Complexity

Slides based on Kevin Wayne / Pearson-Addison Wesley

Decidability

Decision problem

- A problem with a yes / no answer
- E.g. Does graph G have an independent set of size k?
- E.g. Is the maximum flow of graph G greater than k?

Decidability

- A decision problem is decidable if there exists a Turing machine that halts on all inputs and accepts exactly the inputs for which the answer to the decision problem is "yes"
  - Either accepts or rejects, but doesn't loop
- Equivalently, there exists an algorithm to solve the problem

- How do we know if there is a "good" algorithm to solve the problem?

Complexity

What should be the cutoff between a "good" algorithm and a "bad" algorithm?

- Proposal:
  - A "good" algorithm is one whose running time is a polynomial function of the size of the input
  - Other algorithms are "bad"
- This definition was adopted:
  - A problem is called tractable if there exists a "good" (polynomial-time) algorithm that solves it.
  - A problem is called intractable otherwise.

Complexity

- Is this a valid cutoff?

Assuming you can process 1 million computations per second

Time complexity function	Size n 10	20	30	40	50	60
n	.00001 second	.00002 second	.00003 second	.00004 second	.00005 second	.00006 second
n^2	.0001 second	.0004 second	.0009 second	.0016 second	.0025 second	.0036 second
n^3	.001 second	.008 second	.027 second	.064 second	.125 second	.216 second
n^5	.1 second	3.2 seconds	24.3 seconds	1.7 minutes	5.2 minutes	13.0 minutes
2^n	.001 second	1.0 second	17.9 minutes	12.7 days	35.7 years	366 centuries
3^n	.059 second	58 minutes	6.5 years	3855 centuries	2×10^8 centuries	1.3×10^{13} centuries

The Class P

The Class P contains all decision problems that are decidable by an algorithm that runs (on a deterministic Turing machine) in polynomial time in the size of the input

- Note – many of these problems are described as functional optimization problems
  - We'll discuss later how we can cast them as decision problems

- Greedy
  - Huffman: $O(n \log n)$, Interval scheduling: $O(n \log n)$
- Divide-and-Conquer
  - Mergesort: $O(n \log n)$, Select: $O(n)$
- Dynamic Programming
  - Floyd-Warshall: $O(n^3)$, Matrix Multiplication: $O(n^3)$
- Network Flow
  - Edmonds-Karp: $O(m^2n)$

Polynomial-Time Reductions

We've seen some examples already of polynomial-time reductions of one problem to another

- Using max-flow to solve maximum bipartite matching

- Weighted interval scheduling (with weights of 1) to solve regular interval scheduling

Polynomial-Time Reductions

Basic Idea. Suppose we could solve Y in polynomial time. What else could we solve in polynomial time?

Reduction. Problem X polynomial reduces to problem Y if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to black box that solves problem Y.

Notation. $X \leq_P Y$.

Polynomial-Time Reduction

Purpose. Classify problems according to relative difficulty.

Design algorithms. If $X \leq_P Y$ and Y can be solved in polynomial time, then X can also be solved in polynomial time. Why?

Establish intractability. If $X \leq_P Y$ and X cannot be solved in polynomial time, then Y cannot be solved in polynomial time.

Establish equivalence. If $X \leq_P Y$ and $Y \leq_P X$, we use notation $X \equiv_P Y$.

up to cost of reduction

Does Transitivity hold for polynomial-time reduction?

If $X \leq_P Y$ and $Y \leq_P Z$, then $X \leq_P Z$.

YES!

The Class NP

For many problems, no polynomial-time algorithm has been discovered yet to solve the problem.

- Independent Set: *given graph G and number k, does there exist an independent set of size ≥ k?*
- Hamiltonian Cycle: *given directed graph G, does G contain a Hamiltonian Cycle, that is a cycle that visits all nodes without repeating?*

For many such problems, it is possible to check if a given solution is correct in polynomial time

- Independent Set: *given a set of k nodes, check graph G to see if it has any edge that has both endpoints among the k nodes.*
- Hamiltonian Cycle: *given a solution cycle, check graph G to confirm that edges exist for each step along the path.*

The Class NP

The class NP contains all decision problems that have polynomial-time verifiers

- The verifier is an algorithm that takes as input the encoding of the problem along with a proposed solution, and verifies the proposed solution in polynomial time
  - The proposed solution is also known as:
    - Certificate
    - Witness
  - The verification algorithm is also known as:
    - Certifier

- Is P $\subseteq$ NP? **YES!**

The Class NP

NP stands for non-deterministic polynomial time

- An alternative characterization of NP problems
- A decision problem is in NP if it can be solved in non-deterministic polynomial time
  - Imagine an algorithm that can simultaneously follow all branches of computation
  - E.g. Solve Hamiltonian cycle by following all possible paths out of a node simultaneously at each step.
  - E.g. or solve SAT by simultaneously following all branches of assignment (0 or 1) to each variable

P = NP?

Big open question in complexity theory: Does P = NP?

- There are many problems that have eluded polynomial-time solutions.
- Yet no one has been able to prove that there is any problem in NP that does not belong to P

- Most people believe that P ≠ NP
  - Too unlikely that there could be a general transformation from the task of checking a solution to that of finding a solution.
  - A huge amount of unsuccessful effort has gone into finding polynomial-time algorithms for many NP problems

Revisiting Polynomial-Time Reductions

In the absence of a solution to the P=NP question:

- We can identify the most "difficult" of the problems in NP

- An NP problem Y is classified as "difficult" if all other problems in NP can be polynomial reduced to Y

- What would a polynomial-time solution to a "difficult" problem imply?
  - A polynomial time solution to all NP problems!
  - This would show P = NP!

NP-Hard and NP-Complete

The class NP-Complete contains all decision problems Y that are in NP and for which every other problem in NP can be polynomial reduced to Y

- Y is an NP-Complete problem if:
  - $Y \in NP$
  - For all $X \in NP$, $X \leq_P Y$

The class NP-Hard contains all decision problems Y for which every other problem in NP can be polynomial reduced to Y

- Y is an NP-Complete problem if:
  - For all $X \in NP$, $X \leq_P Y$

- There are problems that are in NP-Hard but that are not NP-Complete. This implies that they have no polynomial-time verifier.

NP-Hard and NP-Complete

What is the value of NP-Complete?

- If a polynomial-time algorithm is found for any problem Y in NP-Complete:
  - Since for all $X \in NP$, $X \leq_P Y$, this implies that every problem in NP has a polynomial-time algorithm
  - There's no need to try to solve all NP-Complete problems
  - If a new problem is shown to be NP-Complete, it's probably not worth spending much time trying to find an efficient algorithm for solving it

NP-Hard and NP-Complete

How to demonstrate that a new problem, Z, is NP-Complete?

- First show that Z is in NP
  - Find a "witness" (a solution) of polynomial size
  - Describe how to verify the solution in polynomial time

- Then find another problem in NP-Complete that reduces to Z
  - Find Y in NP-Complete such that $Y \leq_P Z$

  - Why does this show that Z is NP-Complete?
    - Transitivity: $X \leq_P Y$ for all X in NP, so
    - $X \leq_P Y \leq_P Z$ and thus any problem in NP can be reduced to Z in polynomial time

NP-Hard and NP-Complete

How to get started – how to find the first NP-Complete problem

- Once we have a problem in NP-Complete, it's straight-forward to extend the class NP-Complete
- But how to find a first NP-Complete problem?
  - It requires showing that every other problem in NP reduces to it

NP-Hard and NP-Complete

How to get started – how to find the first NP-Complete problem

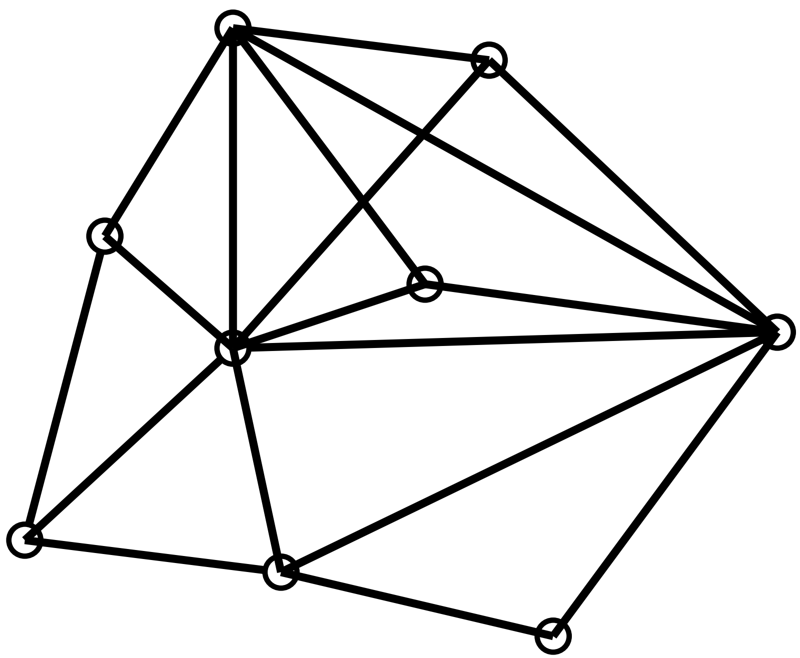
- Cook-Levin Theorem
  - CNF-SAT is NP-Complete
  - Conjunctive Normal Form Boolean Satisfiability
    - Given a collection of boolean variables $x_1, x_2, \ldots, x_n$
    - Given a boolean expression in $x_1, x_2, \ldots, x_n$
    - CNF means logical "AND" of a collection of terms
    - Each term is a logical "OR" of some variables

    - CNF-SAT Problem: Determine if there is an assignment of values to $x_1, x_2, \ldots, x_n$ that satisfies the boolean expression

Examples of NP-Complete Problems

CLIQUE

- Given graph $G = (V,E)$ and a number k, does there exist a clique of size k? That is a set of vertices $S \subset V$ such that for every $u,v \in S$, the edge $(u,v) \in E$?
- Can we show that CLIQUE $\in$ NP?
- Can we show CNF-SAT $\leq_P$ CLIQUE?



Examples of NP-Complete Problems

CLIQUE

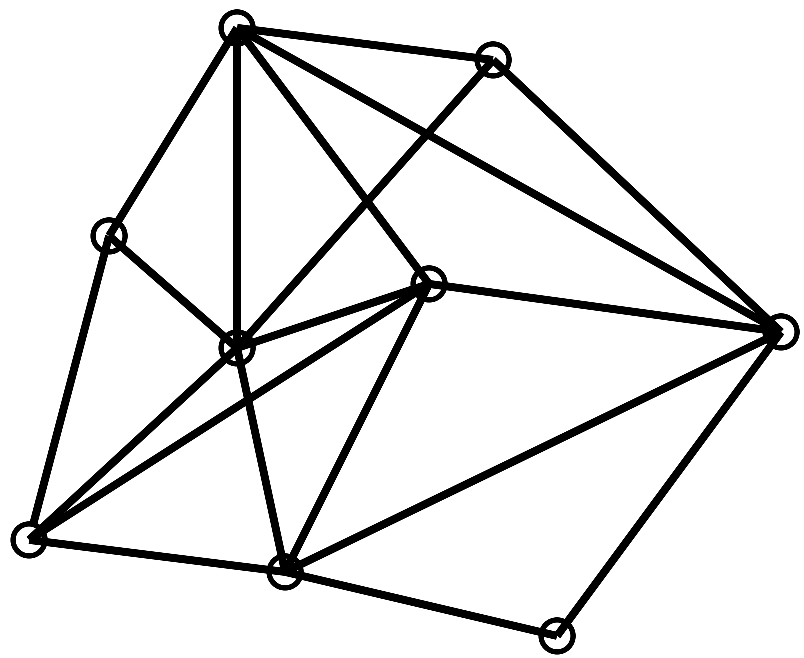
- Note that just because a problem is NP-complete doesn't mean that all instances of it take exponential time.

- Given a graph G, and a number k, does there exist a clique of size less than or equal to k?
  - If you have a clique of size k, you automatically have cliques smaller than that. All you have to do is check for a clique of size 2. That just requires one edge! (or you could argue it's automatically true for k=1 as long as you have a node)
- Given a graph G, does there exist a clique of size 5?
  - Given n nodes, there are $O(n^5)$ different cliques possible of size 5. Just check them all. Polynomial-time.
- Given a graph G, does there exist a clique of size n-5?
  - Given n nodes, there are $O(n^5)$ different cliques possible of size n-5. Just check them all. Polynomial-time.

Examples of NP-Complete Problems

INDEPENDENT SET

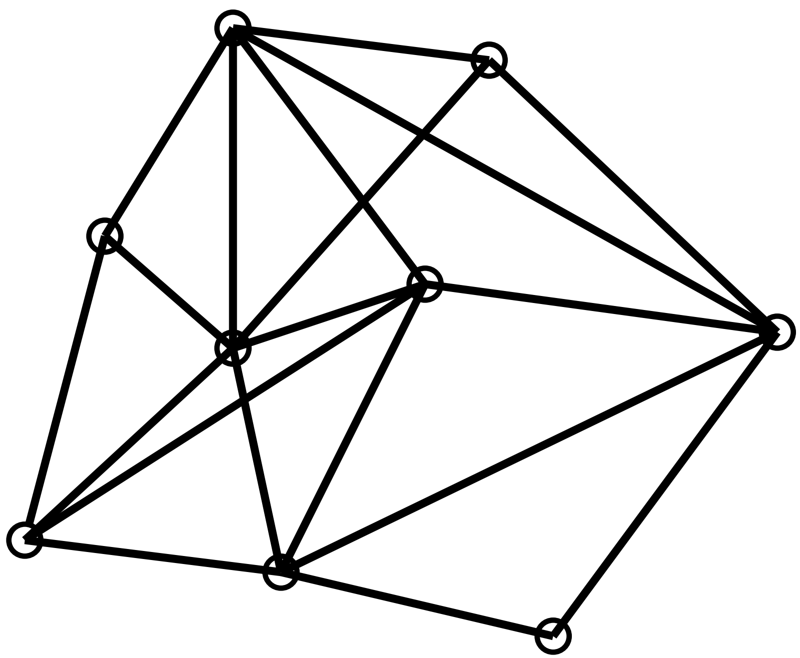
- Given graph $G = (V,E)$ and a number k, does there exist an independent set of size k? That is a set of vertices $S \subset V$ such that for every $u,v \in S$, the edge $(u,v) \notin E$?
- Can we show that INDEPENDENT SET $\in$ NP?
- Can we reduce an existing NP-Complete problem to IND. SET?



Examples of NP-Complete Problems

VERTEX COVER

- Given a graph $G = (V, E)$ and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| = k$, and for each edge, at least one of its endpoints is in S?
- Can we show that VERTEX COVER $\in$ NP?
- Can we reduce an existing NP-Complete problem to VERTEX COVER?



Proving that a Problem Belongs to NP-Complete

Finding the right problem to reduce from

- CNF-SAT $\leq_P$ CLIQUE $\leq_P$ INDEPENDENT SET $\leq_P$ VERTEX COVER

Co-NP

For every decision problem X, there is a natural complementary decision problem X'.

- E.g. Given graph G = (V,E) and a number k, does there NOT exist a clique of size k?

- While problems in NP can be verified easily, the complementary problems aren't verified easily
  - How would you verify that there is NO clique of size k?

The class Co-NP contains all decision problems whose complement belongs to NP.

- Open question: does NP = Co-NP?
  - Prevailing belief is no
- Another open question: is P = NP ∩ Co-NP?
  - Are the only problems in NP, for which the complementary problem can be checked easily, problems that are polynomial to begin with?

More NP-Complete Examples

HAMILTONIAN CYCLE

- Given an undirected graph $G = (V, E)$, does there exist a simple cycle C that visits every node?
- Can we show that HAMILTONIAN CYCLE $\in$ NP?
  - Certificate: a permutation of the vertices
  - Verifier: checks that edges exist for each pair of vertices along the proposed cycle

- It can be shown that 3-SAT $\leq_P$ HAMILTONIAN CYCLE
  - (3-SAT is a variant of SAT in which each clause has exactly 3 variables. 3-SAT is also NP-Complete.)

More NP-Complete Examples

TRAVELING SALESMAN PROBLEM

- Given a complete weighted graph $G = (V, V \times V)$, with weights w, and a threshold t, does there exist a simple cycle C that visits every node exactly once and has a total weight $< t$?
- Can we show that TRAVELING SALESMAN PROBLEM $\in$ NP?
  - Certificate: a permutation of the vertices
  - Verifier: checks that total weight of corresponding edges is less than threshold t

- HAMILTONIAN CYCLE $\leq_P$ TRAVELING SALESMAN PROBLEM
  - Thus TSP is NP-Complete
  - This is a Question on the final homework!

More NP-Complete Examples

3-COLORING

- Given a graph $G = (V, E)$, is it possible to color the vertices of G with 3 colors so that no edge has endpoints both the same color?
- Can we show that 3-COLORING $\in$ NP?
  - Certificate: a coloring of the vertices
  - Verifier: checks that each edge has endpoints of different color

- 3-SAT $\leq_P$ 3-COLORING (proof in book)
  - Thus 3-COLORING is NP-Complete

More NP-Complete Examples

k-COLORING

- Given a graph $G = (V, E)$, is it possible to color the vertices of G with k colors so that no edge has endpoints both the same color?
  - 4-COLORING $\in$ NP-Complete
  - 3-COLORING $\leq_P$ 4-COLORING
  - More generally, 3-COLORING $\leq_P$ k-COLORING for $k > 3$

More NP-Complete Examples

k-COLORING

- Given a graph $G = (V, E)$, is it possible to color the vertices of G with k colors so that no edge has endpoints both the same color?
  - 2-COLORING: what kind of graph is required for a 2-coloring to exist?
    - Bipartite graph
    - We can decide the 2-coloring problem in $O(V + E)$ time
      - 2-COLORING $\in$ P
      - (Do a BFS then confirm that there are no edges between vertices at the same depth.)

More NP-Complete Examples

KNAPSACK PROBLEM

- Given n items each having associated cost and weight, what is the maximum cost of items that can be placed in a knapsack of capacity W?
  - Can solve using Dynamic Programming in O(nW)
  - But KNAPSACK is NP-Complete
  - Why?
    - ✐ O(nW) is not polynomial in the length (number of bits) of the input W

Decision Problem versus Functional Optimization

NP-Completeness is defined for decision problems

- Problems with a Yes / No answer
- E.g. Does graph G have a max flow equal to k

Often, however, we are presented with functional optimization problems

- E.g. Determine the max flow for a graph

Although the optimization problem is at least as hard as the decision problem, in most cases we can show that they are equivalent with respect to polynomial-time solvability

Decision Problem versus Functional Optimization

Example
- Find the longest simple path in undirected graph G (optimization)
- Does undirected graph G have a longest simple path of length k

Often the optimization problem can be solved via a polynomial number of decision problems
- For the longest path problem, at most n decision problems must be solved

It is often easy to solve a decision problem given the solution to the functional optimization problem
- Simply apply the optimization solution to the particular decision problem

- Thus the optimization and decision problem are often equivalent with respect to polynomial-time solvability

Decision Problem versus Functional Optimization

More Examples

- What is the largest size clique in graph G
  - How many decision problems?

- Sorting as a decision problem:
  - Does element a_j end up at position p in the sorted array?
  - How many decision problems needed to know the sorted array?

- Max flow as a decision problem:
  - Does graph G have a maximum flow equal to k?
  - How many decision problems to find max flow?
  - Is this a polynomial number in the size of the input C?
  - Instead:
    - Is the i^{th} bit of (binary representation of) the max flow a 1?
    - Now the number of decision problems is linear in the size of the input C!

Decision Problem versus Construction

NP-Completeness is defined for decision problems

- Problems with a Yes / No answer
- E.g. Does graph G have a clique of size k?

Often, however, we would like to construct the solution, not just know that one exists

- E.g. What is a size k clique of G?

Although the construction problem is at least as hard as the decision problem, in most cases we can show that they are equivalent with respect to polynomial-time solvability

Decision Problem versus Construction

Examples

- What is a clique of size k in graph G?
  - First, make call to decision problem to confirm a solution exists
  - Next, make calls to decision problem with individual vertices removed one by one if they are not necessary to maintain a clique of size k
    - ✐ O(|V|) total number of calls to the decision problem to determine which vertices are part of a clique of size k

- What is an assignment of variables $x_1,x_2, \ldots, x_n$ that satisfies a given SAT expression?
  - First, make call to decision problem to confirm a solution exists
  - Next, make calls to decision problem with individual variables hardcoded to value 0 or 1. At least one of the two must admit a solution. Continue along a path having a solution until all variables have value assigned.
    - ✐ O(n) total number of calls to the decision problem