Review: **P** and **NP**

- *What do we mean when we say a problem is in **P**?*
  - A: A solution can be found in polynomial time
- *What do we mean when we say a problem is in **NP**?*
  - A: A proposed solution can be verified in polynomial time
- *What is the relation between **P** and **NP**?*
  - A: **P** $\subseteq$ **NP**, but no one knows whether **P = NP**

Review: NP-Complete

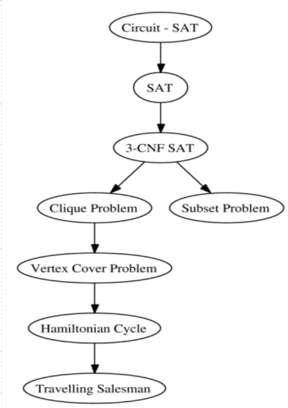
- *What, intuitively, does it mean if we can reduce problem A to problem Q?*
  - A is "no harder than" Q
- *How do we reduce A to Q?*
  - R transforms, in polynomial time, instances in language A into instances in language Q, i.e. a$\in$A iff R(a)$\in$Q
- *What does it mean if Q is NP-Hard?*
  - Every problem A$\in$**NP** can be reduced to Q in polynomial-time
- *What does it mean if Q is NP-Complete?*
  - Q is NP-Hard and Q $\in$ **NP**

Review: Proving Problems NP-Complete

- *How do we usually prove that a problem Q is NP-Complete?*
  - A: Show Q $\in$**NP**, and reduce a known NP-Complete problem *A* to Q

NP-Complete

- Reducibility of some NP-complete problems
- Therefore, these decision problems are NP-hard (NP-complete since ...)
- <u>Decision problems are reducible, but not the optimization problem</u>
- All NP-complete problems are reducible to each other, by definition



Review: Tractable vs. Intractable

- All problems are a decision about whether or not a guess is a valid solution
  - **Tractable (feasible) problems**:
    - a valid guess can be deterministically generated in polynomial time, i.e., the problems in complexity class P
  - **Undecidable problems**:
    - there can be no algorithm to validate a guess
    - must be proven mathematically (e.g., the halting problem)
  - **Intractable (infeasible) problems**:
    - no polynomial time algorithm to deterministically generate a valid guess has <u>yet</u> been found
    - NP-Complete and NP-Hard problems are considered intractable, <u>but we are not sure</u>
    - Includes problems in NP and others not in NP
- There are three categories:
  - Easy (P, tractable), hard (NPH, NPC, intractable), and non-computable (NPH, undecidable)

General Comments

- Literally many hundreds of problems have been shown to be NP-Complete
  - Some reductions are profound,
  - Some are comparatively easy,
  - Many are easy once the key insight is known
- You can expect a simple reduction or NP-Completeness proof on the final

Some NP-Complete Problems

- *0-1 knapsack*: when weights are not just integers
- *Subset-sum*: Given a set of integers, does there exist a subset that adds up to some target *T*?
- *Hamiltonian path*: Given a graph G, is there a path that visits each vertex exactly once?
- *Hamiltonian circuit*: Given a graph G, is there a cycle that visits each vertex exactly once?
- *TSP*: Given a list of cities and their pair-wise distances, is there a tour that visits each city exactly once with total distance at most D?
- *Graph coloring*: Can a given graph be colored with *k* colors such that no adjacent vertices are the same color?
- *Vertex cover*: Given a graph G, is there a set C of vertices with size K such that each edge is incident to at least one vertex in C?
- Register allocation in compilers, type inference in programming languages,
  - *n* is small enough that brute force or approximation algorithms are useful for solving most practical instances of these problems (i.e., most practical programs)
- Etc…

More Graph Problems Which are in NPC?

- Longest Path
  - Given a weighted graph G=(V, E), two vertices *u*, *v* ∈ V, and a positive number K. Is there a simple path between *u* and *v* with total weight at least K?
- Minimum Degree Spanning Tree
  - Given graph G=(V,E) and positive integer K. Is there a spanning tree T =(V,E') such that the maximum degree of any vertex in T is at most K?
- Shortest Total Path Length Spanning Tree
  - Given graph G=(V,E) and positive integer K. Is there a spanning tree T =(V,E') such that the length of the path in T between every pair of vertices u,v∈V is at most K?
- K-minimum Spanning Tree
  - Given graph G=(V,E), positive integer K ≤ |V|, and positive weight W. Is there a tree that spans K vertices with total weight ≤ W?

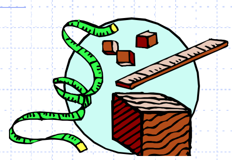
What about?

- Euler Tour
  - Given a weighted graph G=(V, E). Is there a path that visits each edge exactly once?
  - This problem is in P. Yes if all but zero or two vertices have a degree that is an even number.

How to deal with NP-complete optimization problems?

- Apply an approximation algorithm.
  - Typically faster than an exact solution.
  - Assuming the problem has a large number of feasible solutions.
    - Also, has a cost function for the solutions.
    - Want to find a solution with minimum cost in a reasonable time (i.e. polynomial time).
- Apply Heuristic solution
  - Looking for “good enough” solutions.

Approximation Algorithms



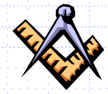
Outline and Reading

- Approximation Algorithms for NP-Complete Problems (§13.4)
  - Approximation ratios
  - Polynomial-Time Approximation Schemes (§13.4.1)
  - 2-Approximation for TSP special case (§13.4.3)
  - 2-Approximation for Vertex Cover (§13.4.2)
  - Log n-Approximation for Set Cover (§13.4.4)

Approximation Ratios

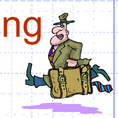
- Optimization Problems
  - We have some problem instance x that has many feasible “solutions”.
  - We are trying to minimize (or maximize) some cost function c(S) for a “solution” S to x. For example,
    - Finding a minimum spanning tree of a graph
    - Finding a smallest vertex cover of a graph
    - Finding a smallest traveling salesperson tour in a graph
- An approximation produces a solution T
  - T is a **k-approximation** to the optimal solution OPT if c(T)/c(OPT) $\le$ k (assuming a min. prob.)
  - a maximization approximation would be the reverse ($\ge$)

Polynomial-Time Approximation Schemes

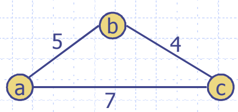


- A problem L has a **polynomial-time approximation scheme (PTAS)** if it has a polynomial-time ($1+\varepsilon$)-approximation algorithm, for any fixed $\varepsilon > 0$ (this value can appear in the running time).
- 0/1 Knapsack has a PTAS, with a running time that is $O(n^3/\varepsilon)$. See §13.4.1 in Goodrich-Tamassia for details.

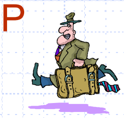
Special Case of the Traveling Salesperson Problem

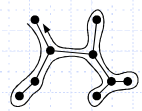


- OPT-TSP: Given a complete, weighted graph, find a cycle of minimum cost that visits each vertex.
  - OPT-TSP is NP-hard
  - Special case: edge weights satisfy the triangle inequality (which is common in many applications):
    - w(a,b) + w(b,c) $\ge$ w(a,c)

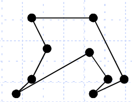


A 2-Approximation for TSP Special Case





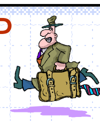
Euler tour *P* of MST *M*



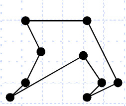
Output tour *T*

Algorithm *TSPApprox*(*G*)
  Input weighted complete graph *G*, satisfying the triangle inequality
  Output a TSP tour *T* for *G*
  M ← a minimum spanning tree for *G*
  P ← an Euler tour traversal of *M*, starting at some vertex *s*
  T ← empty list
  for each vertex *v* in *P* (in traversal order)
    if this is *v*’s first appearance in *P* **then**
      T.insertLast(*v*)
  T.insertLast(*s*)
  return *T*

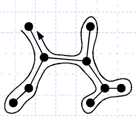
A 2-Approximation for TSP Special Case - Proof



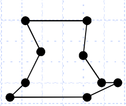
- The optimal tour is a spanning tour; hence |M|$\le$|OPT|.
- The Euler tour P visits each edge of M twice; hence |P|=2|M|
- Each time we shortcut a vertex in the Euler Tour we will not increase the total length, by triangle inequality (w(a,b) + w(b,c) $\ge$ w(a,c)); hence, |T|$\le$|P|.
- Therefore, |T|$\le$|P|=2|M|$\le$2|OPT|



Output tour *T* (at most the cost of *P*)

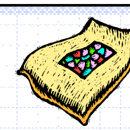


Euler tour *P* of MST *M* (twice the cost of *M*)

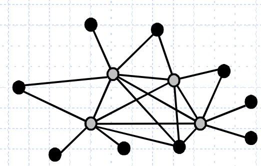


Optimal tour *OPT* (at least the cost of MST *M*)

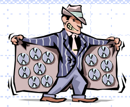
Vertex Cover



- A **vertex cover** of graph G=(V,E) is a subset W of V, such that, for every (a,b) in E, a is in W or b is in W.
- OPT-VERTEX-COVER: Given an graph G, find a vertex cover of G with smallest size.
- OPT-VERTEX-COVER is NP-hard.
- There is 2-approximation scheme for Vertex Cover (see text)

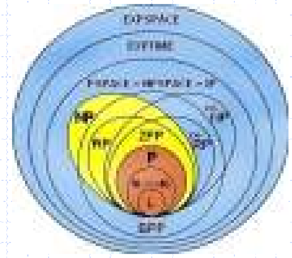


Set Cover



- OPT-SET-COVER: Given a collection of m sets, find the smallest number of them whose union is the same as the whole collection of m sets?
  - OPT-SET-COVER is NP-hard
- Greedy approach produces an O(log n)-approximation algorithm. See §13.4.4 for details.

Additional Complexity Classes of Computable Problems



From MIT Web Page for Theory of Computation Course

Complexity classes L and NL

- L is the class of decision problems that can be solved using logarithmic space
- NL is the class of decision problems that can be solved non-deterministically using logarithmic space
- $L \subseteq NL \subseteq P$
- Open question: Is L=NL=P?

Probabilistic (Randomized) Algorithms

- Algorithms that use some degree of randomness as part of their logical structure
- Examples:
  - Quicksort, Quickselect, Skip List
  - Non-deterministic Algorithms

Verifier

- Definition:
  - A *verifier* for a language L is an algorithm V such that
    - If $x \in L$, then there exists a string w such that V(x,w)=yes
    - If $x \notin L$, then for all strings w, V(x,w)=no
  - If V(x,w)=yes, then w is called a *witness* or a *certificate* (or guess) that verifies that $x \in L$
  - Note that the no answer is based on the collection of all strings, whereas the yes answer is based on the existence of one string w
    - This is what helped me understand the difference between NP and Co-NP
- In the complexity classes of interest, all of the verifiers must run in polynomial time

Complexity Class NP

- $A \in NP$: Non-deterministic polynomial time
  - If there exists a verifier V for language A that runs in polynomial time
    - w is randomly (non-deterministically) generated
    - If $x \notin A$, then V(x,w) always returns no
    - If $x \in A$, then V(x,w) eventually returns yes
      - V keeps returning no until a certificate/witness w is found
      - i.e., if there exists a string w (guess) that verifies that the answer is yes, then w will eventually be the guess
  - Note that the string w (guess) could be a proof that the answer is yes; but the length of the proof must be polynomial in the input string size |x| (a requirement of guesses)

Complexity Classes

- Co-NP
  - Problem A∈Co-NP if and only if the complement of A∈NP
    - There exists a verifier V that runs in polynomial time
    - w is randomly (non-deterministically) generated
    - If x ∈ A, then V(x,w) always returns no
    - If x ∉ A, then V(x,w) eventually returns yes if there exists a string w (guess) that verifies that x ∉ A
      - (i.e., V keeps returning no until a certificate/witness w is found that verifies that x is in the complement of L)
  - Note that the guess (or proof) w verifies that the instance x is not a member of language A, i.e., that x is in the complement of A
  - Thus w could be thought of as a counter example showing that x cannot be in A

Complexity Classes

- Clearly P ⊆ NP and P ⊆ Co-NP
  - Since we can verify whether the answer is either yes or no in polynomial time no matter what the string w is
  - (e.g., for problems in P, we can ignore w and always compute the correct yes/no answer in polynomial time)

Complexity Classes

- RP: Randomized polynomial time
  - Verifier V runs in polynomial time
    - If answer is no, V(x,w) always returns no
    - If answer is yes, V(x,w) returns yes with probability 1/2 (if run *m* times, then probability of getting at least one yes is $1–1/2^m$)
  - Intuitively: If the answer is yes, then the algorithm answers yes half the time (or better) on average
    - (perhaps through some polynomial time algorithm that can produce better guesses than required by an NP algorithm)
- ZPP: Zero-error probabilistic polynomial time
  - Verifier runs in polynomial time
    - Returns yes / no / do-not-know
    - If answer is yes, returns yes with probability $\geq$ 1/2 (or returns do-not-know)
    - If answer is no, returns no with probability $\geq$ 1/2 (or returns do-not-know)
  - ZPP = RP ∩ Co-RP

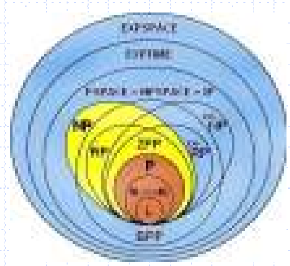
Complexity Classes

- BPP: Bounded-error probabilistic polynomial time
  - Verifier runs in polynomial time
    - If answer is yes, returns yes with probability $\geq$ 2/3
    - If answer is no, returns no with probability $\geq$ 2/3
  - These algorithms can be practical in the sense that we can run them several times, then the answer will be (with high probability) the answer we got the majority of times (the more runs, the higher the probability we have the right answer)
  - Thus this is the largest class that has an efficient probabilistic algorithm (one that would be practical to run on a computer), so BPP is one of the largest practical complexity classes
  - Clearly **BPP=Co-BPP**
  - **Unsolved problem:** Is P = BPP?

Summary

- NP only requires the existence of a witness/certificate/guess that verifies membership
  - Which could take exponential time to find
- RP and ZPP require that there be lots of witnesses (over half of the guesses produce a witness)
- BPP does not require witnesses, although a witness is sufficient to prove membership
  - Instead, the verification algorithm only has to return the right answer more often than the wrong answer (2/3 of the time)

Additional Complexity Classes of Computable Problems



From MIT Web Page for Theory of Computation Course