ($=$ nullity of A)

Row space(A) = Column space(A^T) has dimension r

Left nullspace(A) = Nullspace(A^T) has dimension $m - r$

Rank-Nullity Theorem: rank + nullity = n



$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \quad \begin{array}{l} m = 3 \\ n = 2 \\ r = 2 \end{array}$$

Null space, column space

- Null space - it is the orthogonal complement of the row space
- Every vector in this space is a solution to the equation, $Ax = 0$
- Rank – nullity theorem
- Column space
- Compliment of rank-nullity

Non-square matrices

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 3 \end{pmatrix} \quad \begin{matrix} m = 3 \\ n = 2 \\ r = 2 \end{matrix} \quad \text{System } Ax=b \text{ may not have a solution (x has 2 variables but 3 constraints).}$$
$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 3 \end{pmatrix} \quad \begin{matrix} m = 2 \\ n = 3 \\ r = 2 \end{matrix} \quad \text{System } Ax=b \text{ is underdetermined (x has 3 variables and 2 constraints).}$$
$$\begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

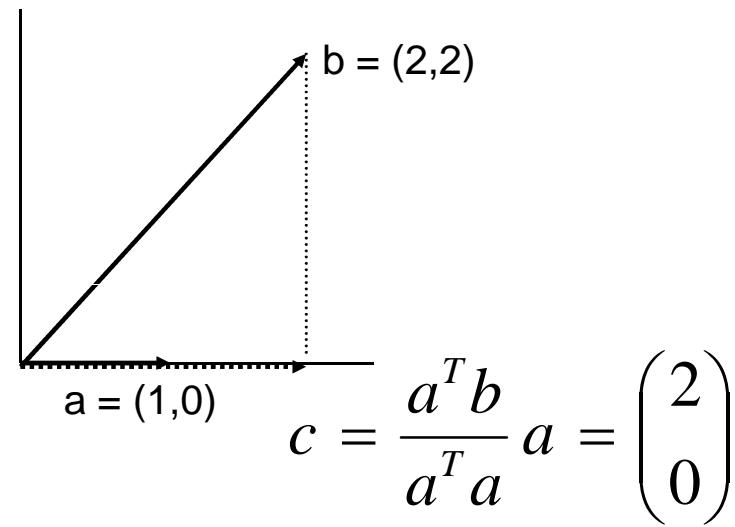
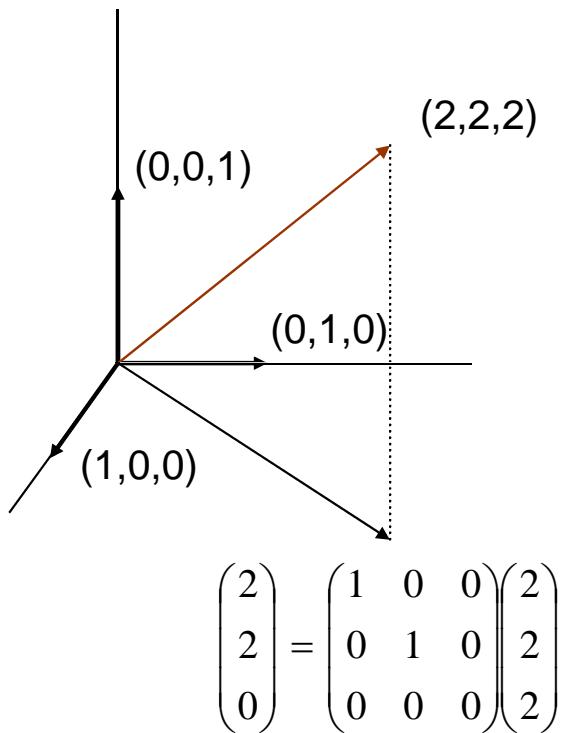
Basis transformations

- Before talking about basis transformations, we need to recall matrix inversion and projections.

Matrix inversion

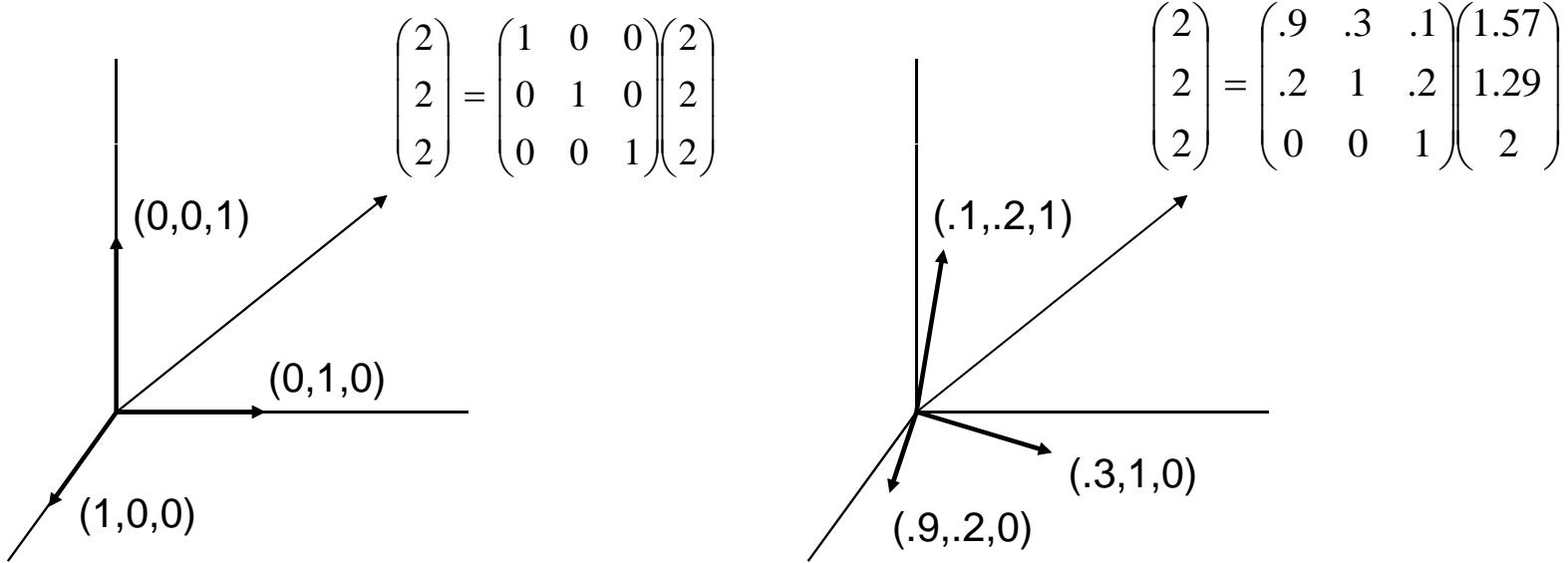
- To solve " $Ax=b$ ", we can write a closed-form solution if we can find a matrix A^{-1}
s.t. $AA^{-1} = A^{-1}A = I$ (identity matrix)
- Then $Ax=b$ iff $x=A^{-1}b$:
$$x = Ix = A^{-1}Ax = A^{-1}b$$
- A is *non-singular* iff A^{-1} exists iff $Ax=b$ has a unique solution.
- Note: If A^{-1}, B^{-1} exist, then $(AB)^{-1} = B^{-1}A^{-1}$,
and $(A^T)^{-1} = (A^{-1})^T$

Projections



Basis transformations

We may write $v=(2,2,2)$ in terms of an alternate basis:



Components of $(1.57, 1.29, 2)$ are projections of v onto new basis vectors, normalized so new v still has same length.

Basis transformations

Given vector v written in standard basis, rewrite as v_Q in terms of basis Q .

If columns of Q are orthonormal, $v_Q = Q^T v$

Otherwise, $v_Q = (Q^T Q)Q^T v$

Special matrices

- Matrix A is *symmetric* if $A = A^T$
- A is *positive definite* if " $x^T A x > 0$ " for all non-zero x (*positive semi-definite* if inequality is not strict)
- Examples:

$$(a \ b \ c) * \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} a \\ b \\ c \end{pmatrix} = a^2 + b^2 + c^2$$

$$(a \ b \ c) * \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} a \\ b \\ c \end{pmatrix} = a^2 - b^2 + c^2$$

Special matrices

Note: Any matrix of form

$A^T A$ is positive semi-definite because,
 $x^T (A^T A) x = (x^T A^T)(Ax) = (Ax)^T (Ax) \geq 0$

Determinants

- If $\det(A) = 0$, then A is **singular**.
- If $\det(A) \neq 0$, then A is **invertible**.

$$\det \begin{pmatrix} a & b \\ c & d \end{pmatrix} = ad - bc$$

Determinants

- m-by-n matrix A is *rank-deficient* if A has rank $r < m (\leq n)$
- Theorem: $\text{rank}(A) < r$ iff
$$\det(A) = 0 \text{ for all } t\text{-by-}t \text{ submatrices,}$$
$$r \leq t \leq m$$

Eigenvalues & eigenvectors

- How can we characterize matrices?
- The solutions to " $Ax = \lambda x$ " in the form of eigenpairs $(\lambda, x) = (\text{eigenvalue}, \text{eigenvector})$ where x is non-zero.
- To solve this, $(A - \lambda I)x = 0$
- λ is an eigenvalue iff $\det(A - \lambda I) = 0$
- The eigenvectors of a matrix are unit vectors that satisfy: $Ax = \lambda x$

Eigenvalues

$$(A - \lambda I) x = 0$$

λ is an eigenvalue iff $\det(A - \lambda I) = 0$

Example:

$$A = \begin{pmatrix} 1 & 4 & 5 \\ 0 & 3/4 & 6 \\ 0 & 0 & 1/2 \end{pmatrix}$$

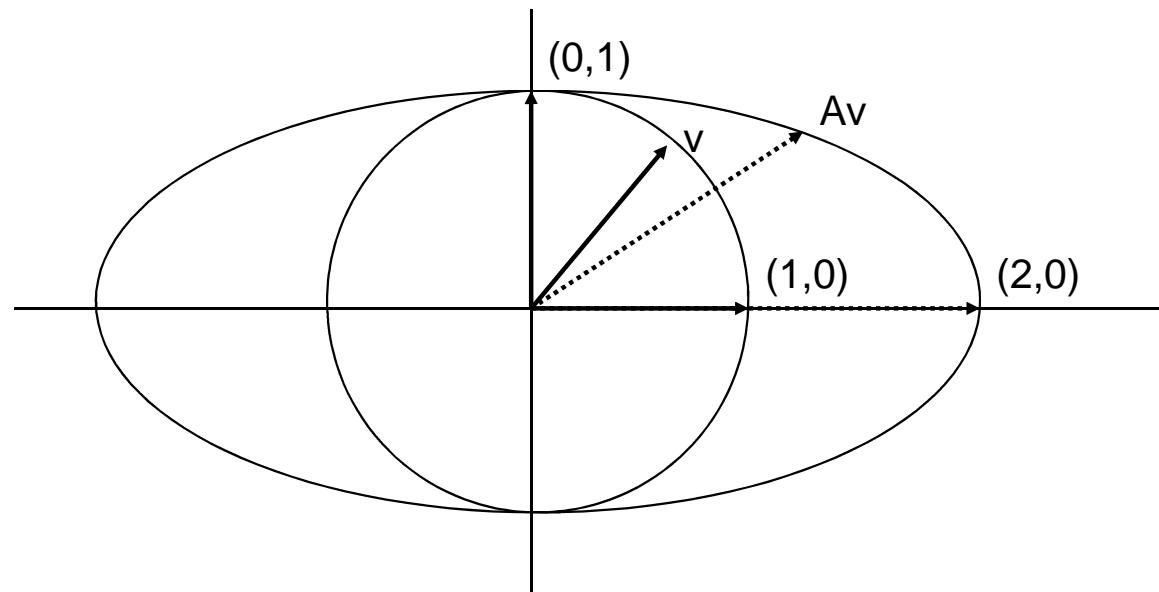
$$\det(A - \lambda I) = \begin{pmatrix} 1 - \lambda & 4 & 5 \\ 0 & 3/4 - \lambda & 6 \\ 0 & 0 & 1/2 - \lambda \end{pmatrix} = (1 - \lambda)(3/4 - \lambda)(1/2 - \lambda)$$

$$\lambda = 1, \lambda = 3/4, \lambda = 1/2$$

Eigenvector example

$$A = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \quad \text{Eigenvalues } \lambda = 2, 1 \text{ with eigenvectors } (1,0), (0,1)$$

Eigenvectors of a linear transformation A are not rotated (but will be scaled by the corresponding eigenvalue) when A is applied.



Solving Ax=b

$$\begin{array}{rcl} x + 2y + z & = & 0 \\ y - z & = & 2 \\ x & & +2z= 1 \\ \hline \end{array}$$

$$\begin{pmatrix} 1 & 2 & 1 & 0 \\ 0 & 1 & -1 & 2 \\ 1 & 0 & 2 & 1 \end{pmatrix}$$

Write system of equations in matrix form.

$$\begin{array}{rcl} x + 2y + z & = & 0 \\ y - z & = & 2 \\ -2y + z & = & 1 \\ \hline \end{array}$$

$$\begin{pmatrix} 1 & 2 & 1 & 0 \\ 0 & 1 & -1 & 2 \\ 0 & -2 & 1 & 1 \end{pmatrix}$$

Subtract first row from last row.

$$\begin{array}{rcl} x + 2y + z & = & 0 \\ y - z & = & 2 \\ -z & = & 5 \\ \hline \end{array}$$

$$\begin{pmatrix} 1 & 2 & 1 & 0 \\ 0 & 1 & -1 & 2 \\ 0 & 0 & -1 & 5 \end{pmatrix}$$

Add 2 copies of second row to last row.

Now solve by back-substitution: $z = -5$, $y = 2-z = -3$, $x = -2y-z = 11$

Solving Ax=b & condition numbers

- Matlab: `linsolve(A,b)`
- How stable is the solution?
- If A or b are changed slightly, how much does it effect x?
- The *condition number* c of A measures this:
$$c = \lambda_{\max} / \lambda_{\min}$$
- Values of c near 1 are good.

LU Decomposition

LUD of a matrix is a method to solve a set of simultaneous linear equations

Which is better:

- Gauss Elimination
- LUD

$$[A] = [L][U] = \begin{bmatrix} 1 & 0 & 0 \\ \ell_{21} & 1 & 0 \\ \ell_{31} & \ell_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

LUD Method

For a non-singular matrix $[A]$, we can use Naive Gauss Elimination forward elimination steps and write A as

$$[A] = [L][U]$$

Where

$[L]$ = lower triangular matrix

$[U]$ = upper triangular matrix

$$L = \begin{bmatrix} 1 & 0 & 0 \\ \ell_{21} & 1 & 0 \\ \ell_{31} & \ell_{32} & 1 \end{bmatrix} \quad [U] = \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

How does LUD work?

If solving a set of linear equations	$[A][X] = [C]$
If $[A] = [L][U]$ then	$[L][U][X] = [C]$
Multiply by	$[L]^{-1}$
Which gives	$[L]^{-1}[L][U][X] = [L]^{-1}[C]$
Remember $[L]^{-1}[L] = [I]$ which leads to	$[I][U][X] = [L]^{-1}[C]$
Now, if $[I][U] = [U]$ then	$[U][X] = [L]^{-1}[C]$
Now, let	$[L]^{-1}[C] = [Z]$
Which ends with	$[L][Z] = [C] \quad (1)$
and	$[U][X] = [Z] \quad (2)$

LUD: How can this be used?

Given $[A][X] = [C]$ to solve

1. Decompose $[A]$ into $[L]$ and $[U]$
2. Solve $[L][Z] = [C]$ for $[Z]$
3. Solve $[U][X] = [Z]$ for $[X]$

Is LUD better than Gaussian Elimination?

To solve $[A][X] = [B]$

Table. Time taken by methods

Gaussian Elimination	LU Decomposition
$T\left(\frac{8n^3}{3} + 12n^2 + \frac{4n}{3}\right)$	$T\left(\frac{8n^3}{3} + 12n^2 + \frac{4n}{3}\right)$

where T = clock cycle time and n = size of the matrix

So both methods are equally efficient.

To find inverse of [A]

Time taken by Gaussian Elimination	Time taken by LU Decomposition
$= n(CT _{FE} + CT _{BS})$	$= CT _{LU} + n \times CT _{FS} + n \times CT _{BS}$
$= T\left(\frac{8n^4}{3} + 12n^3 + \frac{4n^2}{3}\right)$	$= T\left(\frac{32n^3}{3} + 12n^2 + \frac{20n}{3}\right)$

Table 1 Comparing computational times of finding inverse of a matrix using LU decomposition and Gaussian elimination.

n	10	100	1000	10000
$CT _{\text{inverse GE}} / CT _{\text{inverse LU}}$	3.28	25.83	250.8	2501

Method: [A] Decompose to [L] and [U]

$$[A] = [L][U] = \begin{bmatrix} 1 & 0 & 0 \\ \ell_{21} & 1 & 0 \\ \ell_{31} & \ell_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

$[U]$ is the same as the coefficient matrix at the end of the forward elimination step.

$[L]$ is obtained using the *multipliers* that were used in the forward elimination process

LUD Worked Example

Using the Forward Elimination Procedure of Gauss Elimination

$$A = \begin{bmatrix} 25 & 5 & 1 \\ 64 & 8 & 1 \\ 144 & 12 & 1 \end{bmatrix}$$

Finding the $[U]$ matrix

Using the Forward Elimination Procedure of Gauss Elimination

$$A = \begin{bmatrix} 25 & 5 & 1 \\ 64 & 8 & 1 \\ 144 & 12 & 1 \end{bmatrix}$$

$$\text{Step 1: } \frac{64}{25} = 2.56; \quad Row2 - Row1(2.56) = \begin{bmatrix} 25 & 5 & 1 \\ 0 & -4.8 & -1.56 \\ 144 & 12 & 1 \end{bmatrix}$$

$$\frac{144}{25} = 5.76; \quad Row3 - Row1(5.76) = \begin{bmatrix} 25 & 5 & 1 \\ 0 & -4.8 & -1.56 \\ 0 & -16.8 & -4.76 \end{bmatrix}$$

Finding the [U] Matrix

Matrix after Step 1:

$$\begin{bmatrix} 25 & 5 & 1 \\ 0 & -4.8 & -1.56 \\ 0 & -16.8 & -4.76 \end{bmatrix}$$

Step 2: $\frac{-16.8}{-4.8} = 3.5$; $Row3 - Row2(3.5) =$

$$\begin{bmatrix} 25 & 5 & 1 \\ 0 & -4.8 & -1.56 \\ 0 & 0 & 0.7 \end{bmatrix}$$

$$[U] = \begin{bmatrix} 25 & 5 & 1 \\ 0 & -4.8 & -1.56 \\ 0 & 0 & 0.7 \end{bmatrix}$$

Finding the $[L]$ matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ \ell_{21} & 1 & 0 \\ \ell_{31} & \ell_{32} & 1 \end{bmatrix}$$

Using the multipliers used during the Forward Elimination Procedure

From the first step
of forward
elimination

$$\begin{bmatrix} 25 & 5 & 1 \\ 64 & 8 & 1 \\ 144 & 12 & 1 \end{bmatrix} \quad \ell_{21} = \frac{a_{21}}{a_{11}} = \frac{64}{25} = 2.56$$
$$\ell_{31} = \frac{a_{31}}{a_{11}} = \frac{144}{25} = 5.76$$

Finding the [L] Matrix

From the second
step of forward
elimination

$$\begin{bmatrix} 25 & 5 & 1 \\ 0 & -4.8 & -1.56 \\ 0 & -16.8 & -4.76 \end{bmatrix} \ell_{32} = \frac{a_{32}}{a_{22}} = \frac{-16.8}{-4.8} = 3.5$$

$$[L] = \begin{bmatrix} 1 & 0 & 0 \\ 2.56 & 1 & 0 \\ 5.76 & 3.5 & 1 \end{bmatrix}$$

Does $[L][U] = [A]$?

$$[L]*[U] = \begin{bmatrix} 1 & 0 & 0 \\ 2.56 & 1 & 0 \\ 5.76 & 3.5 & 1 \end{bmatrix} * \begin{bmatrix} 25 & 5 & 1 \\ 0 & -4.8 & -1.56 \\ 0 & 0 & 0.7 \end{bmatrix} =$$

Using LU Decomposition

Solve the following set of linear equations using LU Decomposition

$$\begin{bmatrix} 25 & 5 & 1 \\ 64 & 8 & 1 \\ 144 & 12 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 106.8 \\ 177.2 \\ 279.2 \end{bmatrix}$$

find the $[L]$ and $[U]$ matrices

$$[A] = [L][U] = \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 25 & 5 & 1 \\ 2.56 & 1 & 0 & 0 & -4.8 & -1.56 \\ 5.76 & 3.5 & 1 & 0 & 0 & 0.7 \end{array} \right]$$

Example

Set $[L][Z] = [C]$

$$\begin{bmatrix} 1 & 0 & 0 \\ 2.56 & 1 & 0 \\ 5.76 & 3.5 & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 106.8 \\ 177.2 \\ 279.2 \end{bmatrix}$$

Solve for $[Z]$

$$z_1 = 10$$

$$2.56z_1 + z_2 = 177.2$$

$$5.76z_1 + 3.5z_2 + z_3 = 279.2$$

Example

Complete the forward substitution to solve for [Z]

$$z_1 = 106.8$$

$$\begin{aligned} z_2 &= 177.2 - 2.56z_1 \\ &= 177.2 - 2.56(106.8) \\ &= -96.2 \end{aligned}$$

$$\begin{aligned} z_3 &= 279.2 - 5.76z_1 - 3.5z_2 \\ &= 279.2 - 5.76(106.8) - 3.5(-96.21) \\ &= 0.735 \end{aligned}$$

$$[Z] = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 106.8 \\ -96.21 \\ 0.735 \end{bmatrix}$$

Example

Set $[U][X] = [Z]$

$$\begin{bmatrix} 25 & 5 & 1 \\ 0 & -4.8 & -1.56 \\ 0 & 0 & 0.7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 106.8 \\ -96.21 \\ 0.735 \end{bmatrix}$$

Solve for $[X]$

The 3 equations become

$$25a_1 + 5a_2 + a_3 = 106.8$$

$$-4.8a_2 - 1.56a_3 = -96.21$$

$$0.7a_3 = 0.735$$

Now use backward substitution

From the 3rd equation

$$0.7a_3 = 0.735$$

$$a_3 = \frac{0.735}{0.7}$$

$$a_3 = 1.050$$

Substituting in a_3 and using the second equation

$$-4.8a_2 - 1.56a_3 = -96.21$$

$$a_2 = \frac{-96.21 + 1.56a_3}{-4.8}$$

$$a_2 = \frac{-96.21 + 1.56(1.050)}{-4.8}$$

$$a_2 = 19.70$$

To get the solution

Substituting in a_3 and a_2 using
the first equation

$$25a_1 + 5a_2 + a_3 = 106.8$$

$$\begin{aligned} a_1 &= \frac{106.8 - 5a_2 - a_3}{25} \\ &= \frac{106.8 - 5(19.70) - 1.050}{25} \\ &= 0.2900 \end{aligned}$$

Hence the Solution Vector is:

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 0.2900 \\ 19.70 \\ 1.050 \end{bmatrix}$$

Matrix Inverse

$$\begin{bmatrix} \cos 90^\circ & \sin 90^\circ \\ -\sin 90^\circ & \cos 90^\circ \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Finding the inverse of a square matrix

The inverse [B] of a square matrix [A] is defined as

$$[A][B] = [I] = [B][A]$$

Finding the inverse of a square matrix

How can LU Decomposition be used to find the inverse?

Assume the first column of $[B]$ to be $[b_{11} \ b_{21} \ \dots \ b_{n1}]^T$

Using this and the definition of matrix multiplication

First column of $[B]$

$$[A] \begin{bmatrix} b_{11} \\ b_{21} \\ \vdots \\ b_{n1} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Second column of $[B]$ is

$$[A] \begin{bmatrix} b_{12} \\ b_{22} \\ \vdots \\ b_{n2} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

The remaining columns in $[B]$ can be found similarly

Example: Inverse of a Matrix

Find the inverse of a square matrix $[A]$

$$[A] = \begin{bmatrix} 25 & 5 & 1 \\ 64 & 8 & 1 \\ 144 & 12 & 1 \end{bmatrix}$$

Using the decomposition procedure, the $[L]$ and $[U]$ matrices are found to be

$$[A] = [L][U] = \begin{bmatrix} 1 & 0 & 0 \\ 2.56 & 1 & 0 \\ 5.76 & 3.5 & 1 \end{bmatrix} \begin{bmatrix} 25 & 5 & 1 \\ 0 & -4.8 & -1.56 \\ 0 & 0 & 0.7 \end{bmatrix}$$

Example: Inverse of a Matrix

Solving for the each column of $[B]$ requires two steps

- Solve $[L] [Z] = [C]$ for $[Z]$
- Solve $[U] [X] = [Z]$ for $[X]$

$$\text{Step 1: } [L][Z] = [C] \rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 2.56 & 1 & 0 \\ 5.76 & 3.5 & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

This generates the equations:

$$z_1 = 1$$

$$2.56z_1 + z_2 = 0$$

$$5.76z_1 + 3.5z_2 + z_3 = 0$$

Example: Inverse of a Matrix

Solving for $[Z]$

$$z_1 = 1$$

$$\begin{aligned} z_2 &= 0 - 2.56z_1 \\ &= 0 - 2.56(1) \\ &= -2.56 \end{aligned}$$

$$\begin{aligned} z_3 &= 0 - 5.76z_1 - 3.5z_2 \\ &= 0 - 5.76(1) - 3.5(-2.56) \\ &= 3.2 \end{aligned}$$

$$[Z] = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 1 \\ -2.56 \\ 3.2 \end{bmatrix}$$

Example: Inverse of a Matrix

Solving $[U][X] = [Z]$ for $[X]$

$$\begin{bmatrix} 25 & 5 & 1 \\ 0 & -4.8 & -1.56 \\ 0 & 0 & 0.7 \end{bmatrix} \begin{bmatrix} b_{11} \\ b_{21} \\ b_{31} \end{bmatrix} = \begin{bmatrix} 1 \\ -2.56 \\ 3.2 \end{bmatrix}$$

$$25b_{11} + 5b_{21} + b_{31} = 1$$

$$-4.8b_{21} - 1.56b_{31} = -2.56$$

$$0.7b_{31} = 3.2$$

Example: Inverse of a Matrix

Using Backward Substitution

$$b_{31} = \frac{3.2}{0.7} = 4.571$$

$$\begin{aligned} b_{21} &= \frac{-2.56 + 1.560b_{31}}{-4.8} \\ &= \frac{-2.56 + 1.560(4.571)}{-4.8} = -0.9524 \end{aligned}$$

$$\begin{aligned} b_{11} &= \frac{1 - 5b_{21} - b_{31}}{25} \\ &= \frac{1 - 5(-0.9524) - 4.571}{25} = 0.04762 \end{aligned}$$

So the first column of
the inverse of $[A]$ is:

$$\begin{bmatrix} b_{11} \\ b_{21} \\ b_{31} \end{bmatrix} = \begin{bmatrix} 0.04762 \\ -0.9524 \\ 4.571 \end{bmatrix}$$

Example: Inverse of a Matrix

Repeating for the second and third columns of the inverse

Second Column

$$\begin{bmatrix} 25 & 5 & 1 \\ 64 & 8 & 1 \\ 144 & 12 & 1 \end{bmatrix} \begin{bmatrix} b_{12} \\ b_{22} \\ b_{32} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} b_{12} \\ b_{22} \\ b_{32} \end{bmatrix} = \begin{bmatrix} -0.08333 \\ 1.417 \\ -5.000 \end{bmatrix}$$

Third Column

$$\begin{bmatrix} 25 & 5 & 1 \\ 64 & 8 & 1 \\ 144 & 12 & 1 \end{bmatrix} \begin{bmatrix} b_{13} \\ b_{23} \\ b_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} b_{13} \\ b_{23} \\ b_{33} \end{bmatrix} = \begin{bmatrix} 0.03571 \\ -0.4643 \\ 1.429 \end{bmatrix}$$

Example: Inverse of a Matrix

The inverse of $[A]$ is

$$[A]^{-1} = \begin{bmatrix} 0.04762 & -0.08333 & 0.03571 \\ -0.9524 & 1.417 & -0.4643 \\ 4.571 & -5.000 & 1.429 \end{bmatrix}$$

To check your work do the following operation

$$[A][A]^{-1} = [I] = [A]^{-1}[A]$$

LP: Linear Programming

LP: Linear Programming

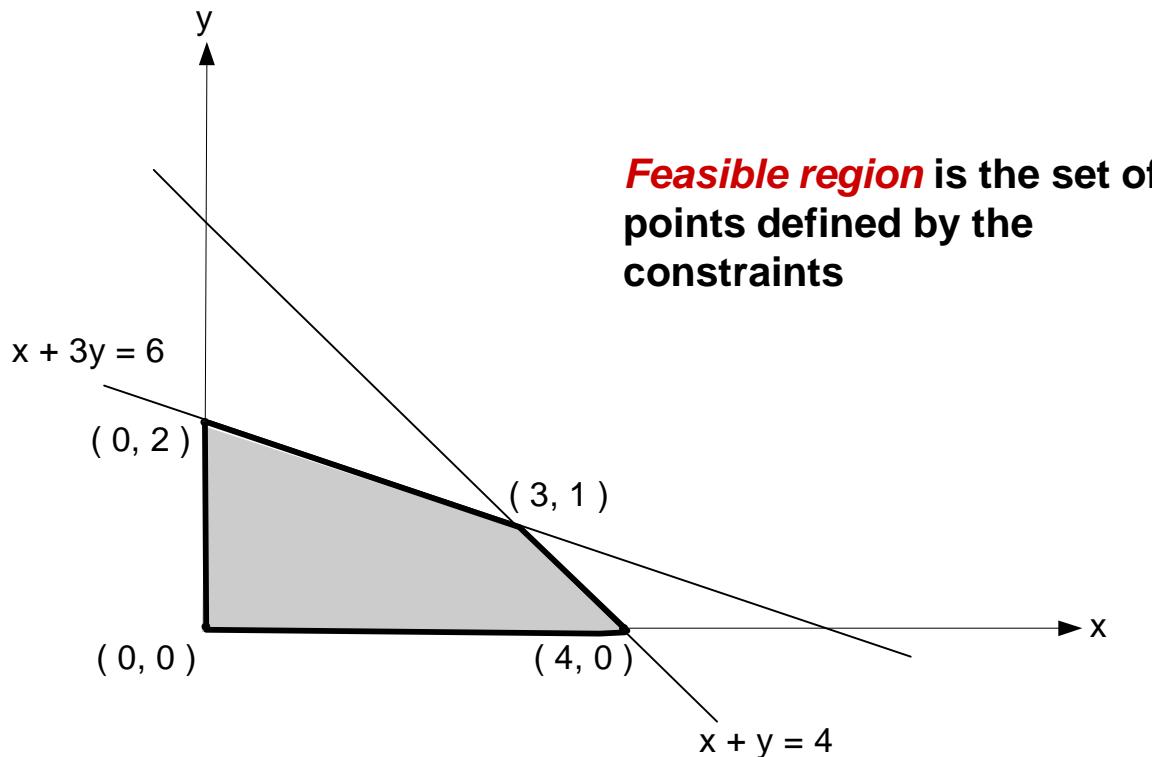
- Large proportion of all scientific and financial computations
- What is LP: Finding optimal solution under linear constraints.
- LP *is used to allocate resources,*
 - *plan production,*
 - *schedule workers,*
 - *plan investment portfolios and*
 - *formulate marketing (and military) strategies.*
- The versatility and economic impact of LP in today's industrial world is truly astounding.

Important Examples

- Simplex method
- Ford-Fulkerson algorithm for maximum flow problem
- Maximum matching of graph vertices
- Gale-Shapley algorithm for the stable marriage problem

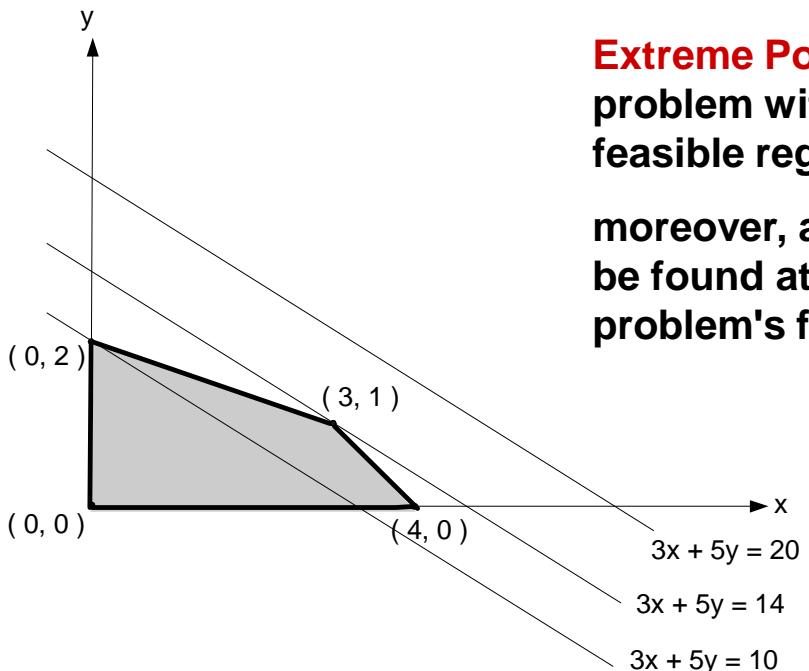
maximize	$3x + 5y$
subject to	$x + y \leq 4$
	$x + 3y \leq 6$
$x \geq 0, y \geq 0$	

Example



maximize	$3x + 5y$
subject to	$x + y \leq 4$
	$x + 3y \leq 6$
$x \geq 0, y \geq 0$	

Geometric solution



Extreme Point Theorem Any LP problem with a nonempty bounded feasible region has an optimal solution; moreover, an optimal solution can always be found at an **extreme point** of the problem's feasible region.

Possible outcomes in solving an LP problem

- has a **finite optimal solution**, which may not be unique.
- ***unbounded***: the objective function of maximization (minimization) LP problem is unbounded from above (below) on its feasible region.
- ***infeasible***: there are no points satisfying all the constraints, i.e. the constraints are contradictory.

Graphing 2-Dimensional LPs

Example 1:

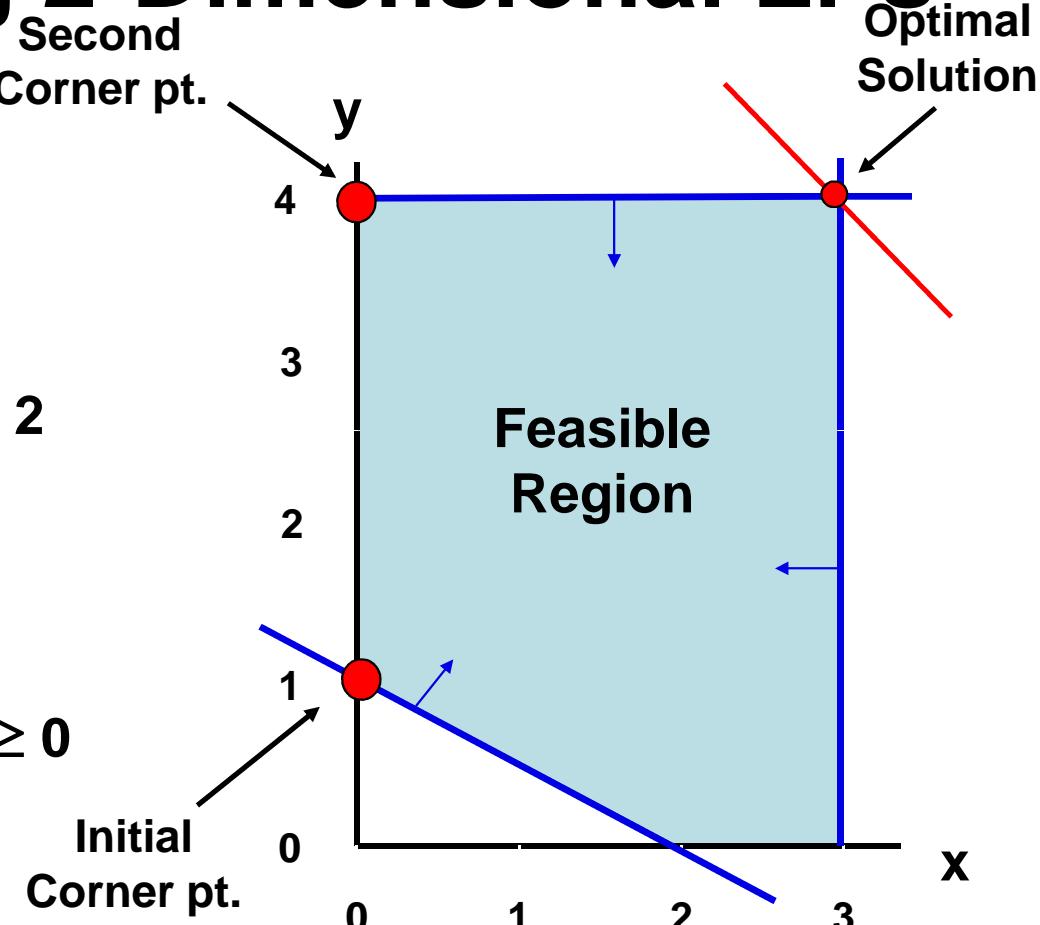
Maximize $x + y$

Subject to: $x + 2y \geq 2$

$x \leq 3$

$y \leq 4$

$x \geq 0 \quad y \geq 0$



Graphing 2-D LP – Unique solution

Example 1:

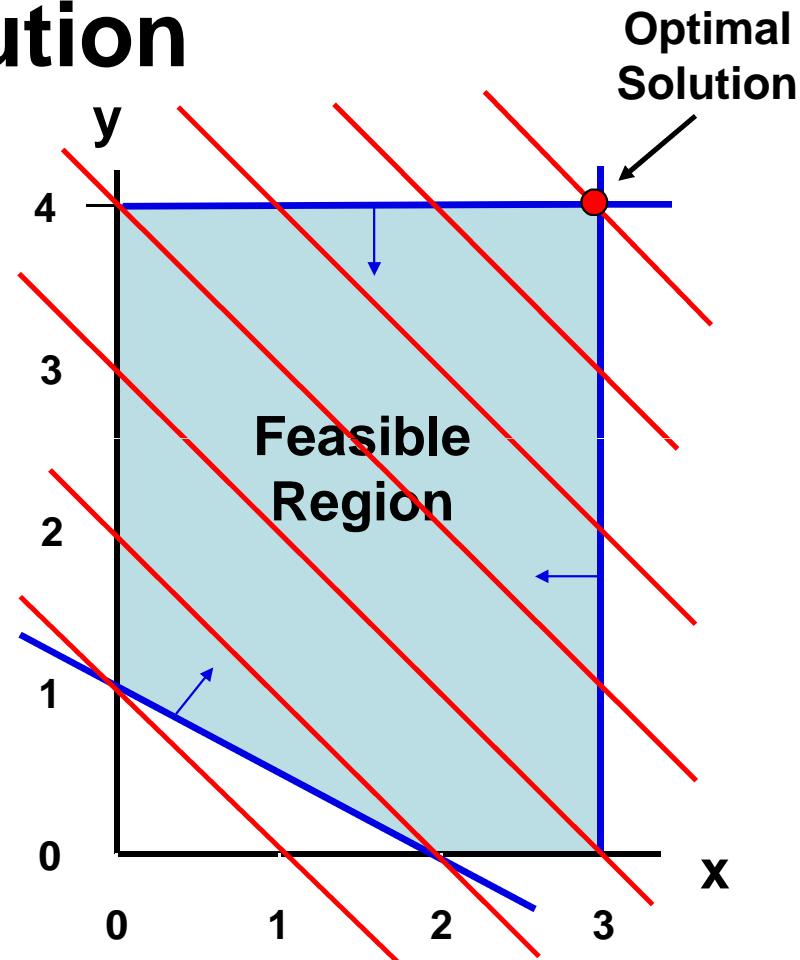
Maximize $x + y$

Subject to: $x + 2y \geq 2$

$x \leq 3$

$y \leq 4$

$x \geq 0 \quad y \geq 0$



Graphing LP multiple solutions

Example 2:

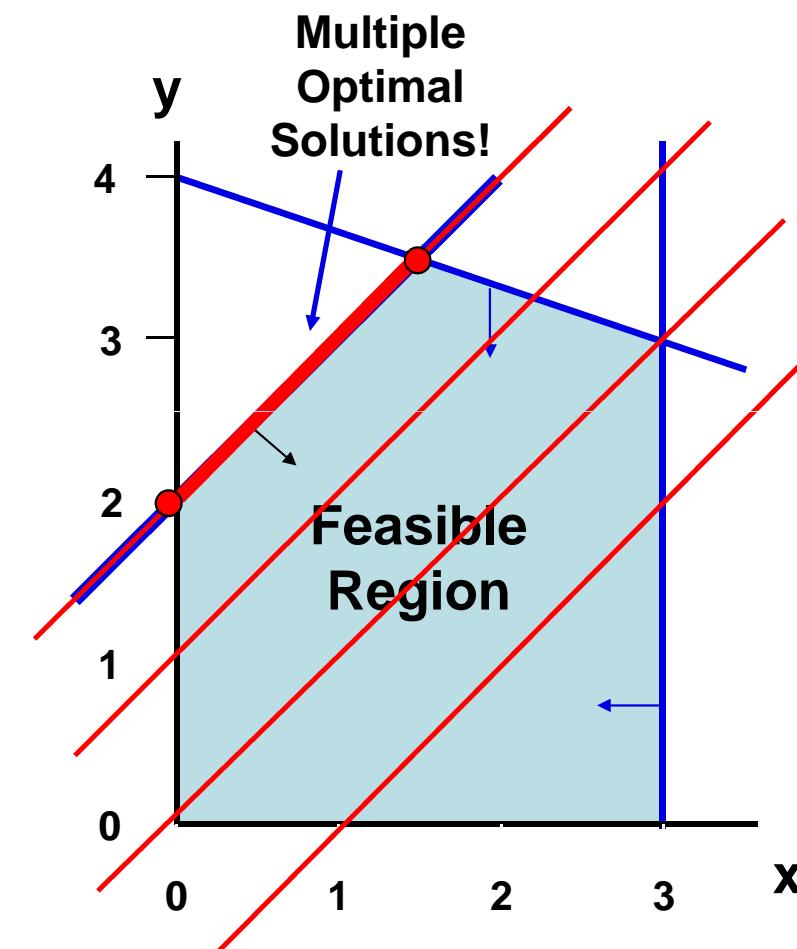
Minimize ** $x - y$

Subject to: $\frac{1}{3}x + y \leq 4$

$-2x + 2y \leq 4$

$x \leq 3$

$x \geq 0 \quad y \geq 0$



Graphing 2-D LP

Example 3:

Minimize $x + 1/3 y$

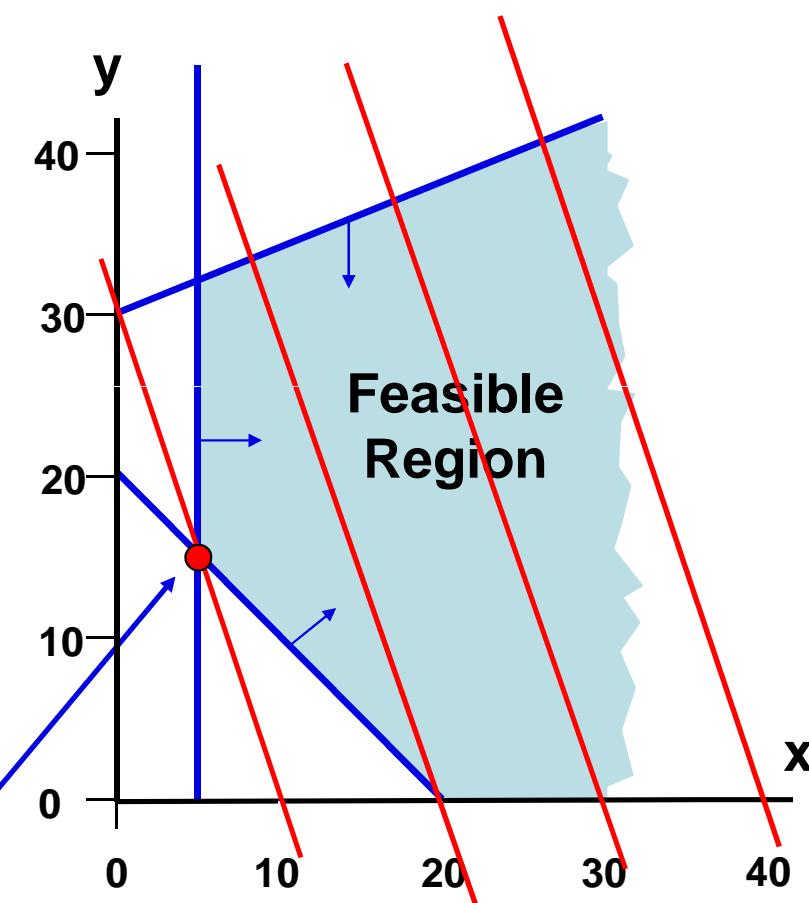
Subject to: $x + y \geq 20$

$-2x + 5y \leq 150$

$x \geq 5$

$x \geq 0 \quad y \geq 0$

Optimal Solution



Graphing 2-D LP, Unbounded

Example 4:

Maximize $x + 1/3 y$

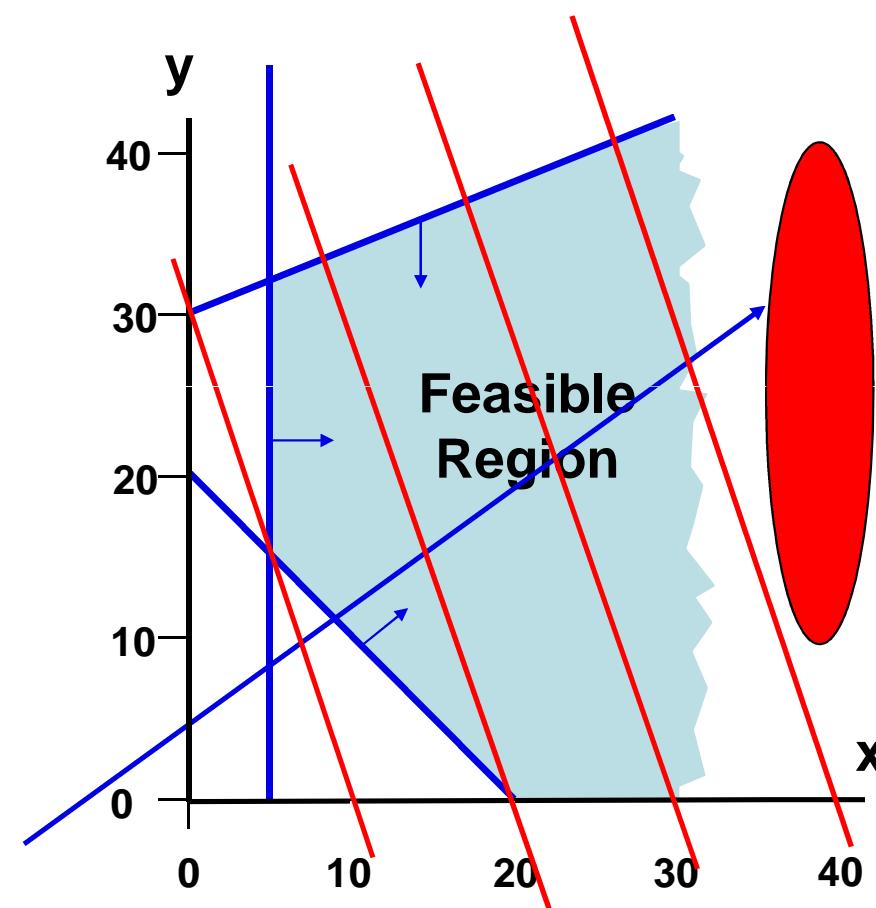
Subject to: $x + y \geq 20$

$$-2x + 5y \leq 150$$

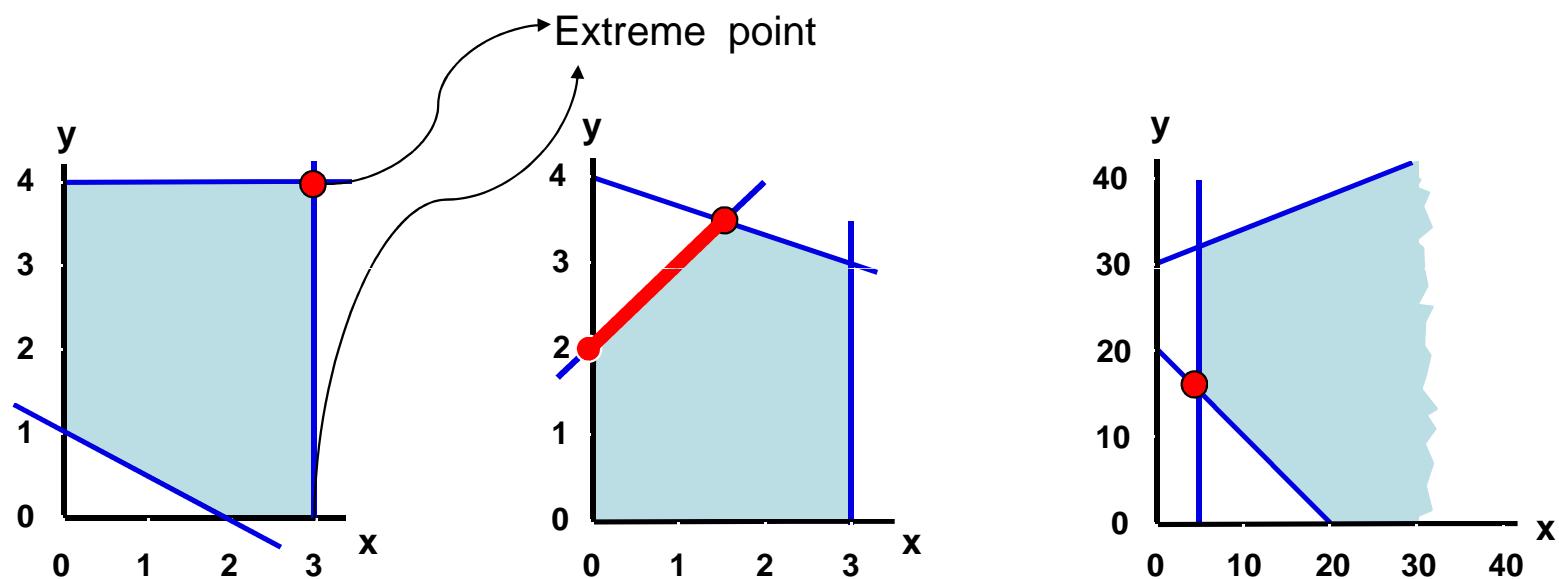
$$x \geq 5$$

$$x \geq 0 \quad y \geq 0$$

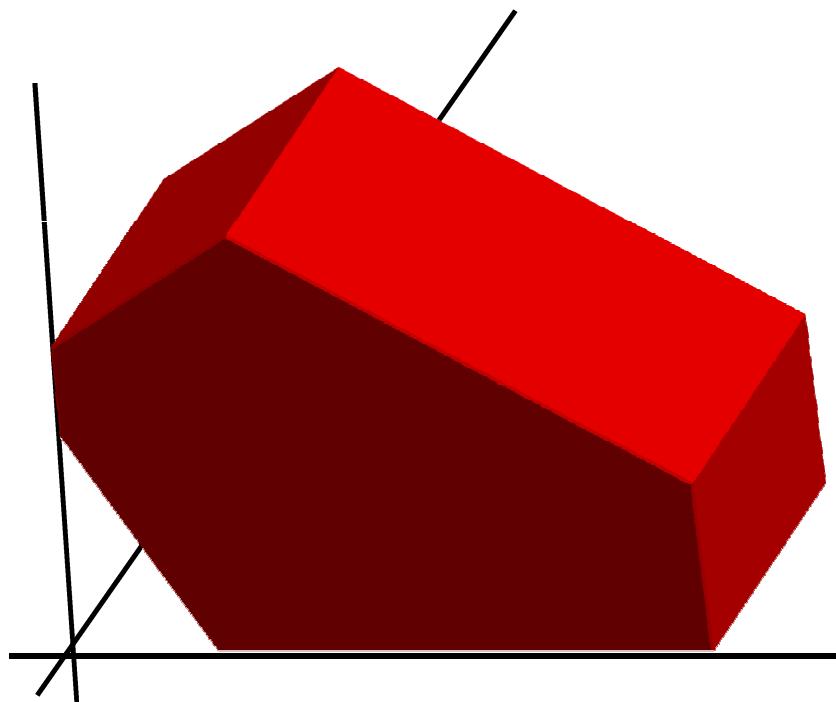
**Optimal Solutions
unbounded**



Optimal lies on a corner



Extend to Higher Dimensions



To solve an LP

- The constraints of an LP give rise to a polytope.
- If we can determine all the corner points of the polytope, then we can calculate the objective value at these points and take the best one as our optimal solution.
- The *Simplex Method* intelligently moves from corner to corner until it can prove that it has found the optimal solution.

Linear Programs in higher dimensions

maximize

$$z = -4x_1 + x_2 - x_3$$

subject to

$$c1: -7x_1 + 5x_2 + x_3 \leq 8$$

$$c2: -2x_1 + 4x_2 + 2x_3 \leq 10$$

$$x_1, x_2, x_3 \geq 0$$

Linear Programming

- *Linear programming (LP) problem* is to optimize a linear function of several variables subject to linear constraints:

maximize (or minimize) $c_1 x_1 + \dots + c_n x_n$
subject to $a_{i1}x_1 + \dots + a_{in}x_n \leq (\text{or } \geq \text{ or } =) b_i,$
 $i = 1, \dots, m, x_1 \geq 0, \dots, x_n \geq 0$

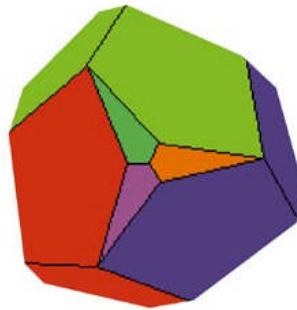
The function $z = c_1 x_1 + \dots + c_n x_n$ is called the

objective function;

constraints $x_1 \geq 0, \dots, x_n \geq 0$ are called

non-negativity constraints

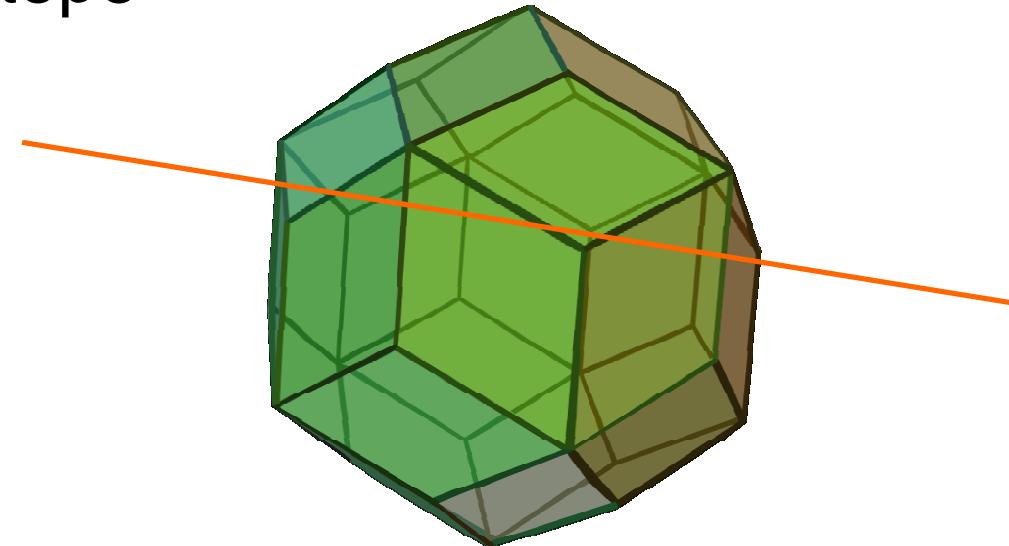
LP Geometry



- n-dimensional polytope
- convex: mid-point of any two feasible solutions is also feasible.

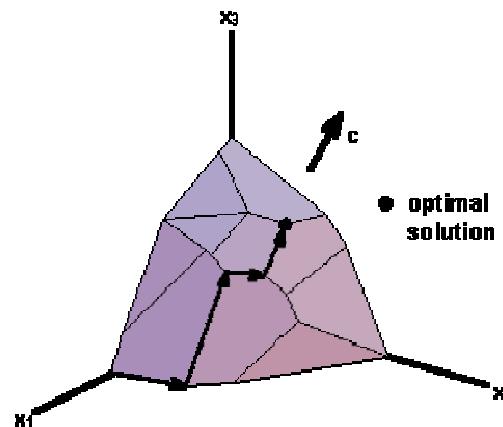
LP Geometry: Plane in the Polytope

- Feasible region: convex polytope
(given by inequalities)
- Objective function: A plane intersecting this polytope

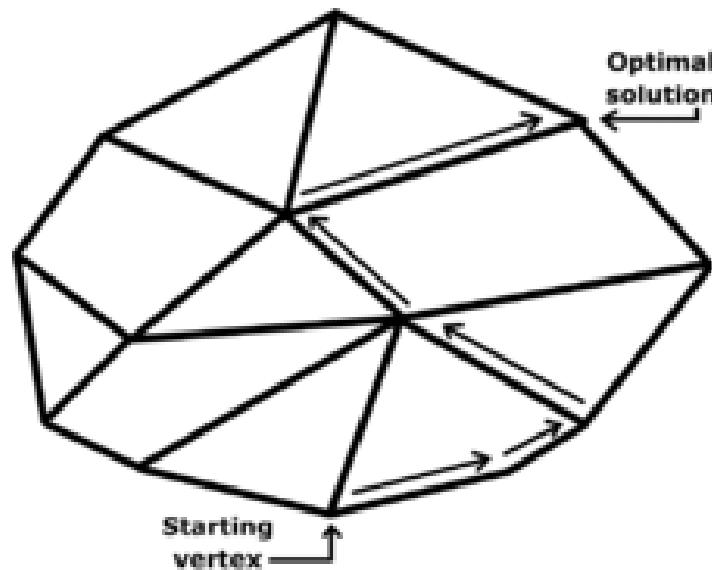


LP Geometry

- Extreme point theorem: If there exists an optimal solution to an LP Problem, then there exists one extreme point where the optimum is achieved.

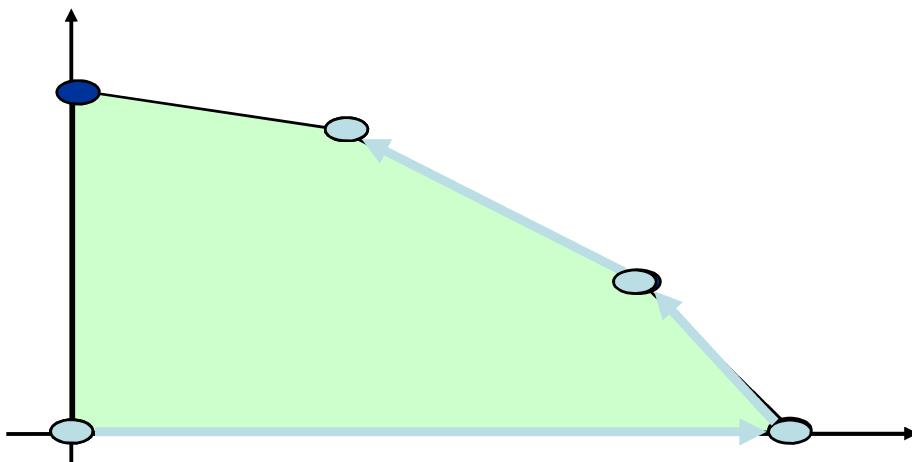


LP Algorithms



- Simplex by Dantzig 1947

- Practical solution method that moves from one extreme point to a neighboring extreme point.
- Exponential in worse case complexity
- Number of vertices = $C(n,m)=n!/(m!(n-m)!)$
 - n = number of variables
 - m = number of constraints
- Very fast in practice, steps usually $O(m)$, n effect cost per iteration.
- TWO PHASE simplex:
 1. Phase 1. Finding an initial basic feasible solution may pose a problem
 2. Phase 2. Move from vertex to vertex.
- ❖ Theoretical possibility of cycling



Worst case simplex takes exponential time to complete.

- Imagine N^2 path in 3-Dimension.



Standard form of LP problem

- Must be a **maximization** problem
- All constraints (except the non-negativity constraints) must be in the form of **linear equations**
- All the variables must be required to be **nonnegative**

Thus, the general linear programming problem in standard form with m constraints and n unknowns ($n \geq m$) is

maximize $c_1 x_1 + \dots + c_n x_n$

subject to $a_{i1}x_1 + \dots + a_{in}x_n = b_i, , i=1, \dots, m,$

$x_1 \geq 0, \dots, x_n \geq 0$

Every LP problem can be represented in such form

In Matrix terms

$$\text{Max } c^T x$$

subject to $Ax \leq b$

$$A_{n \times d}, c_{d \times 1}, x_{d \times 1}$$

LP Example: Diet problem

The Diet Problem

- Dietician preparing a diet consisting of two foods: Egg and Dal

	Cost	Protein	Fat	Carbohydrate	Buy
Egg	0.60 Rs	20g	12g	30g	x1
Dal	0.40 Rs	30g	6g	15g	x2
daily required		60g	24g	30g	
	Minimize				

Looking for *minimum cost diet*

Diet LP

	Cost	Protein	Fat	Carbohydrate	Buy
Egg	0.60 Rs	20g	12g	30g	x1
Dal	0.40 Rs	30g	6g	15g	x2
daily required		60g	24g	30g	
	Minimize				



$$\min C = 0.60x_1 + 0.40x_2$$

$$s.t. \quad 20x_1 + 30x_2 \geq 60$$

$$12x_1 + 6x_2 \geq 24$$

$$30x_1 + 15x_2 \geq 30$$

Minimize cost of egg + dal

60gm of protein

24gm of fat

30gm of carb

where

$$x_1, x_2 \geq 0$$

Dual of Diet LP

$$\min C = 0.60x_1 + 0.40x_2$$

$$s.t. \quad 20x_1 + 30x_2 \geq 60$$

$$12x_1 + 6x_2 \geq 24$$

$$30x_1 + 15x_2 \geq 30$$

where

$$x_1, x_2 \geq 0$$



$$\max z = 60 u_1 + 24 u_2 + 30 u_3$$

s.t.

$$20 u_1 + 12 u_2 + 30 u_3 \leq 0.6$$

$$30 u_1 + 6 u_2 + 15 u_3 \leq 0.4$$

$$u_1, u_2, u_3 \geq 0$$



	Cost	Protein	Fat	Carbohydrate	Buy
Egg	0.60 Rs	20g	12g	30g	x1
Dal	0.40 Rs	30g	6g	15g	x2
daily required		60g	24g	30g	
	Minimize				

Another Diet and its dual example

	Chocolate	Sugar	Cream Cheese	Cost
Brownie	3	2	2	50
Cheesecake	0	4	5	80
Requirements	6	10	8	

$$\begin{array}{ll}
 \min_{x_1, x_2} & 50x_1 + 80x_2 \\
 \text{subject to} & 3x_1 \geq 6, \\
 & 2x_1 + 4x_2 \geq 10, \\
 & 2x_1 + 5x_2 \geq 8, \\
 & x_1, x_2 \geq 0,
 \end{array}
 \quad
 \begin{array}{ll}
 \max_{u_1, u_2, u_3} & 6u_1 + 10u_2 + 8u_3 \\
 \text{subject to} & 3u_1 + 2u_2 + 2u_3 \leq 50, \\
 & 4u_2 + 5u_3 \leq 80, \\
 & u_1, u_2, u_3 \geq 0.
 \end{array}$$

Buyer's perspective: "How can I buy cheapest brownie and cheesecake with so much chocolate, sugar, and cream cheese?"

Wholesaler's perspective: "How can I set the prices per ounce of chocolate, sugar, and cream cheese so that the baker will buy from me, and so that I will maximize my revenue?"

Exercise: Linear Programming LP

Q. Write Linear programming equations to describe the following optimization problem:

A furniture manufacturer makes two types of furniture:
chairs and sofas.

The production of the sofas and chairs requires three operations:
carpentry, finishing, and upholstery.

Manufacturing a chair requires 3 hours of carpentry, 9 hours of
finishing, and 2 hours of upholstery.

Manufacturing a sofa requires 2 hours of carpentry, 4 hours of finishing,
and 10 hours of upholstery.

The factory has allocated at most 66 labor hours for carpentry, 180
labor hours for finishing, and 200 labor hours for upholstery.

The profit per chair is Rs 900 and the profit per sofa is Rs 750.

What are the variables?

What are we optimizing?

What are the constraints?

LP Farm Example



LP Farm Example

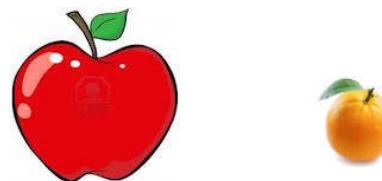
- Farmer produces (**Apples,Oranges**)=(**x,y**)
- Each crop needs { land, fertilizer, time }.



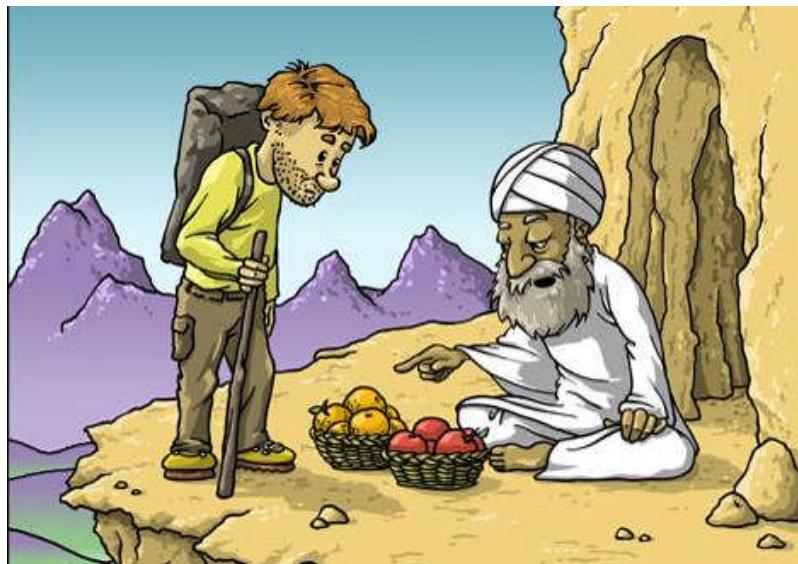
LP example to help a farmer

We have following linear constraints:

- 6 acres of land: $3x + y \leq 6$
- 6 tons of fertilizer: $2x + 3y \leq 6$
- 8 hour work day: $x + 5y \leq 8$
- Apples sell for twice as much as oranges
- We want to maximize profit: $z = 2x + y$
- Production is positive: $x \geq 0, y \geq 0$

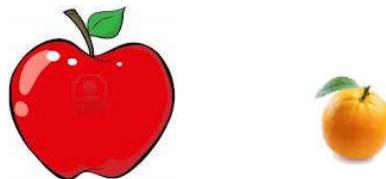


Maximize profit



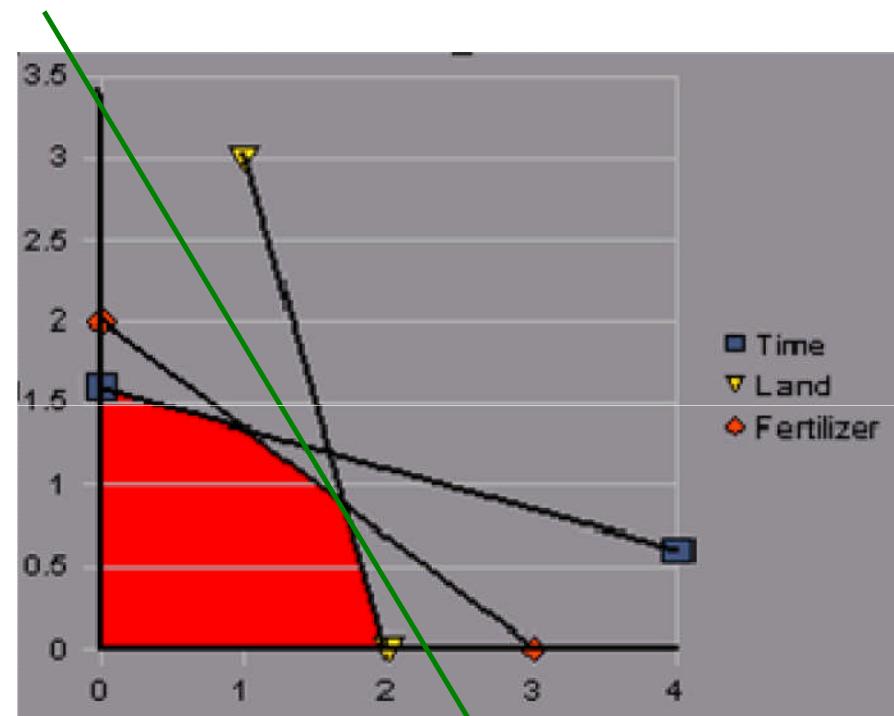
Apples sell for twice as much as oranges

We want to maximize profit (z): $2x + y = z$



Traditional Method

$$\begin{aligned}x &= 1.71 \\y &= .86 \\z &= 4.29\end{aligned}$$



- Graph the inequalities
- Look at the line we're trying to maximize.

Convert to linear program

- $-Z + 2x + y = 0$:Objective Equation
- $s_1 + x + 5y = 8$: C1
- $s_2 + 2x + 3y = 6$: C2
- $s_3 + 3x + y = 6$: C3
- Initial feasible solution
- $\{s1,s2,s3\}$ = slack variables
- Z is the objective function to optimize
- {C1,C2,C3} are the linear constraints.

More definitions

- Non-basic: {x, y}
- Basic variables: { s_1 , s_2 , s_3 , $\textcolor{green}{z}$ }
- First solution: non-basic variables $(x,y)=0$
means $\textcolor{green}{z}$ is also zero in $-z + 2x + y = 0$
- Start with zero profit $\textcolor{green}{z}$, and improve on $\textcolor{green}{z}$.

Next step...

Select a non-basic variable with largest coeff,
ie. x, increasing this will give most **profit** (x
gives twice the profit of y):

$$-\text{z} + 2x + 1y = 0$$

Select the basic variable with the smallest
positive ratio to exit (s3 go from 6 to 0):

$$1x + 5y + s1 = 8 \quad \text{ratio=8}$$

$$2x + 3y + s2 = 6 \quad \text{ratio=3}$$

$$3x + y + s3 = 6 \quad \text{ratio=2}$$

Algorithm

Step 1. Add slack variables, convert \leq to $=$.

Step 2. Write the tableau (combined matrix)

Step 3. Find pivot element

Step 3a. Pivot column is the most negative coefficient of the objective row. If there is column with negative entry, you have the solution, exit.

Step 3b. Compute ratios of last column divided by pivot column. Pick the row with the minimum positive ratio as the pivot row.

Step 4. Make the pivot element 1 (by dividing the row by the pivot element).

Step 5. Make remaining elements in the pivot column '0', using only row operations.

Reading the tableau: Unit columns are basic variables, whose values are given by the last column, in row with '1'. Other variables are '0' (non-basic).

Goto step 3.

Simplex using Maple package

Put the equations into a tableau (matrices):

```
x  y  s1  s2  s3    // x,y = 0 (non basic)
[1  5  1  0  0]    // s1 = 8
A := [2  3  0  1  0]    // s2 = 6
[3  1  0  0  1]    // s3 = 6
```

```
[8]
b := [6]
[6]
```

```
c := [-2 -1 0 0 0]    // optimization function, note signs
change
z := 0                  // optimal value
```

Use row 3, column 1 as pivot

- > ratios(1); // enter x into basic, find who will leave.
- row 1: Upper bound = 8.0000
- row 2: Upper bound = 3.0000
- row 3: Upper bound = 2.0000 // smallest + ratio, so s3 will leave
- > pivot(3,1); // enter x into basic, leave s3.
- | x | y | s1 | s2 | s3 | | b |
|------|------|------|------|------|--|----------|
| 0.00 | 4.66 | 1.00 | 0.00 | -.33 | | 6.00 |
| 0.00 | 2.33 | 0.00 | 1.00 | -.66 | | 2.00 |
| 1.00 | .33 | 0.00 | 0.00 | .33 | | 2.00 |
| 0.00 | -.33 | 0.00 | 0.00 | .66 | | 4.00 = z |
- The optimal z = 4, at (x,y)=(2,0), can be further improved
- Because of negative value in last row: -0.33, y can enter.

Use row 2, column 2 as pivot

- > ratios(2); // enter y, find who will leave
- row 1: Upper bound = 1.2857
- row 2: Upper bound = .8571 // smallest +ve, so s2 will leave
- row 3: Upper bound = 6.0000
- > pivot(2,2); // enter y into basic, leave s2.
- | x | y | s1 | s2 | s3 | b |
|------|------|------|-------|------|----------|
| 0.00 | 0.00 | 1.00 | -2.00 | 1.00 | 2.00 |
| 0.00 | 1.00 | 0.00 | .42 | -.28 | .85 |
| 1.00 | 0.00 | 0.00 | -.14 | .42 | 1.71 |
| 0.00 | 0.00 | 0.00 | .14 | .57 | 4.28 = z |
- No more negative values in last row
- So $z = 4.28$ is optimal, at $(x,y)=(1,0.85)$



Homework

- Download glpk (gnu linear programming package) and solve the above problem.
- Use Microsoft Excel to solve the problem.

Simplex Algorithm Example

LP Example, convert to std form

$$\text{maximize } 3x + 5y$$

subject to

$$x + y \leq 4$$

$$x + 3y \leq 6$$

$$x \geq 0, \quad y \geq 0$$

$$\text{maximize } 3x + 5y + 0u + 0v$$

subject to

$$x + y + u = 4$$

$$x + 3y + v = 6$$

$$x \geq 0, \quad y \geq 0, \quad u \geq 0, \quad v \geq 0$$

Variables u and v , transforming inequality constraints into equality constraints, are called **slack variables**

Basic feasible solutions

A *basic solution* to a system of m linear equations in n unknowns ($n \geq m$) is obtained by setting $n - m$ variables to 0 and solving the resulting system to get the values of the other m variables.

The variables set to 0 are called *nonbasic*;
the variables obtained by solving the system are called *basic*.

A basic solution is called *feasible* if all its (basic) variables are nonnegative.

Example $x + y + u = 4$

$$x + 3y + v = 6$$

(0, 0, 4, 6) is basic feasible solution

(x, y are non-basic; u, v are basic)

Simplex Tableau

maximize
 $z = 3x + 5y + 0u + 0v$
subject to
 $x + y + u = 4$
 $x + 3y + v = 6$
 $x \geq 0, y \geq 0, u \geq 0, v \geq 0$

	x	y	u	v	
u	1	1	1	0	4
v	1	3	0	1	6

objective row	-3	-5	0	0	0

basic variables
basic feasible solution
(0, 0, 4, 6)
value of z at (0, 0, 4, 6)

Outline of the Simplex Method

Step 0 [Initialization] Present a given LP problem in standard form and set up initial tableau.

Step 1 [Optimality test] If all entries in the objective row are nonnegative, then stop: the tableau represents an optimal solution.

Step 2 [Find entering variable] Select (the most) negative entry in the objective row.

Mark its column to indicate the **entering variable** and the pivot column.

Outline of the Simplex Method

Step 3 [Find departing variable]

- For each positive entry in the pivot column, calculate the θ -ratio by dividing that row's entry in the rightmost column by its entry in the pivot column. (If there are no positive entries in the pivot column then stop: the problem is unbounded.)
- Find the row with the **smallest positive θ -ratio**, mark this row to indicate the departing variable and the pivot row.

Step 4 [Form the next tableau]

- Divide all the entries in the **pivot row** by its entry in the pivot column.
- Subtract from each of the other rows, including the objective row, the new pivot row multiplied by the entry in the pivot column of the row in question.
- Replace the label of the pivot row by the variable's name of the pivot column and go back to Step 1.

maximize

$$3x + 5y + 0u + 0v = z$$

subject to

$$x + y + u = 4$$

$$x + 3y + v = 6$$

$$x \geq 0, y \geq 0, u \geq 0, v \geq 0$$

Example of Simplex Method

	x	y	u	v	
u	1	1	1	0	4
$\leftarrow v$	1	3	0	1	6
	-3	-5	0	0	0
					z



$$\text{BFS: } (0, 0, 4, 6)$$

$$z = 0$$

Pivot column 2

$$\text{Ratios: } 4/1=4, 6/3=\underline{2}$$

Pivot row 2

	x	y	u	v	
u	$\frac{2}{3}$	0	1	$-\frac{1}{3}$	2
y	$\frac{1}{3}$	1	0	$\frac{1}{3}$	2
	$-\frac{4}{3}$	0	0	$\frac{5}{3}$	10
					z



$$\text{BFS: } (0, 2, 2, 0)$$

$$\mathbf{z = 10}$$

Pivot column 1

$$\text{Ratios: } 2/(2/3)=3, \\ 2/(1/3)=6$$

Pivot row 1

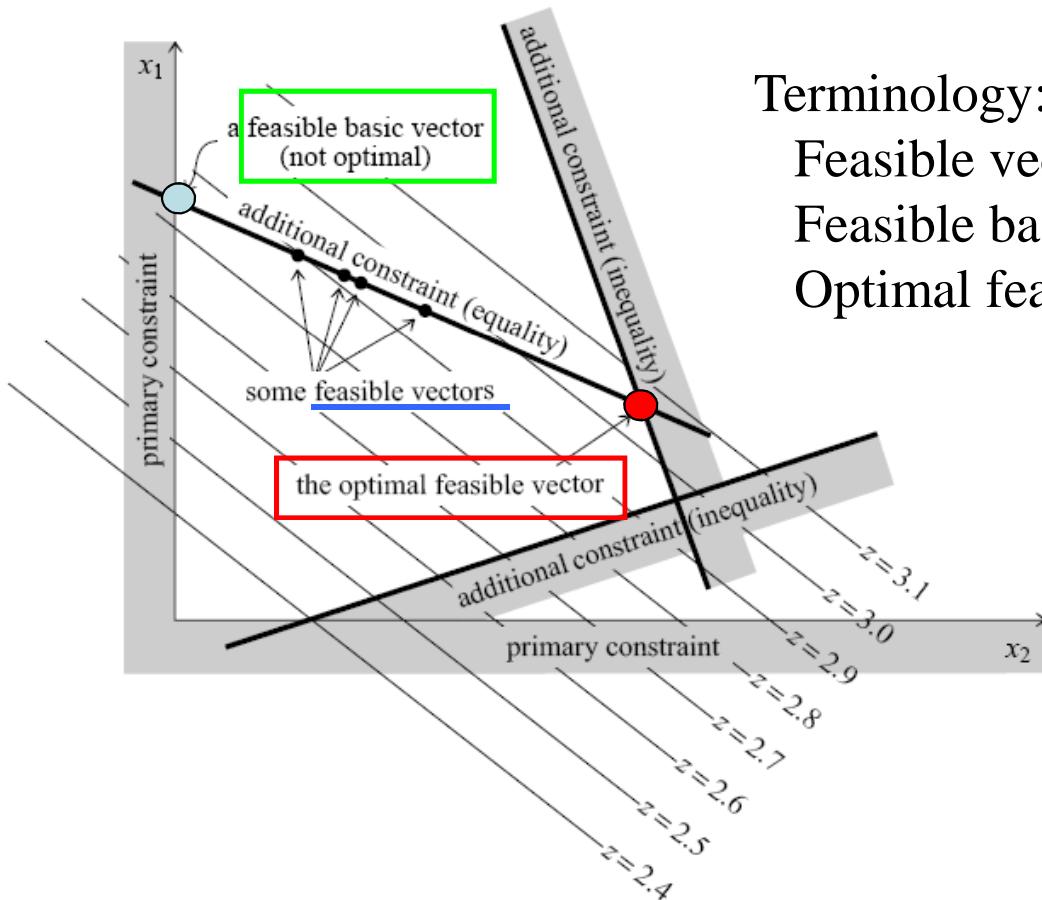
	x	y	u	v	
x	1	0	$\frac{3}{2}$	$-\frac{1}{2}$	3
y	0	1	$-\frac{1}{2}$	$\frac{1}{2}$	1
	0	0	2	1	14
					z

$$\text{BFS: } (3, 1, 0, 0)$$

$$\mathbf{z = 14}$$

LP Theory

Theory of Linear Optimization

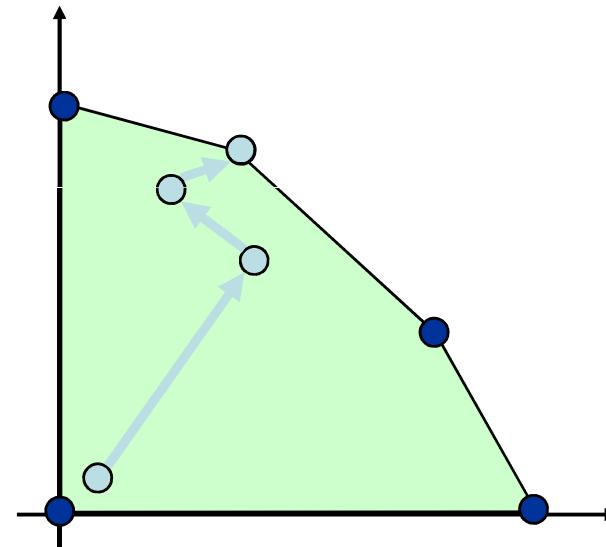


Terminology:

- Feasible vector
- Feasible basic vector
- Optimal feasible vector

Faster LP Algorithms

- Ellipsoid. ([Khachian 1979, 1980](#))
 - Solvable in polynomial time: $O(n^4 L)$ bit operations.
 - $n = \# \text{ variables}$
 - $L = \# \text{ bits in input}$
 - Theoretical tour de force.
 - Not remotely practical.
- Karmarkar's algorithm. ([Karmarkar 1984](#))
 - $O(n^{3.5} L)$.
 - Polynomial and reasonably efficient implementations possible.
- Interior point algorithms.
 - $O(n^3 L)$.
 - Competitive with simplex!
 - Dominates on simplex for large problems.
 - Extends to even more general problems.



Theory

- *Feasible region* is convex and bounded by hyperplanes
- Feasible vectors that satisfy N of the original constraints are termed *feasible basic vectors*.
- *Optimal* occur at boundary (gradient vector of objective function is always nonzero)
- *Combinatorial problem*: determining which N constraints (out of the $N+M$ constraints) would be satisfied by the optimal feasible vector

Duality

- Primal problem

maximize $c^T x$

subject to $Ax \leq b$, $x \geq 0$

- Dual problem

minimize $b^T y$

subject to $A^T y \geq c$, $y \geq 0$

Duality

- If a linear program has a finite optimal solution then so does the dual.
Furthermore, *the optimal values of the two programs are the same.*
- If either problem has an unbounded optimal solution, then the other problem has no feasible solutions.

Original

$$\text{Maximize } u = 5x + 2y, \text{ s.t.}$$

$$1x + 3y \leq 12 \quad \longleftarrow a$$

$$3x - 4y \leq 9 \quad \longleftarrow b$$

$$7x + 8y \leq 20 \quad \longleftarrow c$$

$$x, y \geq 0$$

Dual

$$\text{Minimize } w = 12a + 9b + 20c, \text{ s.t.}$$

$$a + 3b + 7c \geq 5$$

$$3a - 4b + 8c \geq 2$$

$$a, b, c \geq 0$$

Original constraints

$$\begin{array}{ccccc} 1 & x + & 3 & y \leq 12 \\ 3 & x - & 4 & y \leq 9 \\ 7 & x + & 8 & y \leq 20 \end{array}$$

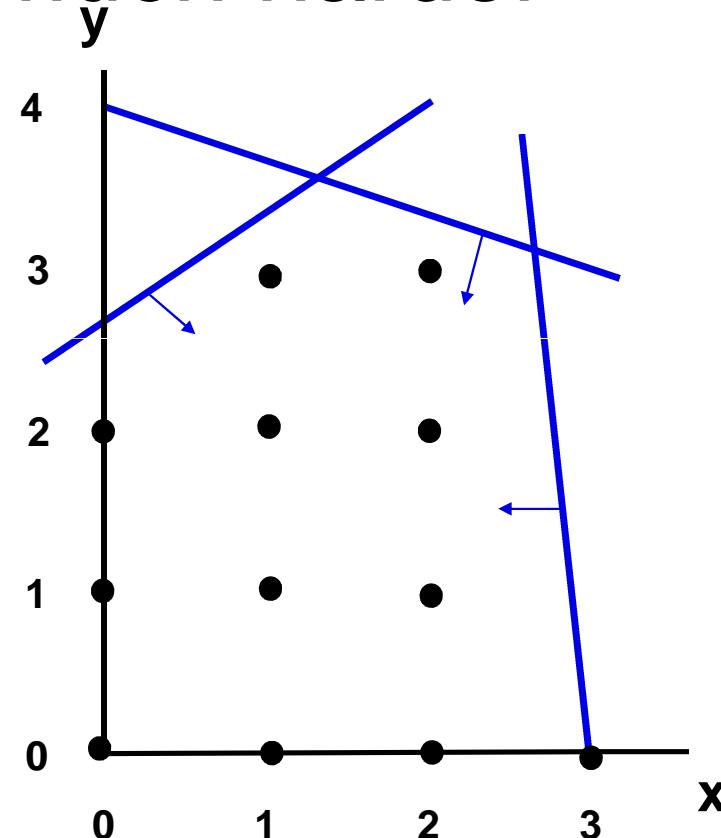
Dual constraints

$$1a + 3b + 7c \geq 5$$

$$3a - 4b + 8c \geq 2$$

But an Integer Program is different and much harder

1. Feasible region is a set of discrete points.
2. Can't be assured a corner point solution.
3. There are no "efficient" ways to solve an IP.
4. Solving it as an LP provides a relaxation and a bound on the solution.



Rounding ILP solutions can cause infeasibility

Maximize

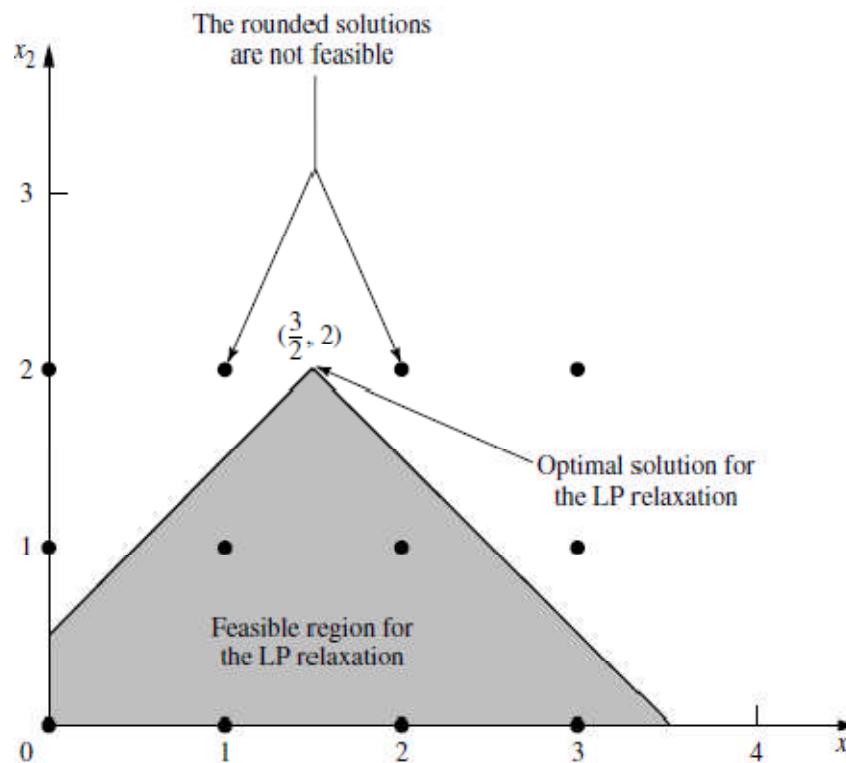
$$Z = x_2,$$

such that:

$$-x_1 + x_2 \leq 1/2$$

$$x_1 + x_2 \leq 3.5$$

$$\text{int } x_1, x_2 \geq 0.$$



From Hillier and Lieberman

Rounded ILP solutions can be very far from integer solutions

Maximize

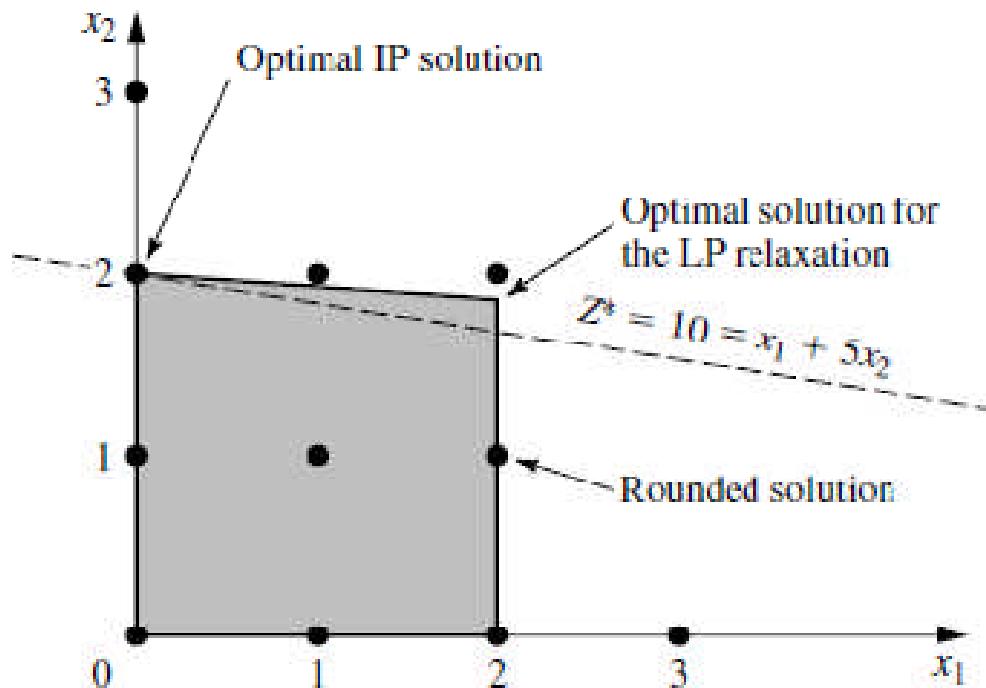
$$Z = x_1 + 5x_2,$$

such that:

$$x_1 + 10x_2 \leq 20$$

$$x_1 \leq 2$$

$$\text{int } x_1, x_2 \geq 0.$$



From Hillier and Lieberman

LP Homework

- Download glpk (gnu linear programming package) and
- Solve all the problems discussed in class.

Solve with GLPK

```
C:\mosh> cat cormen29-3.mod
var x1 >= 0;
var x2 >= 0;
var x3 >= 0;
maximize obj: 3 * x1 + 1 * x2 + 2 * x3 ;
s.t. c1: 1 * x1 + 1 * x2 + 3 * x3 <= 30;
s.t. c2: 2 * x1 + 2 * x2 + 5 * x3 <= 24;
s.t. c3: 4 * x1 + 1 * x2 + 2 * x3 <= 36;
solve;
display x1, x2, x3;
end;
```

- C:\> glpsol --math cormen29-3.mod

```
GLPSOL: GLPK LP/MIP Solver, v4.47
Parameter(s) specified in the command line:
--math cormen29-3.mod
Reading model section from cormen29-3.mod...
18 lines were read
Generating obj...
Generating c1...
Generating c2...
Generating c3...
Model has been successfully generated
GLPK Simplex Optimizer, v4.47
4 rows, 3 columns, 12 non-zeros
Preprocessing...
3 rows, 3 columns, 9 non-zeros
Scaling...
A: min|aij| = 1.000e+00 max|aij| = 5.000e+00 ratio = 5.000e+00
Problem data seem to be well scaled
Constructing initial basis...
Size of triangular part = 3
* 0: obj = 0.000000000e+00 infeas = 0.000e+00 (0)
* 3: obj = 2.800000000e+01 infeas = 0.000e+00 (0)
```

OPTIMAL SOLUTION FOUND

```
Time used: 0.0 secs
Memory used: 0.1 Mb (108677 bytes)
Display statement at line 17
```

x1.val = 8

x2.val = 4

x3.val = 0

1. Model has been successfully processed

Solve with Mathematica

```
In[9] = Maximize[ {  
 3 x1 + x2 + 2 x3,  
  x1 + x2 + 3 x3 <= 30 &&  
 2 x1 + 2 x2 + 5 x3 <= 24 &&  
 4 x1 + x2 + 2 x3 <= 36,  
 x1 >= 0 && x2 >= 0 && x3 >= 0 },  
 { x1, x2,x3} ]
```

```
Out[9] = {28, {x1 -> 8, x2 -> 4, x3 -> 0}}
```

Mathematica Interior Point

```
In = LinearProgramming[ {-3, -1, -2},  
  {{-1, -1, -3}, {-2, -2, -5}, {-4, -1, -2}},  
  {-30, -24, -36},  
  Method -> "InteriorPoint"]
```

```
Out = {8., 4., 9*10-7}
```

Solve with Maple V.4 simplex

```
Maple> with(simplex);
maximize(3*x1+x2+2*x3, {
    x1+ x2+3*x3<=30,
    2*x1+2*x2+5*x3<=24,
    4*x1+ x2+2*x3<=36
}, NONNEGATIVE);
```

$$\{x_3 = 0, x_1 = 8, x_2 = 4\}$$

Using python pysimplex

C:\> cat example.py

```
from pysimplex import *
if __name__ == '__main__':
    set_printoptions(precision=2)
    t = Tableau([-3,-1,-2])          # max z = 3x + 1y + 2z
    t.add_constraint([1, 1, 3], 30)   # x + y + 3z <= 30
    t.add_constraint([2, 2, 5], 24)   # 2x + 2y + 5z <= 24
    t.add_constraint([4, 1, 2], 36)   # 4x + 1y + 2z <= 36
    t.solve(5)
```

Output of example.py

solve:

```
[[ -3 -1 -2 0 0 0 0]
 [ 1 1 3 1 0 0 30]
 [ 2 2 5 0 1 0 24]
 [ 4 1 2 0 0 1 36]]
```

ratios: [30, 12, 9]

step=1, pivot column: 2, row: 4

```
[[ 0 -0.25 -0.5 0 0 0.75 27]
 [ 0 0.75 2.5 1 0 -0.25 21]
 [ 0 1.5 4. 0 1 -0.5 6]
 [ 1 0.25 0.5 0 0 0.25 9]]
```

ratios: [8.4, 1.5, 18]

step=2, pivot column: 4, row: 3

```
[[ 0 -0.06 0 0 0.13 0.69 27.75]
 [ 0 -0.19 0 1 -0.63 0.06 17.25]
 [ 0 0.38 1 0 0.25 -0.13 1.5 ]
 [ 1 0.06 0 0 -0.13 0.31 8.25]]
```

ratios: [1724999982.75, 4, 132]

step=3, pivot column: 3, row: 3

```
[[ 0 0 0.17 0 0.17 0.67 28 ]
 [ 0 0 0.5 1 -0.5 0. 18 ]
 [ 0 1 2.67 0 0.67 -0.33 4 ]
 [ 1 0 -0.17 0 -0.17 0.33 8 ]]
```

Solved in 4 steps
