COT 6936: Topics in Algorithms

Giri Narasimhan

ECS 254A / EC 2443; Phone: x3748

giri@cs.fiu.edu

http://www.cs.fiu.edu/~giri/teach/COT6936_S12.html

https://moodle.cis.fiu.edu/v2.1/course/view.php?id=**174**

Purpose of this class

- First course in algorithms is inadequate preparation for most PhD students
  - Learn standard techniques
  - Solve standard problems
  - Learn basic analysis techniques
  - Need to go beyond that!
- This course
  - Model/formalize a problem
  - Leverage existing solutions
  - Create your own solutions

Expectations

- Attend class
- Do required reading **before** class
- Participate in class discussions
- Team work; discussion groups
- Solve practical research problems
- Make a presentation; write a report
  - need a research component; may implement
- Write research paper
- No cell phones, SMS, or email during class

Evaluation

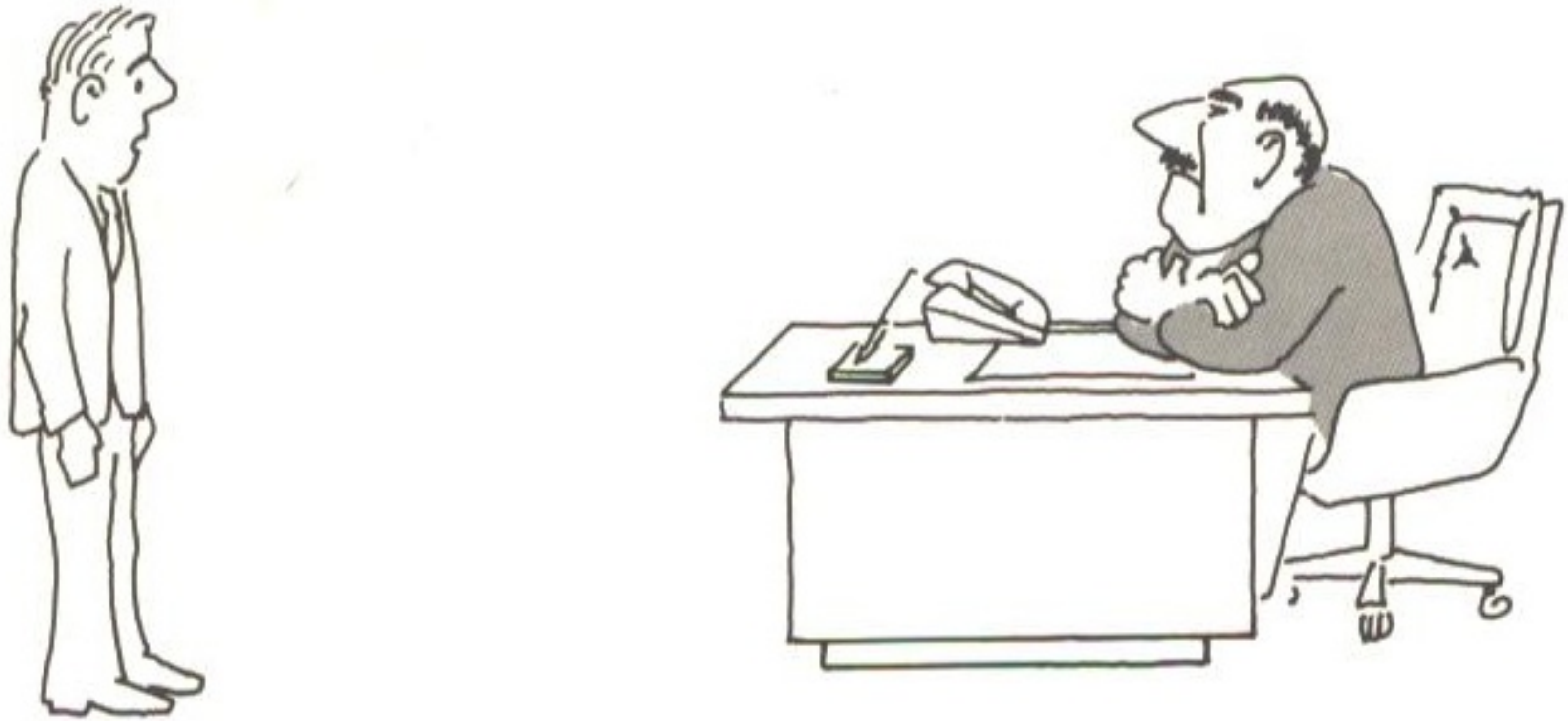
• Exam (1)	20%
• Quizzes	5%
• Homework Assignments	15%
• Semester Project	40%
• Class Participation	20%

Semester Project & Exam Schedule

- Milestones:
  - By Jan 23: Meet with me and discuss project
  - By Jan 30: Send me email with project team information and topic
  - Feb 20: Short presentation giving intro to project, problem definition, notation, and background
  - March 5: Take-home Exam
  - April 16, 23: Final presentation of project
  - April 24: Written report on project

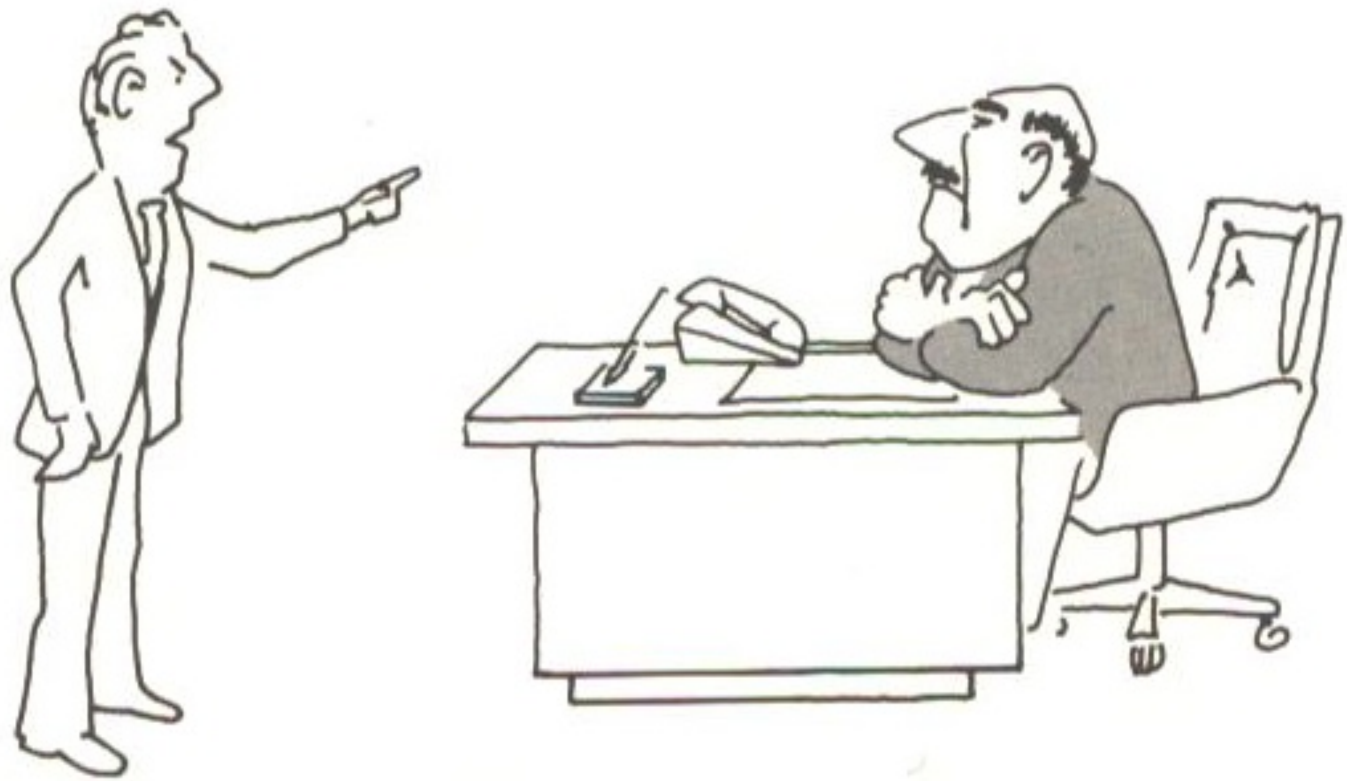
Why should I care about Algorithms?

Cartoon from *Intractability* by Garey and Johnson



"I can't find an efficient algorithm, I guess I'm just too dumb."

Why are theoretical results useful?



“I can’t find an efficient algorithm, because no such algorithm is possible!”

Cartoon from *Intractability* by Garey and Johnson

Why are theoretical results useful?

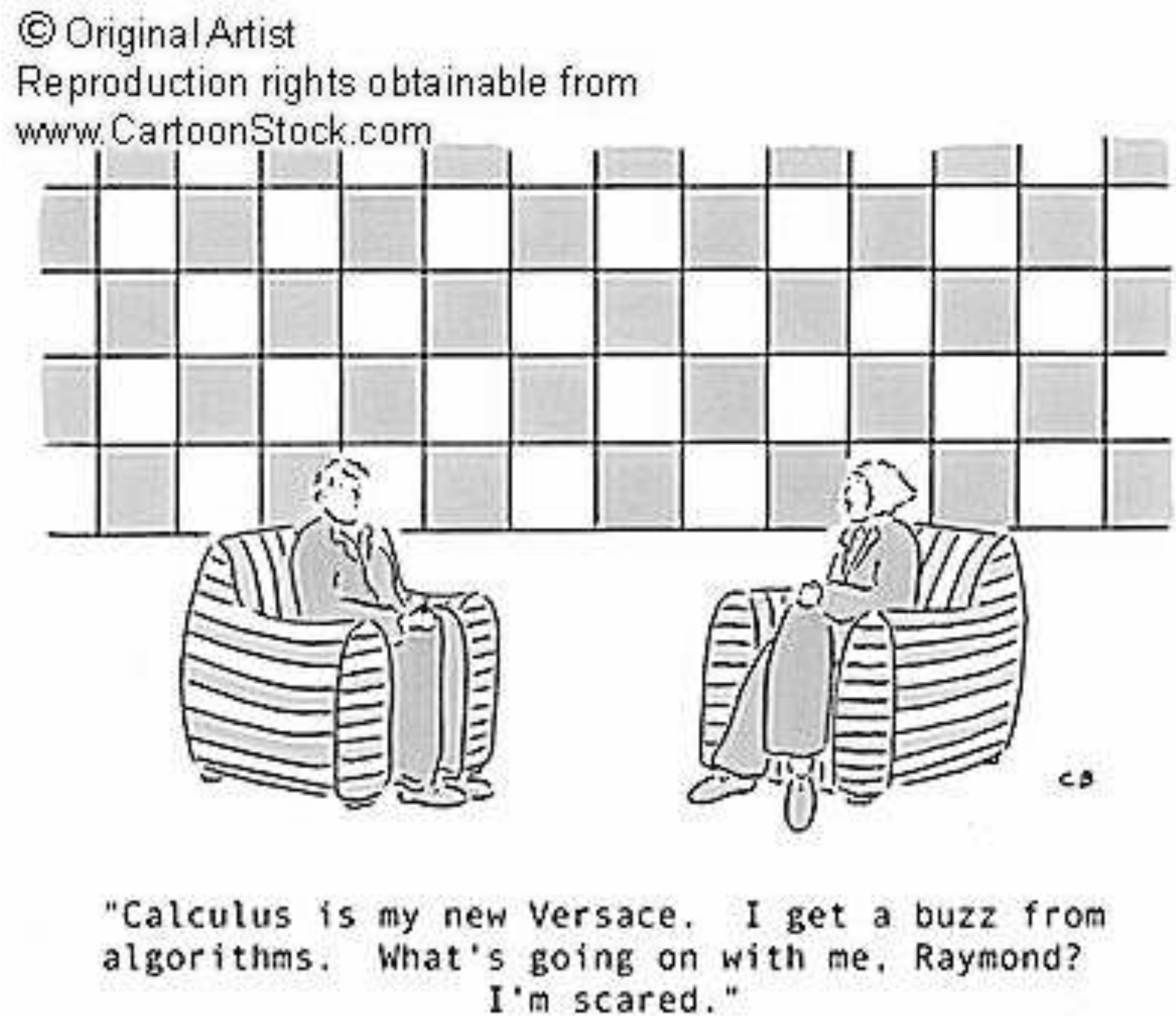
"I can't find an efficient algorithm, but neither can all these famous people."

Cartoon from *Intractability* by Garey and Johnson

What if efficient algorithms don't exist

- Find good approximation algorithms
  - Quality of the solution is guaranteed
- Find good heuristic algorithms
- Understand nature of inputs in practice
- Perform many experiments after implementing

If you like Algorithms, nothing to worry about!



Classical (Theoretical) Algorithmic Model

- Input-output description provided
- Input provided & stored in memory
- Output computed & stored or output immediately
- Entire program stored in memory
- Algebraic Computation-Tree Model (Variants: indirection, floor function, square root)
- Space (?) and time (?) efficiency
- Deterministic and Sequential algorithms
- Worst-case analysis
- No other factors to consider

Find a “good” student

- Director of SCIS says to you: “Find me a good CS student.”
- You ask: “What do you mean by good?”
- Director says: “S/he must be at least as good as at least half of our current students.”

Naïve Solution

- Solution 1
  - Email (or contact or inspect) $N/2 + 1$ students and pick best among them
  - Too inefficient
- Solution 2
  - Pick a random student
  - May be wrong about $\frac{1}{2}$ the time
- Solution 3
  - Pick r random students and pick best among them

Solution 3

- Prob of failure: $\frac{1}{2}$
- Prob of failure: $(1/2)^r$

Randomized algorithms

- Useful when you can tolerate failure
- 2 kinds of randomized algorithms
  - Always fast, sometimes wrong (Monte Carlo)
  - Always correct, sometimes slow (Las Vegas)
- Complexity classes: RP, BPP, ZPP, …
- Focus of study
  - Design
  - Analysis
    - Time, Failure probability, Performance, Tradeoffs

Applications of Randomized Algorithms

- Contention Resolution: network protocol, resource sharing
- Hashing
- Storage: multi-level storage management
- Packet Routing
- Load Balancing

Facility Location

- Given: Location of all fire-stations in Miami
- Output: Optimal location of next fire-station
- Strategy: find largest empty region

Achieving Height Diversity

- Given: Heights of all students in class
- Problem:
  - Want to achieve diversity in heights
  - Allowed to add a student. How to pick?
- Approach:
  - Minimize the largest empty height range
- Solution:
  - Find biggest empty height range and pick student in that range

Achieving Height Diversity: a variant

- Given: Heights of all students in class
- Problem:
  - Want to achieve diversity in heights
  - Allowed to remove a student. How to pick?
- Approach:
  - Maximize the smallest empty height range
- Solution:
  - Find smallest empty height range and pick one of two students

Heights of Students: What we know

- One problem is harder than the other!
- Which one and why? Homework!
- One has a lower bound!
  - Relationship to EUP?
- The other can be solved faster, but with a different/stronger computational model!

Updating a Binary Counter

- How many bits are changed when a binary number is incremented?
  - Worst-case?
  - Average-case?
  - Amortized analysis? Average cost over a worst-case sequence of operations.

Binary Counter: What we know

- Worst case per increment = O(# bits)
- Average case per increment = O(# bits)
- Amortized complexity = ??

Other Algorithmic Models

- Practical problems
  - Making spot decisions: <u>ON-LINE Algorithms</u>
    - Often randomized
    - Use current state
    - Sophisticated: use past history
  - Not having enough memory or computing power: <u>STREAMING Algorithms</u>

Practical Algorithmic Models

- Sequential Algorithms
  - Worst-case / average-case analysis
  - Amortized Analysis
- Parallel Algorithms
- On-line Algorithms
- Randomized Algorithms
- Streaming Algorithms
- External Memory Algorithms
- Limited space/time/power Algorithms
- Making use of cache: Cache-aware Algorithms

Experimental Algorithms

- How to do good experiments in practice?
  - Testing for correctness
  - Testing for performance
    - Modeling inputs in practice
    - Trying different input distributions
    - Optimizing performance for special input distributions

Additional Topics

- Approximation Algorithms
- Computational Geometry
- Computational Biology
  - String Algorithms
- Computational Finance
- Combinatorial Optimization
- Algorithmic Game Theory
- Heuristic Algorithms
- Problem Modeling and Transformations

Paging Algorithms

Here are 3 well-known paging algorithms

- Least Recently Used (LRU): evict item whose most recent request was furthest in the past
- First-in, First-out (FIFO): evict item that was brought in furthest in the past
- Least Frequently Used (LFU): evict item that has been requested least often

Which ones are good algorithms and why?

What is an optimal algorithm?

Drunken sailors and cabins

- A ship arrives at a port. 40 sailors go ashore for revelry. They return to the ship rather inebriated. Being unable to remember their cabin location, they find a random unoccupied cabin to sleep the night. How many sailors are expected to sleep in their own cabins?
- Variants? Generalizations?

Homework #1 – is here!

- Achieving diversity in heights:
  - Largest empty range problem
  - Smallest empty range problem
  - Which is harder and why?
- Binary Counter
- 2SAT
- Drunken Sailors problem
  - How many sailors will sleep in their own cabins?
- ACM Programming Contest Problems

NP-Completeness

- *Computers and Intractability: A Guide to the theory of NP-Completeness*, by Garey and Johnson
  - Compendium (100 pages) of NP-Complete and related problems

Polynomial-time computations

- An algorithm has (worst-case) time complexity $O(T(n))$ if it runs in time at most $cT(n)$ for some $c > 0$ and for every input of length n. [Time complexity ≈ worst-case.]
- An algorithm is a polynomial-time algorithm if its (worst-case) time complexity is $O(p(n))$, where $p(n)$ is some polynomial in n. [Polynomial in what?]
- Composition of polynomials is a polynomial. [What are the implications?]

The class $\mathcal{P}$

- A problem is in $\mathcal{P}$ if there exists a polynomial-time algorithm for the problem. [$\mathcal{P}$ is therefore a class of problems, not algorithms.]
- Examples of problems in $\mathcal{P}$
  - ***DFS:*** Linear-time algorithm exists
  - ***Sorting:*** O(n log n)-time algorithm exists
  - ***Bubble Sort:*** Quadratic-time algorithm $O(n^2)$
  - ***APSP:*** Cubic-time algorithm $O(n^3)$

The class $\mathcal{NP}$

- A problem is in $\mathcal{NP}$ if there exists a non-deterministic polynomial-time algorithm that solves the problem.
- [Alternative definition] A problem is in $\mathcal{NP}$ if there exists a (deterministic) polynomial-time algorithm that verifies a solution to the problem.
- All problems in $\mathcal{P}$ are in $\mathcal{NP}$. [The converse is the big deal!]

TSP: Traveling Salesperson Problem

- **Input:**
  - Weighted graph, G
  - Length bound, B
- **Output:**
  - Is there a TSP tour in G of length at most B?
- Is TSP in $\mathcal{NP}$?
  - YES. Easy to verify a given solution.
- Is TSP in $\mathcal{P}$?
  - OPEN!
  - One of the greatest unsolved problems of this century!
  - Same as asking: **Is $\mathcal{P} = \mathcal{NP}$?**

So, what is *NP-Complete*?

- *NP-Complete* problems are the "hardest" problems in *NP*.
- We need to formalize the notion of "hardest".

Terminology

- Problem:
  - An abstract problem is a function (relation) from a set I of instances of the problem to a set S of solutions.

    $p: I \rightarrow S$
  - An instance of a problem *p* is obtained by assigning values to the parameters of the abstract problem.
  - Thus, describing set of all instances (i.e., possible inputs) and the set of corresponding outputs defines a problem.
- Algorithm:
  - An algorithm that solves problem *p* must give correct solutions to all instances of the problem.
- Polynomial-time algorithm:

Terminology (Cont' d)

- Input Length:
  - length of an <u>encoding</u> of an instance of the problem.
  - Time and space complexities are written in terms of it.
- Worst-case time/space complexity of an algorithm
  - Maximum time/space required by algorithm on any input of length n.
- Worst-case time/space complexity of a problem
  - UPPER BOUND: worst-case time complexity of best existing algorithm that solves the problem.
  - LOWER BOUND: (provable) worst-case time complexity of best algorithm (need not exist) that could solve the problem.
  - LOWER BOUND ≤ UPPER BOUND
- Complexity Class $\mathcal{P}$:
  - Set of all problems *p* for which polynomial-time algorithms exist

Terminology (Cont’d)

- Decision Problems:
  - Problems for which the solution set is {yes, no}
  - Example: Does a given graph have an odd cycle?
  - Example: Does a given weighted graph have a TSP tour of length at most B?
- Complement of a decision problem:
  - Problems for which the solution is “complemented”.
  - Example: Does a given graph NOT have an odd cycle?
  - Example: Is every TSP tour of a given weighted graph of length > B?
- Optimization Problems:
  - Problems where one is maximizing/minimizing an objective function.
  - Example: Given a weighted graph, find a MST.
  - Example: Given a weighted graph, find an optimal TSP tour.
- Verification Algorithms:
  - Given a problem instance i and a certificate s, is s a solution for instance i?

Terminology (Cont'd)

- Complexity Class *P*:
  - Set of all problems *p* for which polynomial-time algorithms exist.
- Complexity Class *NP*:
  - Set of all problems *p* for which polynomial-time verification algorithms exist.
- Complexity Class *co-NP*:
  - Set of all problems *p* for which polynomial-time verification algorithms exist for their **complements**, i.e., their complements are in *NP*.

Terminology (Cont'd)

- **Reductions:** $p_1 \rightarrow p_2$
  - A problem p_1 is reducible to p_2, if there exists an algorithm R that takes an instance i_1 of p_1 and outputs an instance i_2 of p_2, with the constraint that the solution for i_1 is YES if and only if the solution for i_2 is YES.
  - Thus, R converts YES (NO) instances of p_1 to YES (NO) instances of p_2.
- Polynomial-time reductions: $p_1 \xrightarrow{P} p_2$
  - Reductions that run in polynomial time.

- If $p_1 \xrightarrow{P} p_2$, then
  - If p_2 is easy, then so is p_1. $\quad p_2 \in \mathcal{P} \Rightarrow p_1 \in \mathcal{P}$
  - If p_1 is hard, then so is p_2. $\quad p_1 \notin \mathcal{P} \Rightarrow p_2 \notin \mathcal{P}$

What are *NP-Complete* problems?

- These are the hardest problems in *NP*.
- A problem p is *NP-Complete* if
  - there is a polynomial-time reduction from **every** problem in *NP* to p.
  - p $\in$ *NP*
- How to prove that a problem is *NP-Complete*?

- **Cook's Theorem**: [1972]
  - The SAT problem is *NP-Complete*.

Steve Cook, Richard Karp, Leonid Levin

NP-Complete vs *NP-Hard*

- A problem p is *NP-Complete* if
  - there is a polynomial-time reduction from **every** problem in *NP* to p.
  - p $\in$ *NP*
- A problem p is *NP-Hard* if
  - there is a polynomial-time reduction from **every** problem in *NP* to p.

- Remember: to prove problem p is *NP-Complete* you have to reduce a *NP-Complete* problem to p.

The SAT Problem: an example

- Consider the boolean expression:

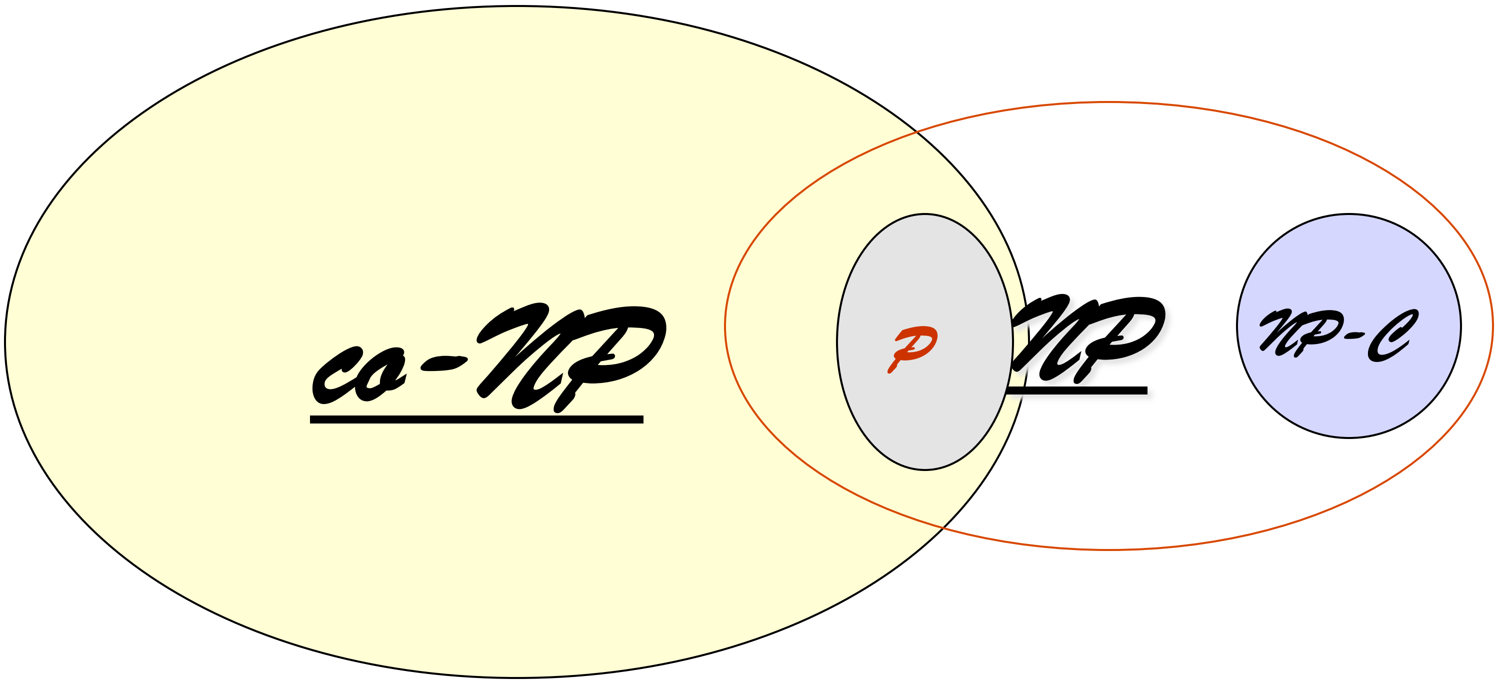
  $C = (a \vee \neg b \vee c) \wedge (\neg a \vee d \vee \neg e) \wedge (a \vee \neg d \vee \neg c)$
- Is C satisfiable? [Does there exist a True/False assignments to the boolean variables a, b, c, d, e, such that C is True?]
- If there are n boolean variables, then there are 2^n different truth value assignments.
- However, a solution can be quickly verified!

The SAT (Satisfiability) Problem

- Input: Boolean expression C in Conjunctive normal form (CNF) in n variables and m clauses.
- Question: Is C satisfiable?
  - Let $C = C_1 \wedge C_2 \wedge \ \ldots \ \wedge C_m$
  - Where each $C_i = \left(y_1^i \vee y_2^i \vee \cdots \vee y_{k_i}^i\right)$
  - And each $y_j^i \in \{x_1, \neg x_1, x_2, \neg x_2, \ldots, x_n, \neg x_n\}$
  - We want to know if there exists a truth assignment to all the variables in the boolean expression C that makes it true.
- Steve Cook showed that the problem of deciding whether a non-deterministic Turing machine T accepts an input w or not can be written as a boolean expression C_T for a SAT problem. The boolean expression will have length bounded by a polynomial in the size of T and w.

> - How to now prove Cook's theorem? Is SAT in $\mathcal{NP}$?
> - Can every problem in $\mathcal{NP}$ be poly. reduced to it ?

The problem classes and their relationships



More *NP-Complete* problems

3SAT

- Input: Boolean expression C in Conjunctive normal form (CNF) in n variables and m clauses. Each clause has at most three literals.
- Question: Is C satisfiable?
  - Let $C = C_1 \wedge C_2 \wedge \ldots \wedge C_m$
  - Where each $C_i = \left(y_1^i \vee y_2^i \vee y_3^i\right)$
  - And each $y_j^i \in \{x_1, \neg x_1, x_2, \neg x_2, \ldots, x_n, \neg x_n\}$
  - We want to know if there exists a truth assignment to all the variables in the boolean expression C that makes it true.

3SAT is *NP-Complete.*

3SAT is *NP-Complete*

- 3SAT is in *NP.*
- SAT can be reduced in polynomial time to 3SAT.
- This implies that every problem in *NP* can be reduced in polynomial time to 3SAT. Therefore, 3SAT is *NP-Complete*.
- So, we have to design an algorithm such that:
  - Input: an instance C of SAT
  - Output: an instance C' of 3SAT such that satisfiability is retained. In other words, C is satisfiable if and only if C' is satisfiable.

3SAT is *NP-Complete*

- Let C be a SAT instance with clauses $C_1, C_2, \ldots, C_m$
- Let C_i be a disjunction of $k > 3$ literals.

$C_i = \quad y_1 \vee y_2 \vee \ldots \vee y_k$

- Rewrite C_i as follows:

$$
\begin{aligned}
C'_i = \quad & (y_1 \vee y_2 \vee z_1) \wedge \\
& (\neg z_1 \vee y_3 \vee z_2) \wedge \\
& (\neg z_2 \vee y_4 \vee z_3) \wedge \\
& \ldots \\
& (\neg z_{k-3} \vee y_{k-1} \vee y_k)
\end{aligned}
$$

- Claim: C_i is satisfiable if and only if C'_i is satisfiable.

More *NP-Complete* problems?

2SAT

- **Input**: Boolean expression C in Conjunctive normal form (CNF) in n variables and m clauses. Each clause has at most <u>three</u> literals.
- **Question**: Is C satisfiable?
  - Let $C = C_1 \wedge C_2 \wedge \ \dots \ \wedge C_m$
  - Where each $C_i = \left(y_1^i \vee y_2^i\right)$
  - And each $y_j^i \in \{x_1, \neg x_1, x_2, \neg x_2, \dots, x_n, \neg x_n\}$
  - We want to know if there exists a truth assignment to all the variables in the boolean expression C that makes it true.

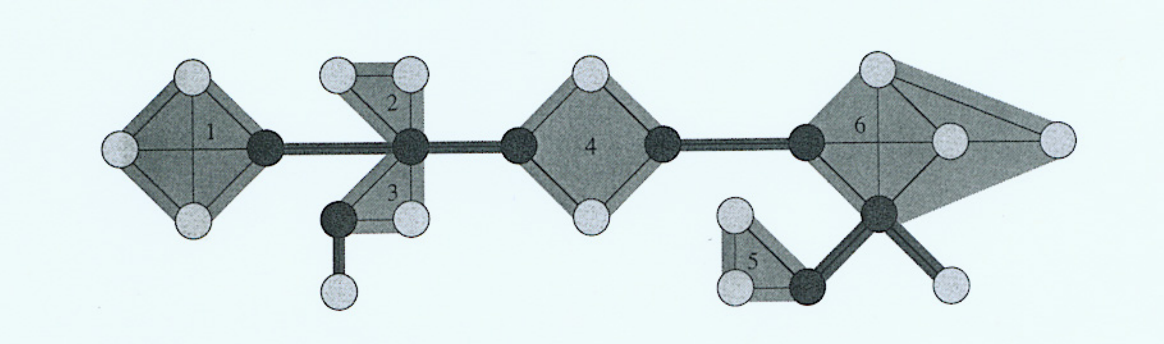
2SAT is in *P*.

2SAT is in $\mathcal{P}$

- If there is only one literal in a clause, it must be set to true.
- If there are two literals in some clause, and if one of them is set to false, then the other must be set to true.
- Using these constraints, it is possible to check if there is some inconsistency.
- How? Homework: do not submit!

The CLIQUE Problem

- A **clique** is a completely connected subgraph.

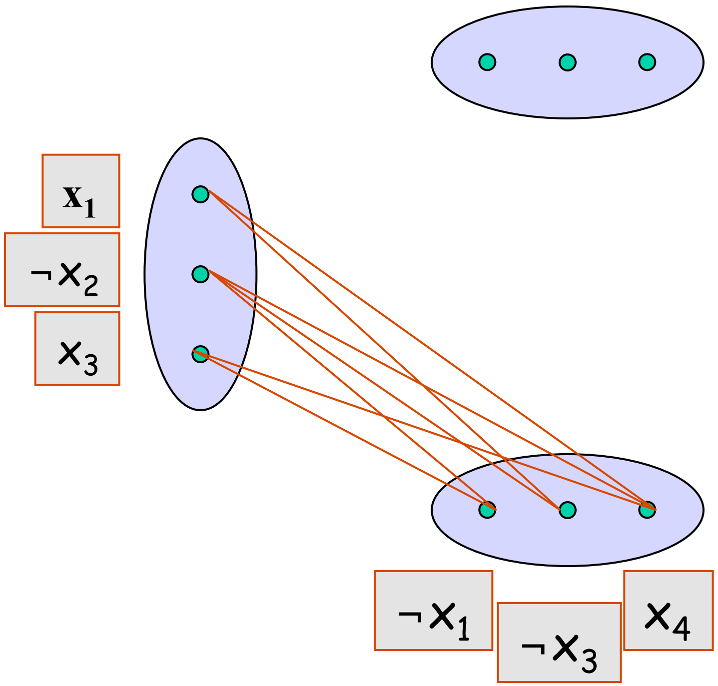


CLIQUE

- **Input**: Graph G(V,E) and integer k
- **Question**: Does G have a clique of size k?

CLIQUE is *NP-Complete*

- CLIQUE is in *NP.*
- Reduce 3SAT to CLIQUE in polynomial time.
- $F = (x_1 \vee \neg x_2 \vee x_3)\ (\neg x_1 \vee \neg x_3 \vee x_4)\ (x_2 \vee x_3 \vee \neg x_4)\ (\neg x_1 \vee \neg x_2 \vee x_3)$

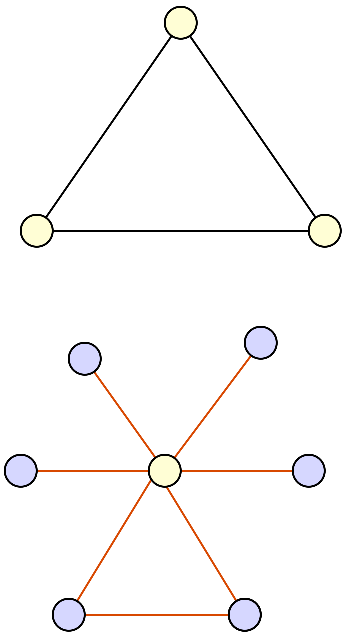


F is satisfiable if and only if G has a clique of size k where k is the number of clauses in F.

Vertex Cover

A vertex cover is a set of vertices that "covers" all the edges of the graph.

Examples

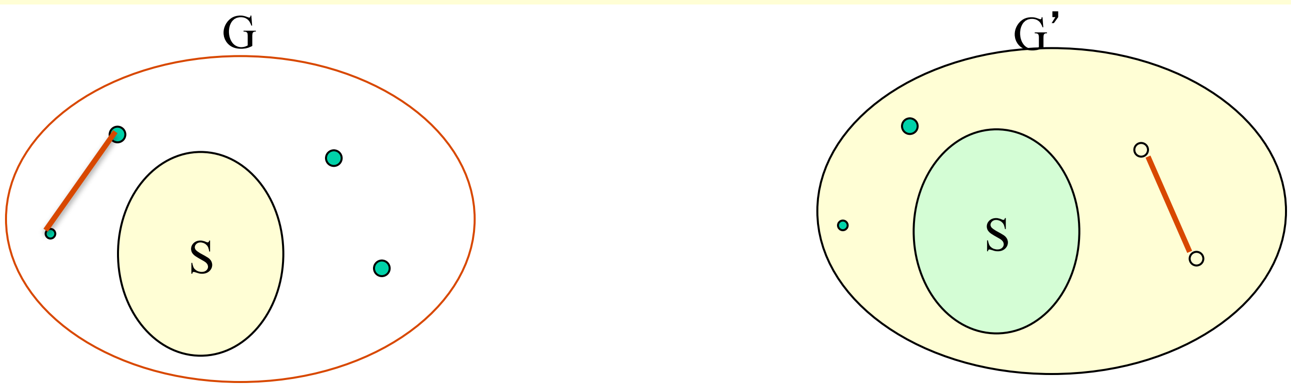


Vertex Cover (VC)

Input: Graph G, integer k

Question: Does G contain a vertex cover of size k?

- VC is in *NP*.
- polynomial-time reduction from CLIQUE to VC.
- Thus VC is *NP-Complete.*



Claim: G' has a clique of size k' if and only if G has a VC of size k = n - k'

Hamiltonian Cycle Problem (HCP)

Input: Graph G

Question: Does G contain a hamiltonian cycle?

- HCP is in NP.
- There exists a polynomial-time reduction from 3SAT to HCP.
- Thus HCP is NP-Complete.

Shortest Path vs Longest Path

Input: Graph G with edge weights, vertices u and v, bound B

Question: Does G contain a shortest path from u to v of length at most B?

Question: Does G contain a longest path from u to v of length at most B?

Homework: Listen to Cool MP3:

http://www.cs.princeton.edu/~wayne/kleinberg-tardos/longest-path.mp3

Perfect (2-D) Matching vs 3-D Matching

1. Input: Bipartite graph, $G(U,V,E)$
   Question: Does G have a perfect matching?

2. Input: Sets U and V, and E = subset of $U \times V$
   Question: Is there a subset of E of size $|U|$ that covers U and V? [Related to 1.]

3. Input: Sets U,V,W, & E = subset of $U \times V \times W$
   Question: Is there a subset of E of size $|U|$ that covers U, V and W?

Coping with NP-Completeness

- Approximation: Search for an "almost" optimal solution with provable quality.
- Randomization: Design algorithms that find "provably" good solutions with high prob and/or run fast on the average.
- Restrict the inputs (e.g., planar graphs), or fix some input parameters.
- Heuristics: Design algorithms that work "reasonably well".

Reading

- Read Background
  - Algorithms & Discrete Math Fundamentals
    - Cormen, et al., Chapters 1-16, 22-25
  - NP-Completeness
    - Cormen et al., Chapter 34
    - Appendix (p187-288) form Garey & Johnson
- Next Class
  - Approximation Algorithms
    - Cormen et al., Chapter 35
    - Kleinberg, Tardos, Chapter 11
    - Books by Vazirani and Hochbaum/Shmoys