NP

CSE 373 - Data Structures

June 5, 2002

Readings and References

- Reading
  - Section 9.7, *Data Structures and Algorithm Analysis in C*, Weiss
- Other References
  - Chapter 34, "NP-Completeness", *Introduction to Algorithms*, Cormen, Leiserson, Rivest, Stein
  - "More than any time in history mankind faces a crossroads. One path leads to despair and utter hopelessness, the other to total extinction. Let us pray that we have the wisdom to choose correctly." Woody Allen

With permission, many of the slides in this lecture are based on slides by Anna Karlin.

Finding Hamiltonian Circuits

- Does G contain a Hamiltonian circuit?
  - No known easy algorithm for checking this…
- Try this
  - Search through *all paths* to find one that visits each vertex exactly once
  - Can use your favorite graph search algorithm (DFS!) to find various paths
  - This is an *exhaustive search* ("brute force") algorithm

Polynomial vs Exponential Time

- Most of our algorithms have been O(log N), O(N), O(N log N) or $O(N^2)$ running time for inputs of size N
  - These are all *polynomial time* algorithms
  - Their running time is $O(N^k)$ for some $k > 0$
- Exponential time B^N is asymptotically worse than *any* polynomial function N^k for any k
  - For any k, N^k is $o(B^N)$ for any constant $B > 1$
- Polynomial time algorithms are "fast" algorithms
- Exponential time algorithms are "not fast"
  - superpolynomial or "dog slow"

The complexity class P

- The set P is defined as the set of all problems that can be *solved* in polynomial worse case time
  - this is the polynomial time complexity class
  - contains problems whose time complexity to solve is $O(N^k)$ for some k
- Examples of problems in P
  - searching, sorting, topological sort, single-source shortest path, Euler circuit, etc.

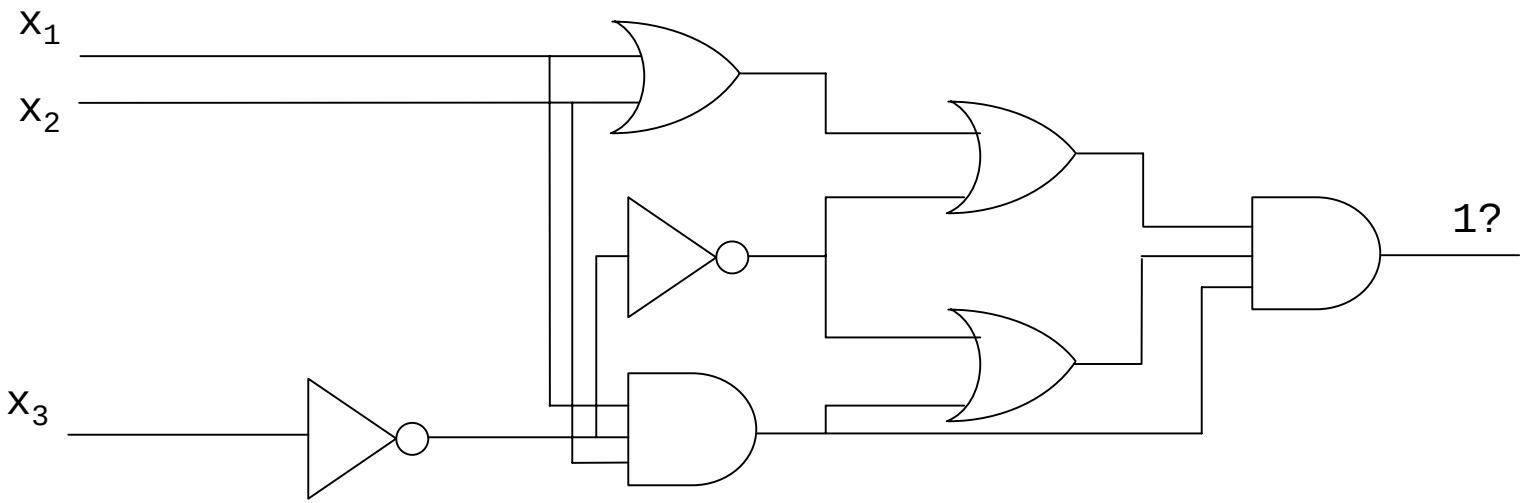
The complexity class NP

- The set NP is the set of all problems for which a given *candidate solution* can be *checked* in polynomial time
- Example of a problem in NP:
  - Hamiltonian circuit problem
  - Given a candidate path, can test in linear time if it is a Hamiltonian circuit – just check if all vertices are visited exactly once in the candidate path, repeating only the start/finish vertex

Nondeterministic Polynomial time

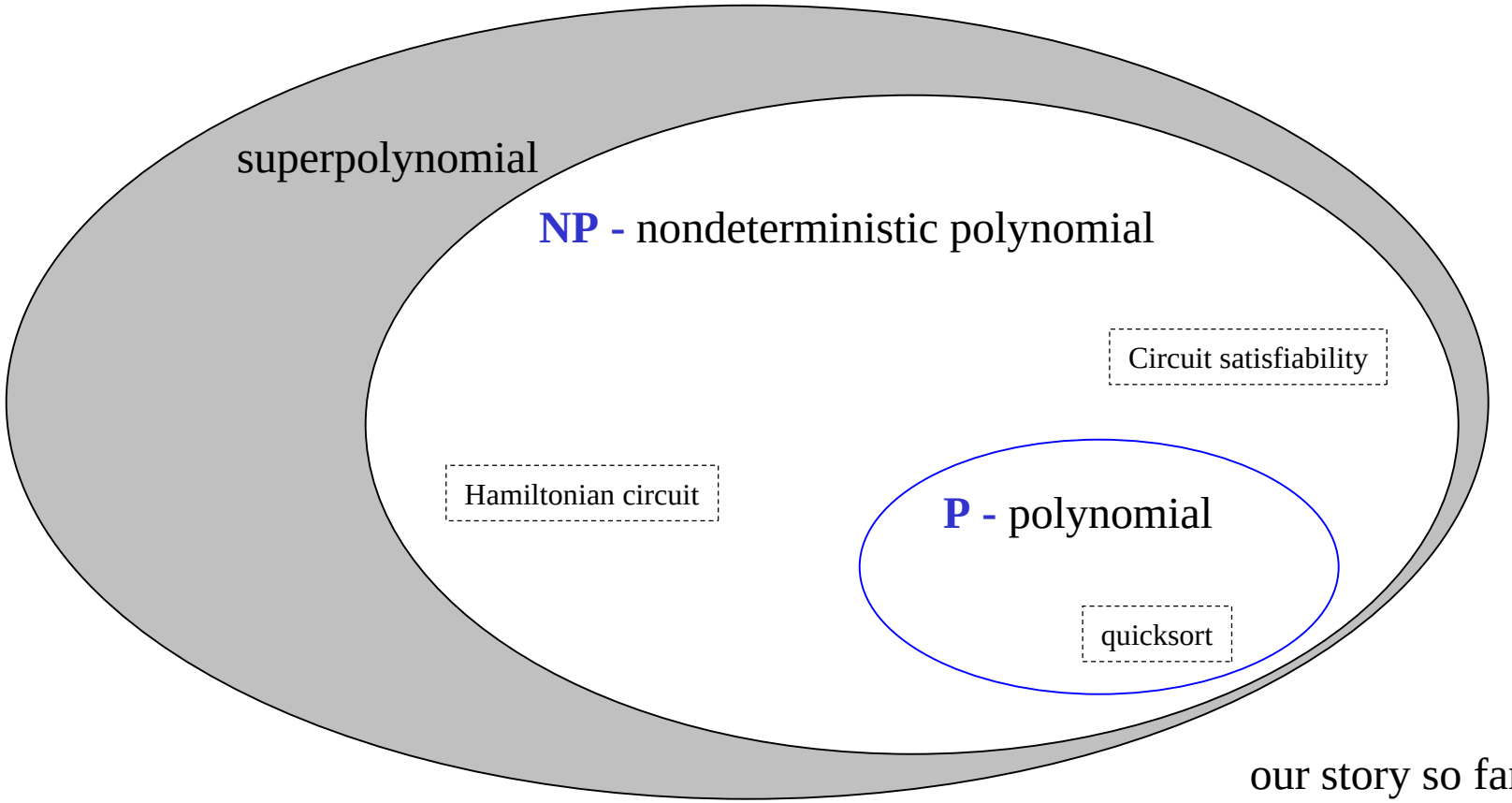
- Why “nondeterministic”?
  - A nondeterministic algorithm is free to correctly choose the next step to execute on the path to a solution
  - Nondeterministic algorithms don’t exist – purely theoretical idea invented to understand how hard a problem could be
- But if we can check a solution in polynomial time, then the complexity analysis corresponds to algorithms that can search all possible solutions in parallel and pick the correct one

Circuit Satisfiability



$(x_1, x_2, x_3) = (1,1,0)$

Complexity class relationships



our story so far ...

Polynomial Time Reductions

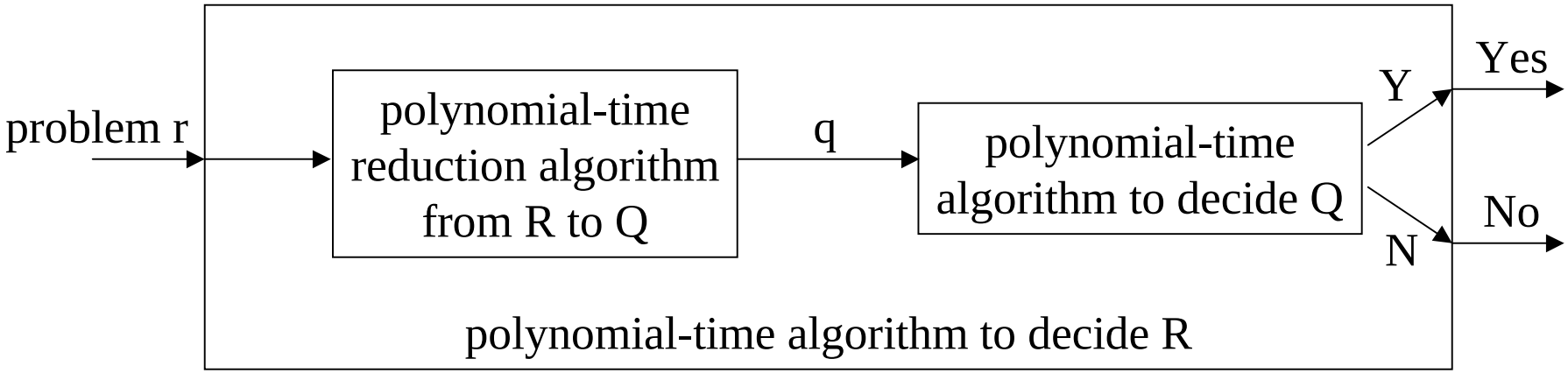
Let R and Q be two problems. We say that R is **polynomially reducible** to Q if there is a polynomial time algorithm that converts each input $r \in R$ to another input $q \in Q$ such that r is a yes-instance of R if and only if q is a yes-instance of Q.

Theorem: If R is polynomially reducible to Q and there is a polynomial time algorithm for Q, then there is a polynomial time algorithm for R.

Reduction Algorithm

- Given a polynomial time reduction from R to Q and a polynomial algorithm to solve a problem from Q, we can solve any problem in R in polynomial time
  - Given an instance *r* of problem R, use the poly-time reduction to transform it to an instance *q* of problem Q
  - Run the poly-time decision algorithm for Q on instance *q*
  - Use the answer for *q* as the answer for *r*

Using a reduction to solve a problem



[CLRS]

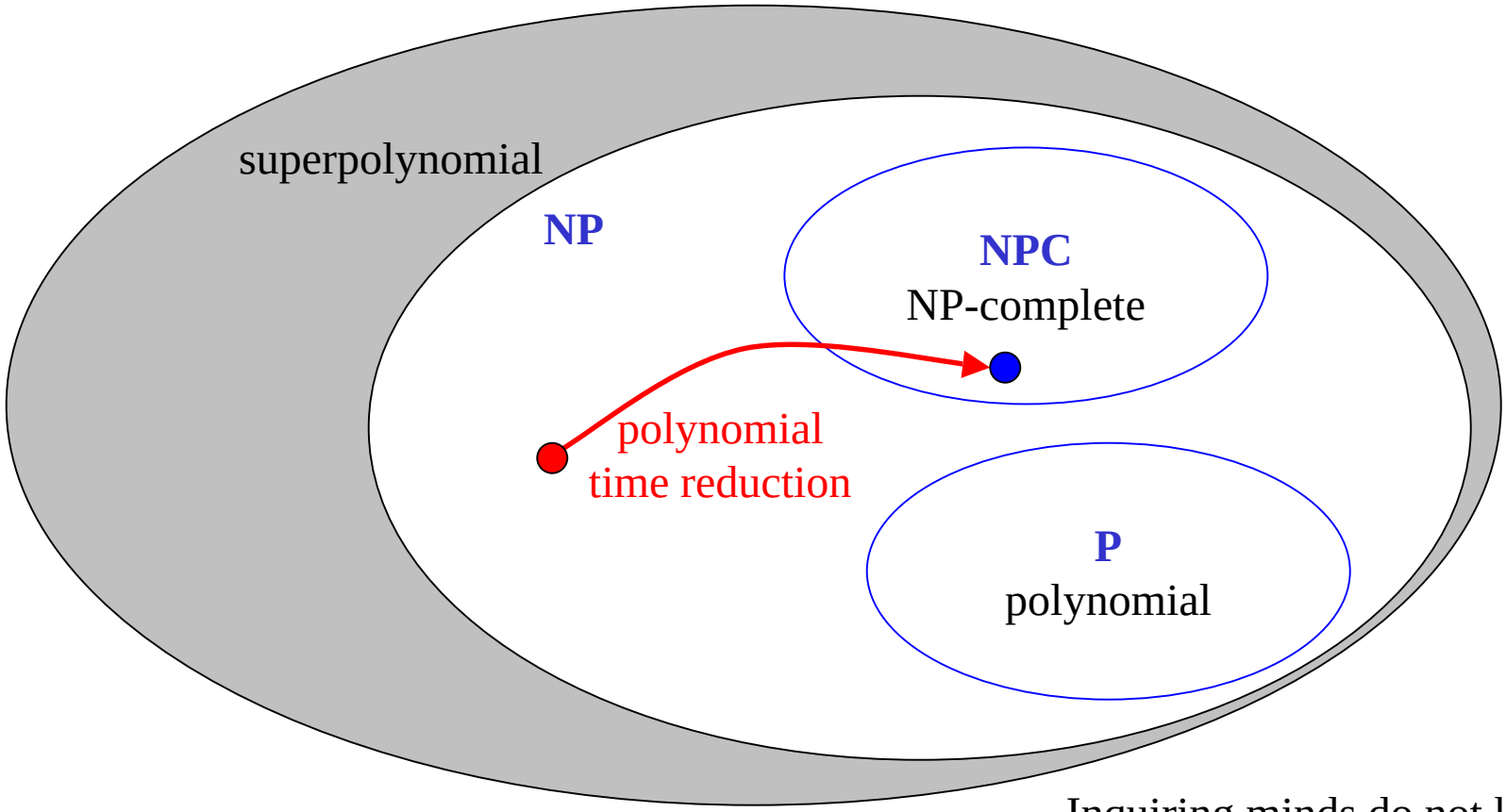
Using reduction on NP problems

- Many of the problems in NP can be converted using a polynomial time reduction from one problem statement to another
  - › using a polynomial time reduction
- All these problems can be considered equivalent from the standpoint of “how hard is it to find a solution?”
  - › solve one in polynomial time and you can solve any of them in polynomial time!

NP-complete

- NP-complete problems are defined to be those that satisfy the following:
  - the problem is in NP, ie, a proposed solution can be checked in polynomial time
  - any other problem in NP can be reduced to this problem in polynomial time, ie, there is a mapping from all other problems in NP to this problem

Are P and NPC distinct?



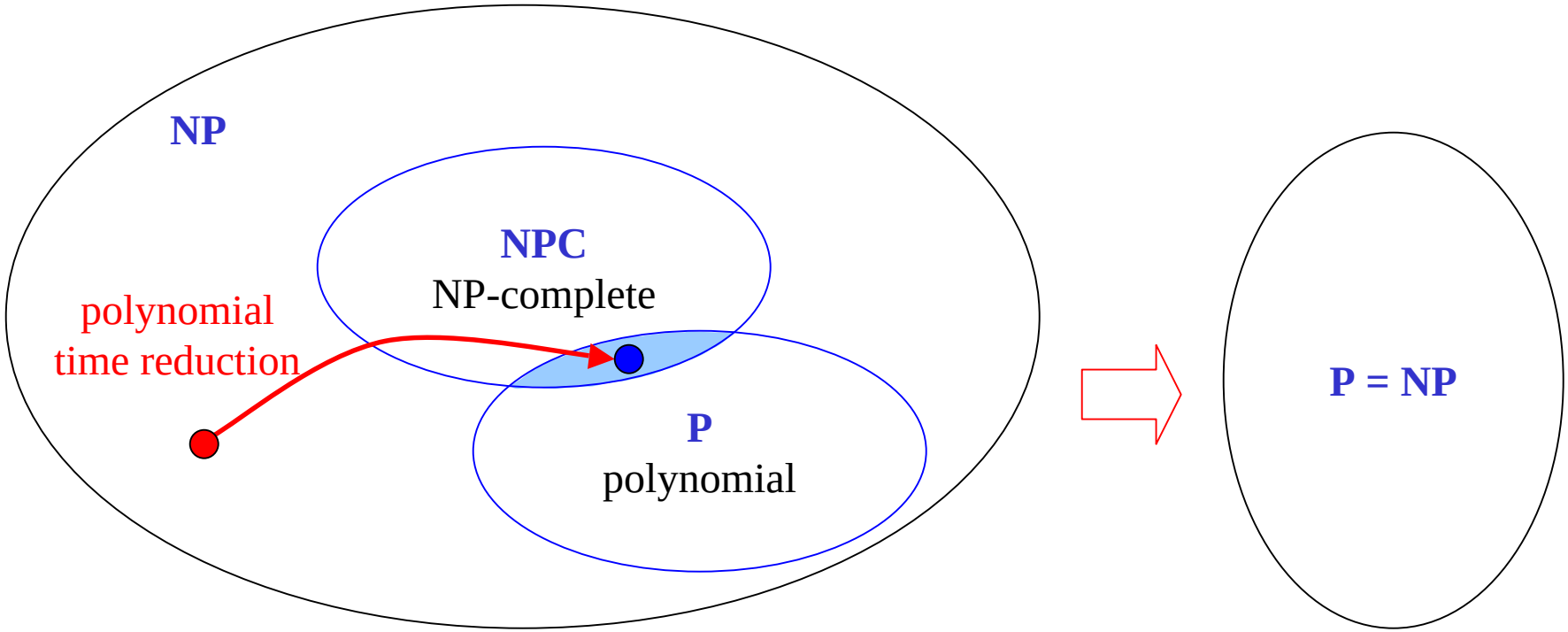
Inquiring minds do not know!

P=NP?

- If *any* NP-complete problem *is proved* polynomial-time solvable, then P=NP and all problems in NP can be solved in polynomial time
- If *any* NP-complete problem *is proved not* polynomial-time solvable, then P∩NPC=∅ and no NP-complete problem can be solved in polynomial time

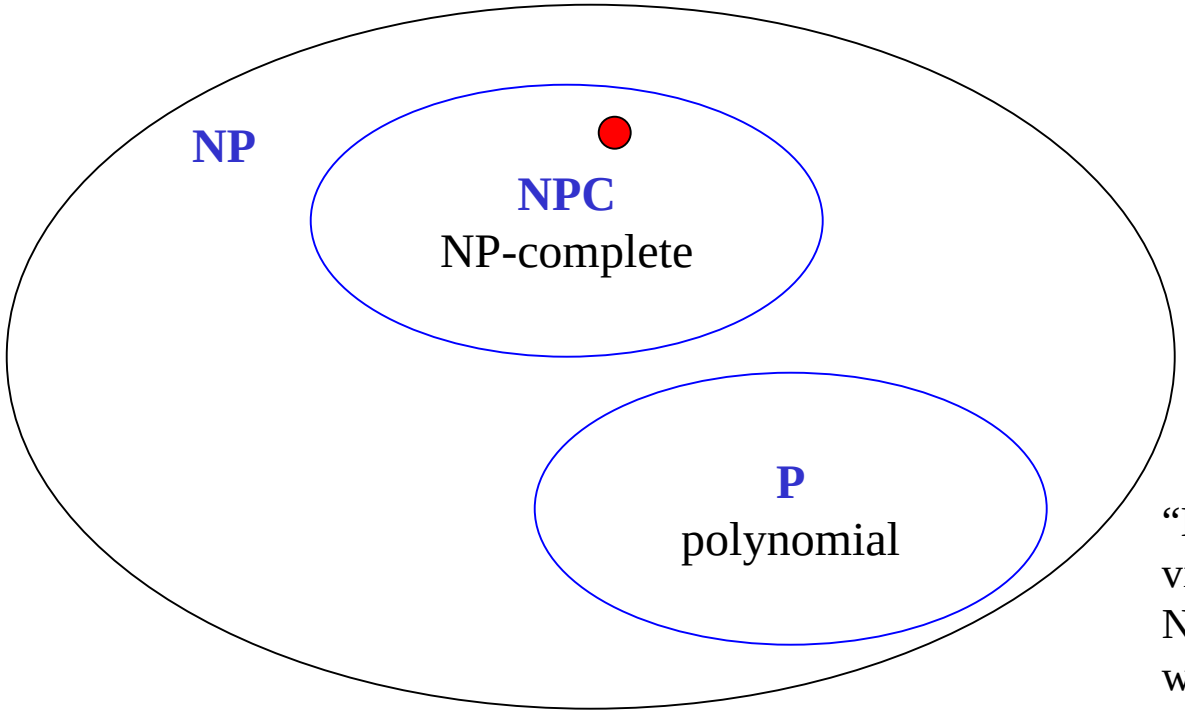
P and NPC intersect? Maybe yes...

If *any* NP-complete problem *is proved* polynomial-time solvable, then P=NP and all problems in NP can be solved in polynomial time



P and NPC intersect? Maybe no ...

If *any* NP-complete problem *is proved not* polynomial-time solvable, then P∩NPC=∅ and no NP-complete problem can be solved in polynomial time



Many people have tried for years to find polynomial time solutions to the many problems in NPC, without success.

"Most theoretical computer scientists view the relationship of P, NP, and NPC this way. Both P and NPC are wholly contained within NP, and P∩NPC=∅" [CLRS]

Why does this matter?

- Assume that P≠NP. Then there are many problems for which we have no hope of finding a polynomial time algorithm
- If we can show that a particular problem is in NP, we should spend our energy on finding heuristics or an approximate solution, rather than trying to find an efficient exact solution

There exists an NP-complete problem

- Steve Cook proved in 1971 that there exists at least one NP-complete problem
- The Satisfiability problem is NP-complete.
- Satisfiability: Given a boolean formula in conjunctive normal form (AND of ORs), is there an assignment of the variables to 0's and 1's so that the resulting formula evaluates to 1?
- Example:

$$(y_1 \vee y_3 \vee y_4) \wedge (\overline{y_1} \vee y_2 \vee \overline{y_4}) \wedge (y_2 \vee \overline{y_3} \vee y_4) \wedge (\overline{y_2} \vee y_3 \vee \overline{y_4})$$

- Since then, hundreds more have been analyzed

What are the problem domains?

- NP-complete problems arise in diverse domains
  - boolean logic, graphs, arithmetic, network design, sets and partitions, storage and retrieval, sequencing and scheduling, mathematical programming, algebra and number theory, games and puzzles, automata and language theory, program optimization, and more

Formula Satisfiability

- Input description: Given a boolean formula in conjunctive normal form
- Problem description: Is there a truth assignment for the variables that causes the formula to evaluate to 1.
- Special case where every clause is disjunction of exactly 3 literals also NP complete (called 3-SAT)
- Example: digital design, hardware testing,….

Traveling Salesman Problem

- Input description: A weighted graph G, L
- Output description: Is there a tour of length at most L that visits each of the vertices exactly once.

- Optimization version: minimize the length of the tour.

Vertex Coloring

- Input description: A graph G=(V,E), k
- Problem description: Is it possible to color the vertices of the graph using at most k colors such that for each edge (i,j) in E, vertices i and j have different colors
- Optimization version: minimize the number of colors used.
- Example: Register allocation for compilers.

Independent Set

- Input description: A graph G=(V,E), k
- Problem description: Is there a subset S of V of size at least k such that no pair of vertices in S has an edge between them.
- Example:
  - Identifying location for a new franchise service such that no two locations are close enough to compete with each other.
  - Highest capacity code for given communication channel.

Hamiltonian Cycle

- Input description: A graph G=(V,E)
- Problem description: Is there an ordering of the vertices such that adjacent vertices in the ordering are connected by an edge and each vertex is visited exactly once.
- Example:

  Triangle strip problem in graphics

Clique

- Input description: A graph G=(V,E), k
- Problem description: Is there a subset S of V of size at least k such that for all x,y in S, (x,y) in E.
- Optimization Version: Find maximum sized subset S.

Graph Partition

- Input description: A weighted graph G=(V,E) and integers j,k
- Problem description: Is there a partition of the vertices into two subsets such that each subset has size at most j, and the weight of edges connecting the two subsets is at most k.
- Example:

  VLSI layout

Essence of NP-completeness

- When we're given a problem we can't solve efficiently, we try to find whether it is equivalent to other problems we can't solve efficiently.
- Hundreds (thousands?) of equivalently hard NP-complete problems of immense practical importance. (scheduling, resource allocation, hardware design and test,……..)

To prove problem Q is NP-complete

1. Prove it's in NP
2. Select a known NP-complete problem R.
3. Describe a polynomial time computable algorithm that computes a function f mapping every instance of R to some instance of Q.
4. Prove that for every yes-instance of R maps to a yes-instance of Q, and every no-instance of R maps to a no-instance of Q.

Coping with NP-completeness

- Given that it is difficult to find fast algorithms for NPC problems, what do we do?
- Alternatives:
  - Dynamic programming: Avoid repeatedly solving the same subproblem – use table to store results (see Chap. 10)
  - Settle for algorithms that are fast on average: Worst case still takes exponential time, but doesn't occur very often
  - Settle for fast algorithms that give near-optimal solutions: In TSP, may not give the cheapest tour, but maybe good enough
  - Try to get a "wimpy exponential" time algorithm: It's okay if running time is $O(1.00001^N)$ – bad only for N > 1,000,000
- Take CSE 417!