CS483 Analysis of Algorithms
Lecture 08 – NP-completeness

Jyh-Ming Lien

July 17, 2017

Hard Problems

- Search problems (formal definition later):
  - Search for shortest path in a graph
  - Search for values of x, y and z to satisfy: $x^2 y^{\frac{1}{2}} z - xz^2 = 7$
  - Search for a path in 3D space among obstacles
  - ...

- Success on solving these hard problems in polynomial time
  - **Greedy** properties: without looking backward or forward, MST, shortest paths
  - Optimality from **subproblems**: divide and conquer, dynamic programming
  - **Convexity**: gradient decent/hill climbing, linear programming
- Success on solving these hard problems is based on some **special properties** of the problems

- Failures
  - Many problems require 2^n, $n!$ or even n^n (intractable)
  - For some (clearly formalized) problems we don't even have algorithms to solve them
  - (not to mention those problems that we can not even formalize)
- In this lecture, we will look at
  - What are these *problems*? (**Search Problems**)
  - Terminology to classify *problems*: **P vs. NP**
  - How do you know if a *problem* can be solved efficiently? (**Problem Reduction**)
  - Do we have any hope of solving these *problems* efficiently?

Search Problems

What Are Search Problems?

- A Search problem has
  - An instance of problem I that is input data specifying the problem
  - Asked to find a solution S that meets a particular specification
  - **Polynomial-time Checkable**: There most be an algorithm C that takes I and S and checks for correctness *efficiently*, i.e., in polynomial time
- Example: Satisfability problem
  - $s : (x \vee y \vee z)(x \vee \bar{y})$
  - $t : \{x = \mathrm{T}, y = \mathrm{F}, z = \mathrm{T}\}$
  - $B(s,t) : (T \vee F \vee T)(T \vee \bar{F})$
- Example: Traveling salesman problem
  - $s : G = \{V, E\}$
  - $t : \{v_i, v_k, \cdots, v_i\}$
  - $B(s,t) :$

Optimization Problems

- We convert an optimization problem to a search problem
  - by introduce a **budge** b
  - Budget does not make the problem harder or easier.
- Example: Traveling salesman problem with budget b
  - $s : G = \{V, E\}, b$
  - $t : \{v_i, v_k, \cdots, v_i\}$
  - $B(s, t)$:
- Why do we convert an optimization problem to a search problem?

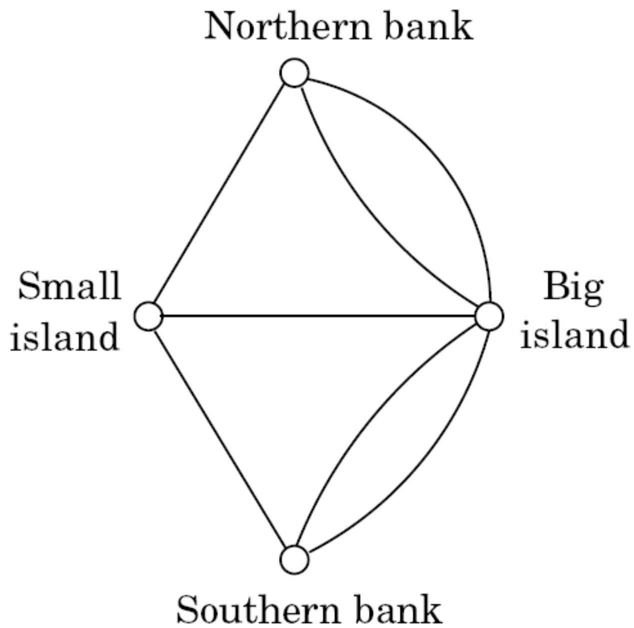
Search Problems

- Many problems we studied in the previous chapters are search problems, e.g., all-pairs shortest paths problem, single-source shortest paths problem, minimum spanning tree, maximum flow minimum cut, matching, ...
  - But why are these problem tractable? These problems seem to have *Very Large* search spaces
  - Many algorithms seem to defeat the curse of expoentiality!
- Now, after we have seen the most brilliant successes, it's about time for us to face some failure in this quest.
  - Satisfability (SAT, 2SAT, 3SAT)
  - MST and TSP (traveling salesman problem) with or without budget b
  - Euler and Rudrata
  - Minimum cuts and balanced cuts
  - Integer linear programming or ILP ($Ax \leq b$) and Zero-one Equations or ZOE ($Ax = 1$)
  - Three dimensional matching
  - Independent set, vertex cover, clique problem (with budget b)
  - Longest path

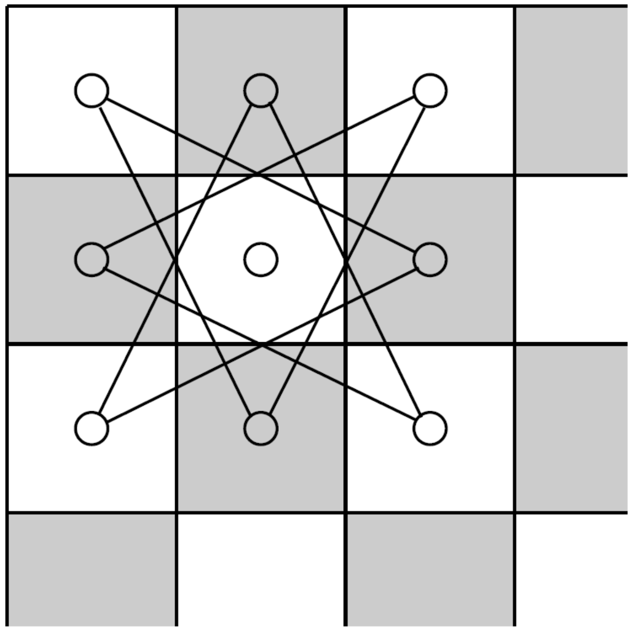
- Satisfability problems (Horn-SAT, 2SAT, 3SAT, KSAT)
- Horn's formula
  - Implications: $(z \wedge w) \Rightarrow u$
  - Pure negative clauses: $(\bar{u} \vee \bar{v} \vee \bar{y})$
  - Horn-SAT: Solvable using greedy algorithm in linear time
- 2SAT
  - in conjunctive normal form (CNF)
  - Each clause has two literals
  - example: $(x \vee y) \wedge (\bar{x} \vee z) \wedge (x \vee \bar{w})$
  - Can be solve in polynomial time (using implication graph + Strongly connected components)
- 3SAT
  - in conjunctive normal form (CNF)
  - Each clause has 3 literals
  - example: $(x \vee y \vee w) \wedge (\bar{x} \vee z \vee \bar{y}) \wedge (x \vee u \vee \bar{w})$
  - No polynomial time algorithm

Search Problems

- Euler's tour and Rudrata's problem



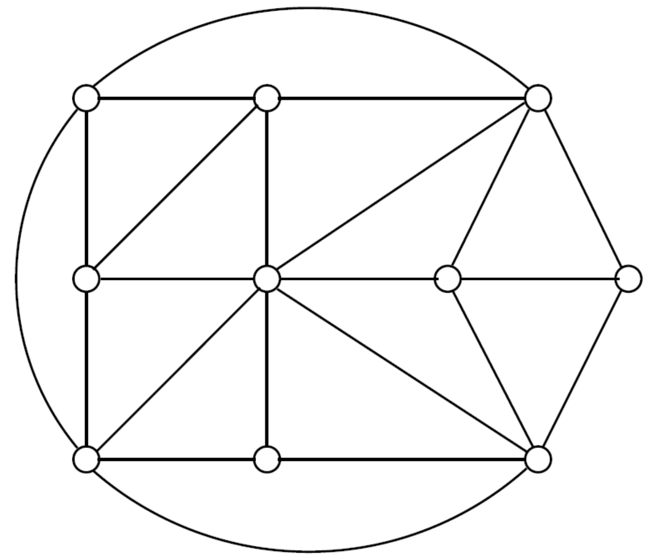
Euler's tour
visit all edges without repeating
Solvable in polynomial time



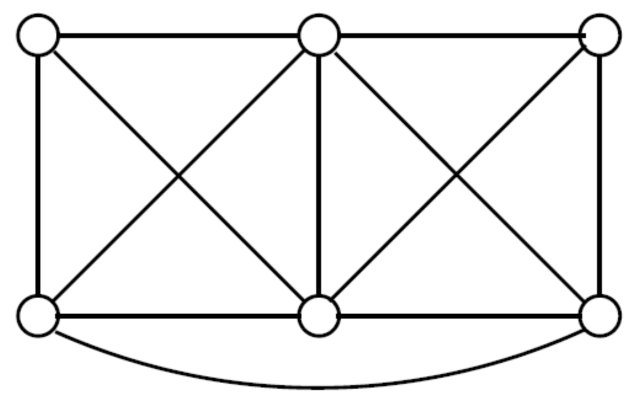
Rudrata's problem (aka Hamiltonian path/cycl
visit all vertices without repeating
No polynomial time algorithm

Search Problems

- Longest path (Taxicab rip-off problem)
  - Give a graph and two nodes s and t, find a path with length at least b from s to t without repeating vertices.
  - no polynomial time algorithm

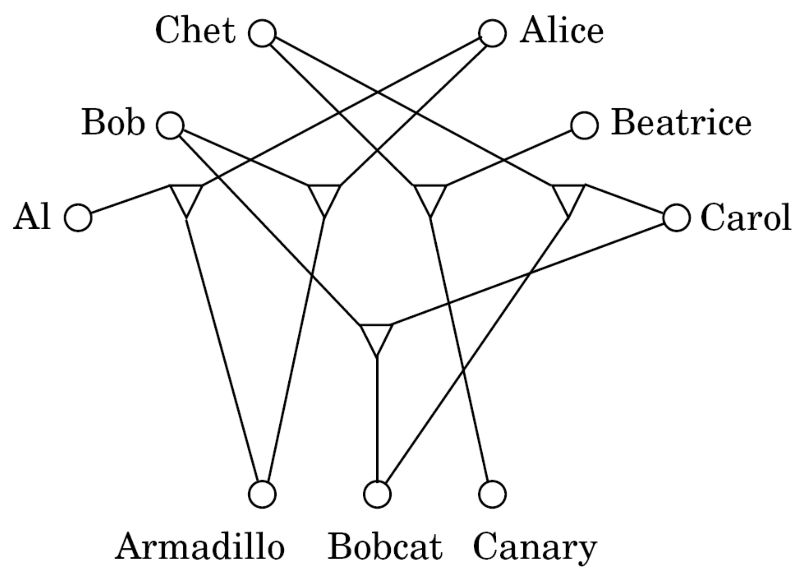


- Minimum cuts and balanced cuts
  - Find cuts that split the graph into two sets S and T
  - Minimum cuts problem can be solved using linear programming
  - Balanced cuts: $|S| \geq n/3$ $|T| \geq n/3$ and there are at most b edges between S and T
  - Balanced cuts problem has no polynomial time algorithm



Search Problems

- Three dimensional matching



- Independent set, vertex cover, clique problem (with budget b)
  - Independent set: A set of b vertices that are not adjacent to each other
  - vertex cover: A set of b vertices that are incident to all edges
  - clique: A set of b vertices that have all possible connections

Search Problems

- Knapsack problem and Subset sum
  - In subset sum, each item has same value and weight
  - Both problems have no polynomial time algorithms

- Integer linear programming or ILP ($Ax \leq b$) and Zero-one Equations or ZOE ($Ax = 1$)
  - Simplex method is not polynomial but LP can be solved in polynomial time
  - ILP requires the values of all variables to be integer
  - ZOE is a special type of ILP where all values in A are 0 or 1
  - Both problems have no polynomial time algorithms

Complexity Class

Problem Complexity

- **Tractable**: a problem is tractable if there is an algorithm can solve the problem deterministically in *polynomial time*

- Is a problem tractable or intractable?
  - yes (given an algorithm to support this answer)
  - no
    - because it's been proved that no algorithm exists at all (e.g., Turing's **Halting Problem**.)
    - because it's been be proved that any algorithm takes exponential time (Traveling Salesman Problem)
  - we have no idea...

P vs. NP

- Hard problems, easy problems

Hard problems **NP-complete**	Easy problems **P**
SAT, 3SAT	2SAT
Traveling Salesman Problem, Rudrata path	Chinese Postman Problem, Euler path
3D matching	Bipartite matching
Independent set	Independent set on trees
Integer linear programming	Linear programming
Balance cut	Minimum cut

- **P**: polynomial
  - Given an instance I, we can find a polynomial time algorithm to find an solution S
- **NP**: nondeterministic polynomial
  - Given an instance I and a proposed solution S, we can find a polynomial time algorithm C to check if S is an solution of I
  - Remember: A problem in **NP** does **NOT** mean it is a hard problem
- **NP hard**: all problems in NP can be **reduced** to a NP hard problem
  - A NP hard problem is at least as hard as the hardest problem in NP
- **NP complete**: in NP and also in NP hard

P vs. NP vs. NP hard vs. NP complete

- Their relationship

P vs. NP vs. NP hard vs. NP complete

- Problems in P
  sorting, MST, ...

- Problems in NP complete
  SAT, TSP, ILP, ...

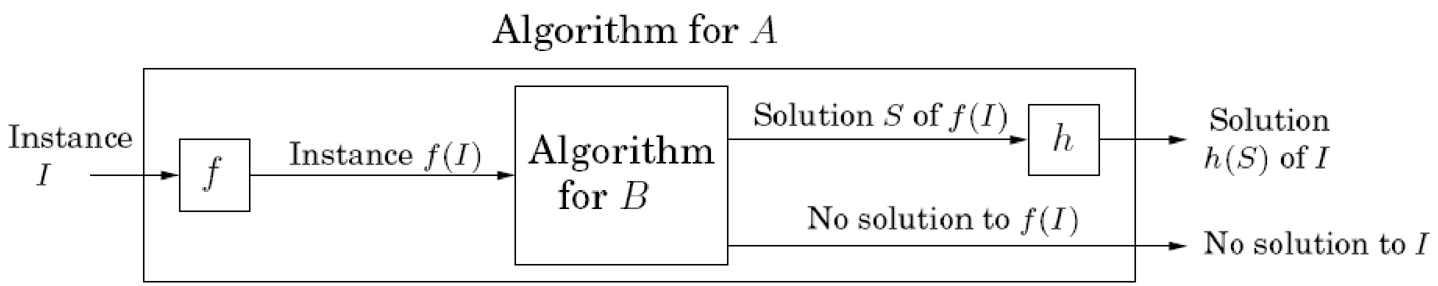
- Problems in NP but not in P or in NP complete
  factoring, graph isomorphism

- Problems not in NP
  Halting problem, counting the number of perfect matching, matrix permanent, ...

- P=NP? (Most Computer Scientists believe P $\neq$ NP)
- There are many more complexity classes than these three (PSpace, Co-NP, ExpTime, ExpSpace,...)
  In fact, there are 462 complexity classes according to the "Complexity zoo"
  http://qwiki.caltech.edu/wiki/Complexity_Zoo (maintained by Scott Aaronson and Greg Kuperberg)

Reductions

- If we reduce a problem A to a problem B in polynomial time (denoted as $A \rightarrow B$, we can say that B is as hard as A if not harder
  - If $A \rightarrow B$ and A is NP-complete then we know that B is also NP-complete



Reductions

- Reductions between NP-complete problems