NP-complete Problems: Search Problems

Alexander S. Kulikov

Steklov Institute of Mathematics at St. Petersburg
Russian Academy of Sciences

Advanced Algorithms and Complexity
Data Structures and Algorithms

Outline

1. Brute Force Search
2. Search Problems
3. Easy and Hard Problems
   - Traveling Salesman Problem
   - Hamiltonian Cycle Problem
   - Longest Path Problem
   - Integer Linear Programming Problem
   - Independent Set Problem
4. P and NP

Polynomial vs Exponential

running time:	n	n^2	n^3	2^n
less than 10^9:	10^9	$10^{4.5}$	10^3	29

Improving Brute Force Search

Usually, an efficient (polynomial) algorithm searches for a solution among an exponential number of candidates:

- there are $n!$ permutations of n objects

Improving Brute Force Search

Usually, an efficient (polynomial) algorithm searches for a solution among an exponential number of candidates:

- there are $n!$ permutations of n objects
- there are 2^n ways to partition n objects into two sets

Improving Brute Force Search

Usually, an efficient (polynomial) algorithm searches for a solution among an exponential number of candidates:

- there are $n!$ permutations of n objects
- there are 2^n ways to partition n objects into two sets
- there are n^{n-2} spanning trees in a complete graph on n vertices

This module

- For thousands of practically important problems we don't have an efficient algorithm yet

This module

- For thousands of practically important problems we don't have an efficient algorithm yet
- An efficient algorithm for one such problem automatically gives efficient algorithms for all these problems!

This module

- For thousands of practically important problems we don't have an efficient algorithm yet
- An efficient algorithm for one such problem automatically gives efficient algorithms for all these problems!
- \$1M prize for constructing such an algorithm or proving that this is impossible!

Outline

1. Brute Force Search
2. Search Problems
3. Easy and Hard Problems
    - Traveling Salesman Problem
    - Hamiltonian Cycle Problem
    - Longest Path Problem
    - Integer Linear Programming Problem
    - Independent Set Problem
4. P and NP

Boolean Formulas

Formula in conjunctive normal form

$(x \vee y \vee z)(x \vee \overline{y})(y \vee \overline{z})(z \vee \overline{x})(\overline{x} \vee \overline{y} \vee \overline{z})$

- x, y, z are Boolean variables (values: `true`/`false` or 1/0)
- literals are variables (x, y, z) and their negations ($\overline{x}, \overline{y}, \overline{z}$)
- clauses are disjunctions (logical or) of literals

Satisfiability (SAT)

Input: Formula F in conjunctive normal form (CNF).

Output: An assignment of Boolean values to the variables of F satisfying all clauses, if exists.

Examples

- The formula $(x \vee \overline{y})(\overline{x} \vee \overline{y})(x \vee y)$ is satisfiable: set $x = 1$, $y = 0$.
- The formula $(x \vee y \vee z)(x \vee \overline{y})(y \vee \overline{z})$ is satisfiable: set $x = 1, y = 1, z = 1$ or $x = 1, y = 0, z = 0$.
- The formula $(x \vee y \vee z)(x \vee \overline{y})(y \vee \overline{z})(z \vee \overline{x})(\overline{x} \vee \overline{y} \vee \overline{z})$ is unsatisfiable.

Satisfiability

- Classical hard problem
- Many applications: e.g., hardware/software verification, planning, scheduling
- Many hard problems are stated in terms of SAT naturally
- SAT solvers (will see later), SAT competition

- SAT is a typical search problem
- Search problem: given an instance I, find a solution S or report that none exists
- Main property: one must be able to check quickly whether S is indeed a solution for I
- By saying quickly, we mean in time polynomial in the length of I. In particular, the length of S should be polynomial in the length of I

Definition

A search problem is defined by an algorithm $\mathcal{C}$ that takes an instance I and a candidate solution S, and runs in time polynomial in the length of I. We say that S is a solution to I iff $\mathcal{C}(S, I) = \texttt{true}$.

Example

For SAT, I is a Boolean formula, S is an assignment of Boolean constants to its variables. The corresponding algorithm $\mathcal{C}$ checks whether S satisfies all clauses of I.

Next part

A few practical search problems for which polynomial algorithms remain unknown

Outline

1. Brute Force Search
2. Search Problems
3. Easy and Hard Problems
   - Traveling Salesman Problem
   - Hamiltonian Cycle Problem
   - Longest Path Problem
   - Integer Linear Programming Problem
   - Independent Set Problem
4. P and NP

Outline

1 Brute Force Search

2 Search Problems

3 Easy and Hard Problems
   - Traveling Salesman Problem
   - Hamiltonian Cycle Problem
   - Longest Path Problem
   - Integer Linear Programming Problem
   - Independent Set Problem

4 P and NP

Traveling salesman problem (TSP)

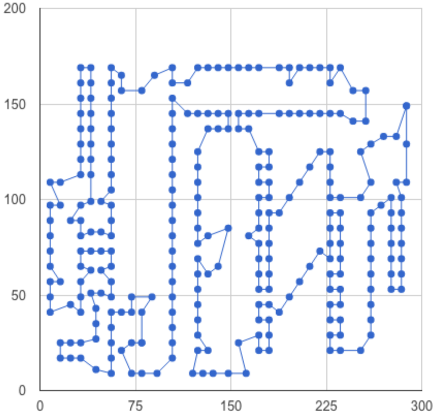
Input: Pairwise distances between n cities and a budget b.

Output: A cycle that visits each vertex exactly once and has total length at most b.

Delivery Company

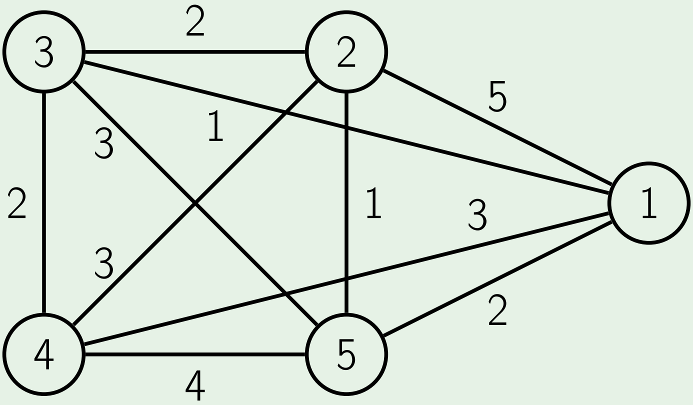
https://simple.wikipedia.org/wiki/Travelling_salesman_problem

Drilling Holes in a Circuit Board

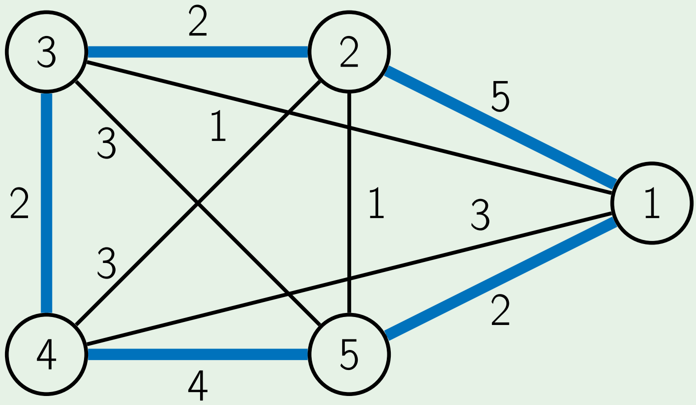


https://developers.google.com/optimization/routing/tsp

Example

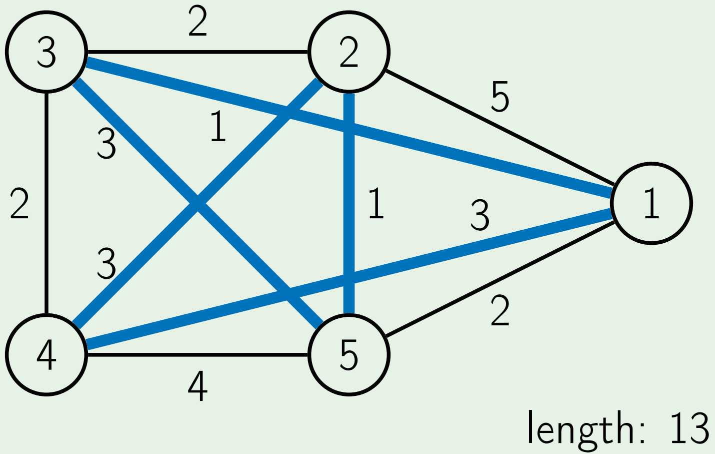


Example

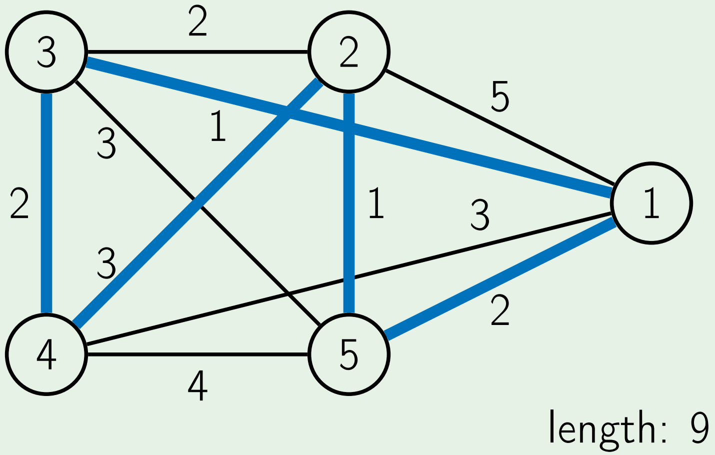


length: 15

Example



Example



Search Problem

- TSP is a search problem: given a sequence of vertices, it is easy to check whether it is a cycle visiting all the vertices of total length at most b
- TSP is usually stated as an optimization problem; we stated its decision version to guarantee that a candidate solution can be efficiently checked for correctness

Algorithms

- Check all permutations: about $O(n!)$, extremely slow
- Dynamic programming: $O(n^2 2^n)$
- No significantly better upper bound is known
- There are heuristic algorithms and approximation algorithms

Comparing to MST

MST

Decision version:
given n cities, connect them by $(n-1)$ roads of minimal total length

Comparing to MST

MST

Decision version: given n cities, connect them by $(n-1)$ roads of minimal total length

Can be solved efficiently

Comparing to MST

MST

Decision version: given n cities, connect them by $(n-1)$ roads of minimal total length

TSP

Decision version: given n cities, connect them in a path by $(n-1)$ roads of minimal total length

Can be solved efficiently

Comparing to MST

MST

Decision version: given n cities, connect them by $(n-1)$ roads of minimal total length

Can be solved efficiently

TSP

Decision version: given n cities, connect them in a path by $(n-1)$ roads of minimal total length

No polynomial algorithm known!

Outline

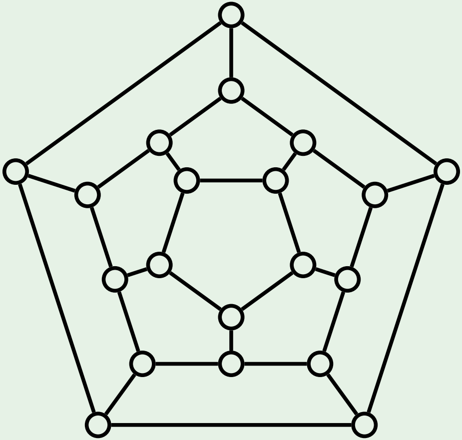
1. Brute Force Search
2. Search Problems
3. Easy and Hard Problems
   - Traveling Salesman Problem
   - Hamiltonian Cycle Problem
   - Longest Path Problem
   - Integer Linear Programming Problem
   - Independent Set Problem
4. P and NP

Hamiltonian cycle

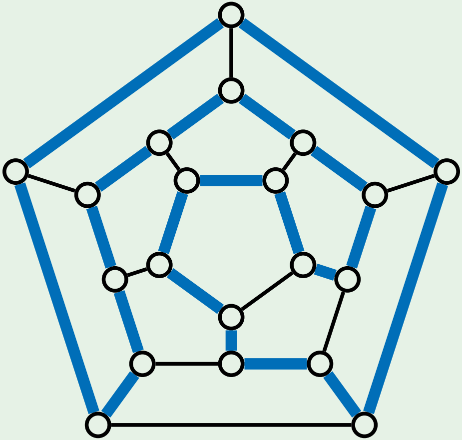
Input: A graph.

Output: A cycle that visits each vertex of the graph exactly once.

Example



Example



Eulerian cycle

Input:	A graph.
Output:	A cycle that visits each edge of the graph exactly once.

Eulerian cycle

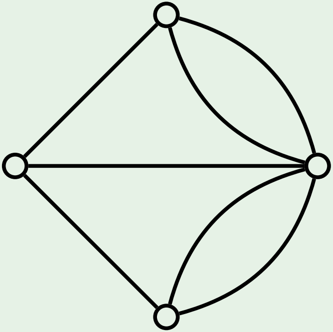
Input: A graph.

Output: A cycle that visits each edge of the graph exactly once.

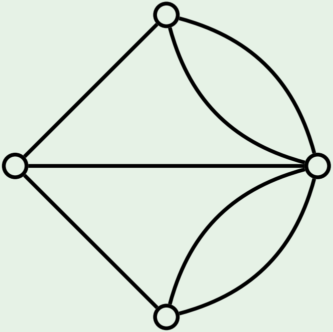
Theorem

A graph has an Eulerian cycle if and only if it is connected and the degree of each vertex is even.

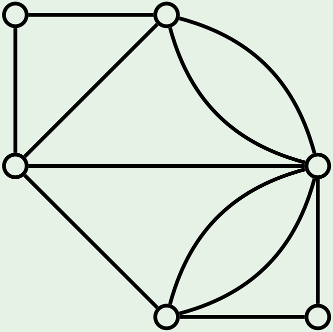
Non-Eulerian graph



Non-Eulerian graph



Eulerian graph



Eulerian cycle

Find a cycle visiting each edge exactly once

Eulerian cycle

Find a cycle visiting each edge exactly once

Can be solved efficiently

Eulerian cycle

Find a cycle visiting each edge exactly once

Can be solved efficiently

Hamiltonian cycle

Find a cycle visiting each vertex exactly once

Eulerian cycle

Find a cycle visiting each edge exactly once

Can be solved efficiently

Hamiltonian cycle

Find a cycle visiting each vertex exactly once

No polynomial algorithm known!

Outline

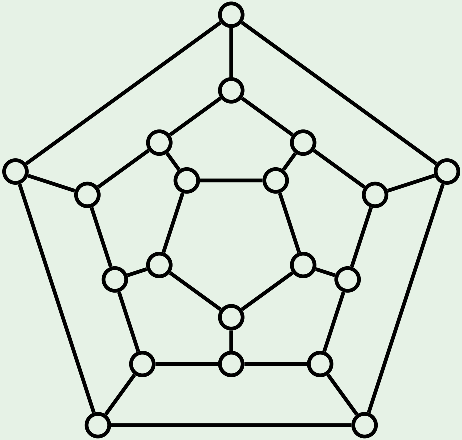
1. Brute Force Search
2. Search Problems
3. Easy and Hard Problems
   - Traveling Salesman Problem
   - Hamiltonian Cycle Problem
   - Longest Path Problem
   - Integer Linear Programming Problem
   - Independent Set Problem
4. P and NP

Longest path

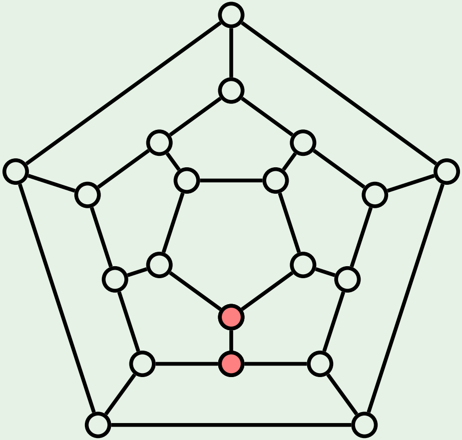
Input: A weighted graph, two vertices s, t, and a budget b.

Output: A simple path (containing no repeated vertices) of total length at least b.

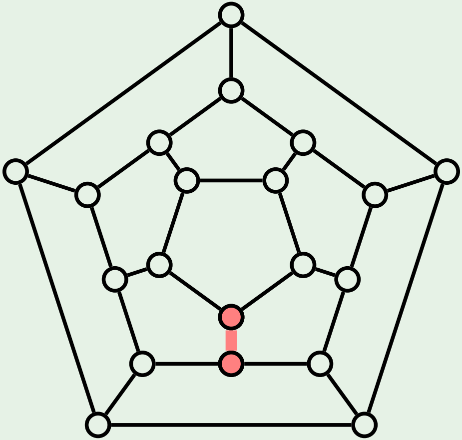
Example



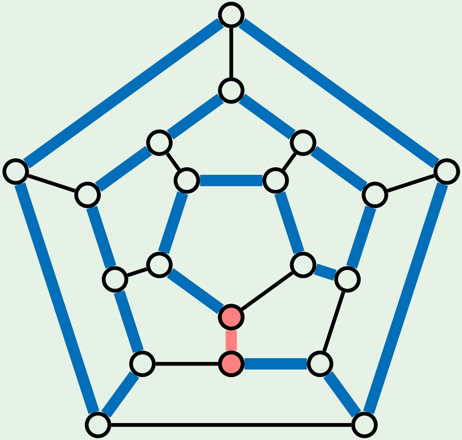
Example



Example



Example



Shortest path

Find a simple path from s to t of total length at most b

Shortest path

Find a simple path from s to t of total length at most b

Can be solved efficiently

Shortest path

Find a simple path from s to t of total length at most b

Longest path

Find a simple path from s to t of total length at least b

Can be solved efficiently

Shortest path

Find a simple path from s to t of total length at most b

Can be solved efficiently

Longest path

Find a simple path from s to t of total length at least b

No polynomial algorithm known!

Outline

1 Brute Force Search

2 Search Problems

3 Easy and Hard Problems

- Traveling Salesman Problem
- Hamiltonian Cycle Problem
- Longest Path Problem
- Integer Linear Programming Problem
- Independent Set Problem

4 P and NP

Integer linear programming

Input: A set of linear inequalities $\mathbf{A}\mathbf{x} \leq \mathbf{b}$.

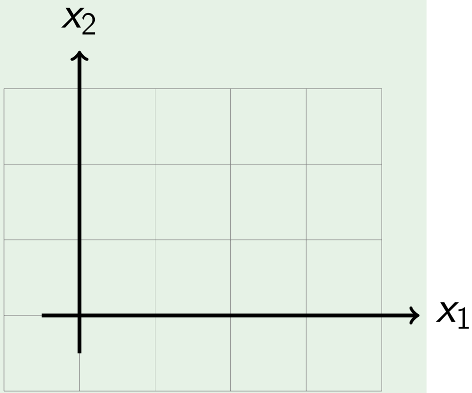
Output: Integer solution.

Example

$$
\begin{aligned}
x_1 &\geq 0.5 \\
-x_1 + 8x_2 &\geq 0 \\
-x_1 - 8x_2 &\geq -8
\end{aligned}
$$

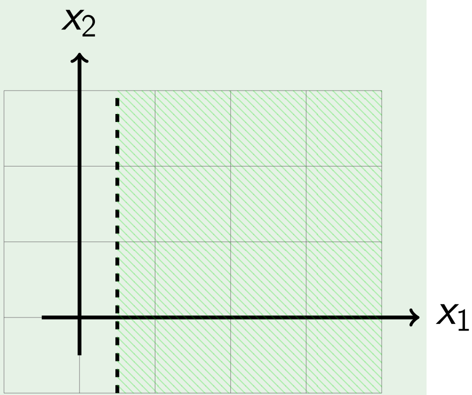
Example

$$\begin{aligned} x_1 &\geq 0.5 \\ -x_1 + 8x_2 &\geq 0 \\ -x_1 - 8x_2 &\geq -8 \end{aligned}$$



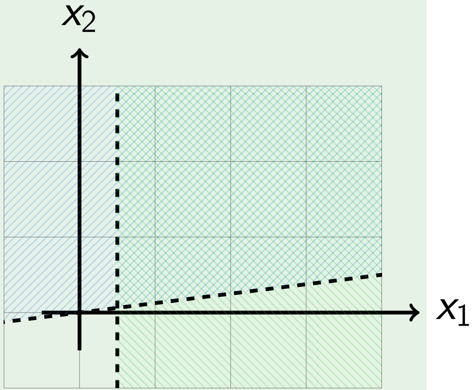
Example

$$
\begin{aligned}
x_1 &\geq 0.5 \\
-x_1 + 8x_2 &\geq 0 \\
-x_1 - 8x_2 &\geq -8
\end{aligned}
$$



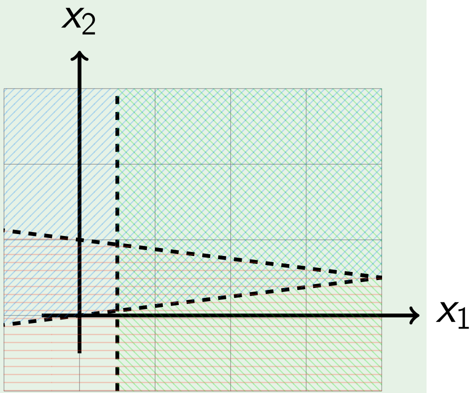
Example

$$x_1 \geq 0.5$$
$$-x_1 + 8x_2 \geq 0$$
$$-x_1 - 8x_2 \geq -8$$



Example

$$x_1 \geq 0.5$$
$$-x_1 + 8x_2 \geq 0$$
$$-x_1 - 8x_2 \geq -8$$



LP (decision version)

Find a real solution of a system of linear inequalities

LP (decision version)

Find a real solution of a system of linear inequalities

Can be solved efficiently

LP (decision version)

Find a real solution of a system of linear inequalities

ILP

Find an integer solution of a system of linear inequalities

Can be solved efficiently

LP (decision version)	ILP
Find a real solution of a system of linear inequalities	Find an integer solution of a system of linear inequalities

Can be solved efficiently	No polynomial algorithm known!

Outline

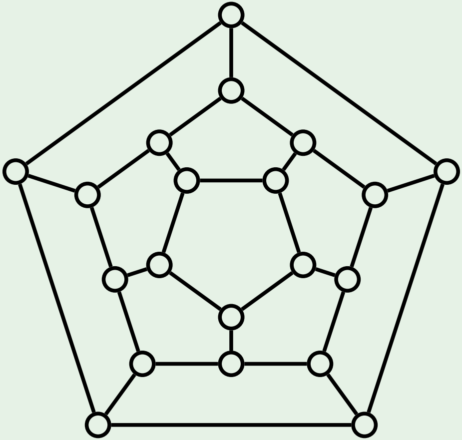
1. Brute Force Search
2. Search Problems
3. Easy and Hard Problems
   - Traveling Salesman Problem
   - Hamiltonian Cycle Problem
   - Longest Path Problem
   - Integer Linear Programming Problem
   - Independent Set Problem
4. P and NP

Independent set

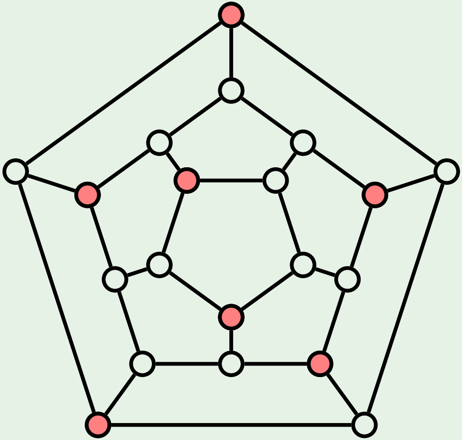
Input: A graph and a budget b.

Output: A subset of vertices of size at least b such that no two of them are adjacent.

Example

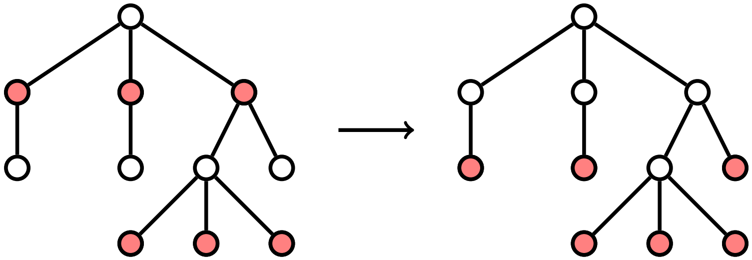


Example



Independent Sets in a Tree

A maximal independent set in a tree can be found by a simple greedy algorithm: it is safe to take into a solution all the leaves.



Independent set in a tree

Find an independent set of size at least b in a given tree

Independent set in a tree

Find an independent set of size at least b in a given tree

Can be solved efficiently

Independent set in a tree

Find an independent set of size at least b in a given tree

Independent set in a graph

Find an independent set of size at least b in a given graph

Can be solved efficiently

Independent set in a tree

Find an independent set of size at least b in a given tree

Independent set in a graph

Find an independent set of size at least b in a given graph

Can be solved efficiently

No polynomial algorithm known!

Next part

It turns out that all these hard problems are in a sense a single hard problem:
a polynomial time algorithm for any of these problems can be used to solve all of them in polynomial time!

Outline

1. Brute Force Search
2. Search Problems
3. Easy and Hard Problems
    - Traveling Salesman Problem
    - Hamiltonian Cycle Problem
    - Longest Path Problem
    - Integer Linear Programming Problem
    - Independent Set Problem
4. P and NP

Class NP

Definition

A search problem is defined by an algorithm $\mathcal{C}$ that takes an instance I and a candidate solution S, and runs in time polynomial in the length of I. We say that S is a solution to I iff $\mathcal{C}(S, I) = \texttt{true}$.

Class NP

Definition

A search problem is defined by an algorithm $\mathcal{C}$ that takes an instance I and a candidate solution S, and runs in time polynomial in the length of I. We say that S is a solution to I iff $\mathcal{C}(S, I) = \texttt{true}$.

Definition

NP is the class of all search problems.

- **NP** stands for “non-deterministic polynomial time”: one can guess a solution, and then verify its correctness in polynomial time
- In other words, the class **NP** contains all problems whose solutions can be efficiently verified

Class **P**

Definition

P is the class of all search problems that can be solved in polynomial time.

Class P

Problems whose solution can be found efficiently

Class P

Problems whose solution can be found efficiently

- MST
- Shortest path
- LP
- IS on trees

Class P

Problems whose solution can be found efficiently

Class NP

Problems whose solution can be verified efficiently

- MST
- Shortest path
- LP
- IS on trees

Class **P**

Problems whose solution can be found efficiently

- MST
- Shortest path
- LP
- IS on trees

Class **NP**

Problems whose solution can be verified efficiently

- TSP
- Longest path
- ILP
- IS on graphs

The main open problem in Computer Science

Is $\mathbf{P}$ equal to $\mathbf{NP}$?

The main open problem in Computer Science

Is P equal to NP?

Millenium Prize Problem

Clay Mathematics Institute: \$1M prize for solving the problem

- If **P**=**NP**, then all search problems can be solved in polynomial time.

- If **P**=**NP**, then all search problems can be solved in polynomial time.
- If **P**$\neq$**NP**, then there exist search problems that cannot be solved in polynomial time.

Next part

We'll show that the satisfiability problem, the traveling salesman problem, the independent set problem, the integer linear programming are the hardest problems in **NP**.