Theory of Algorithms I

NP Problems

Guoqiang Li

School of Software, Shanghai Jiao Tong University

Instructor and Teaching Assistants

- Guoqiang LI

Instructor and Teaching Assistants

- Guoqiang LI
  - Homepage: http://basics.sjtu.edu.cn/~liguoqiang
  - Course page: http://basics.sjtu.edu.cn/~liguoqiang/teaching/Ualgo18/index.htm
  - Email: li.g@outlook.com
  - Office: Rm. 1212, Building of Software
  - Phone: 3420-4167

Instructor and Teaching Assistants

- Guoqiang LI
  - Homepage: http://basics.sjtu.edu.cn/~liguoqiang
  - Course page: http://basics.sjtu.edu.cn/~liguoqiang/teaching/Ualgo18/index.htm
  - Email: li.g@outlook.com
  - Office: Rm. 1212, Building of Software
  - Phone: 3420-4167
- TA:
  - Min GAO: gaomin86 (AT) sjtu (DOT) edu (DOT) cn

Instructor and Teaching Assistants

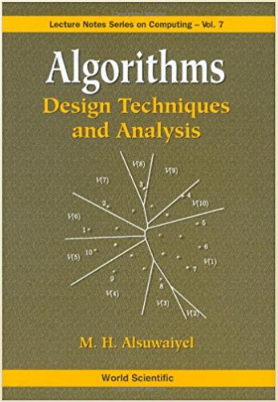
- Guoqiang LI
    - Homepage: http://basics.sjtu.edu.cn/~liguoqiang
    - Course page: http://basics.sjtu.edu.cn/~liguoqiang/teaching/Ualgo18/index.htm
    - Email: li.g@outlook.com
    - Office: Rm. 1212, Building of Software
    - Phone: 3420-4167
- TA:
    - Min GAO: gaomin86 (AT) sjtu (DOT) edu (DOT) cn
- Office hour: Tue. 14:00-17:00 @ Software Building 3203

Instructor and Teaching Assistants

- Guoqiang LI
  - Homepage: http://basics.sjtu.edu.cn/~liguoqiang
  - Course page: http://basics.sjtu.edu.cn/~liguoqiang/teaching/Ualgo18/index.htm
  - Email: li.g@outlook.com
  - Office: Rm. 1212, Building of Software
  - Phone: 3420-4167
- TA:
  - Min GAO: gaomin86 (AT) sjtu (DOT) edu (DOT) cn
- Office hour: Tue. 14:00-17:00 @ Software Building 3203
- Office hour: Wed. 13:30-15:00 @ Software Building 1212 (need reserve!)

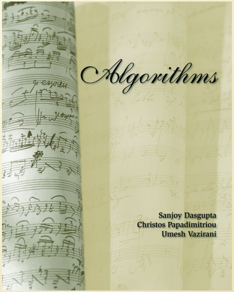
Text Book

- Algorithms: Design Techniques and Analysis
  - M. H. Alsuwalyel
  - World Scientific Publishing, 1999.



Text Book

- Algorithms
  - Sanjoy Dasgupta
    University of California
  - San Diego Christos Papadimitriou
    University of California at Berkeley
  - Umesh Vazirani
    University of California at Berkeley
  - McGraw-Hill, 2007.



- Available at:
  http://www.cs.berkeley.edu/~vazirani/algorithms.html

Which one comes first, computer or algorithms?

Al Khwarizmi

Al Khwarizmi (780 - 850)

Al Khwarizmi

Al Khwarizmi (780 - 850)

In the 12th century, Latin translations of his work on the Indian numerals, introduced the decimal system to the Western world. (Source: Wikipedia)

Algorithms

- Al Khwarizmi laid out the basic methods for
  - adding,
  - multiplying,
  - dividing numbers,
  - extracting square roots,
  - calculating digits of π.

Algorithms

- Al Khwarizmi laid out the basic methods for
  - adding,
  - multiplying,
  - dividing numbers,
  - extracting square roots,
  - calculating digits of π.
- These procedures were precise, unambiguous, mechanical, efficient, correct.
- They were algorithms, a term coined to honor the wise man after the decimal system was finally adopted in Europe, many centuries later.

Efficient Algorithms

- We have developed algorithms for

Efficient Algorithms

- We have developed algorithms for
  - Finding shortest paths in graphs,
  - Minimum spanning trees in graphs,
  - Matchings in bipartite graphs,
  - Maximum increasing subsequences,
  - Maximum flows in networks,
  - ……

Efficient Algorithms

- We have developed algorithms for
  - Finding shortest paths in graphs,
  - Minimum spanning trees in graphs,
  - Matchings in bipartite graphs,
  - Maximum increasing subsequences,
  - Maximum flows in networks,
  - ……
- All these algorithms are efficient, because in each case their time requirement grows as a polynomial function (such as n, n^2, or n^3) of the size of the input.

Exponential Search Space

- A solution (path, tree, matching) is searched from among an exponential population of possibilities.

Exponential Search Space

- A solution (path, tree, matching) is searched from among an exponential population of possibilities.
- Be solved in exponential time by checking through all candidate solutions.

Exponential Search Space

- A solution (path, tree, matching) is searched from among an exponential population of possibilities.
- Be solved in exponential time by checking through all candidate solutions.
- An algorithm with running time 2^n, or worse, is useless in practice.

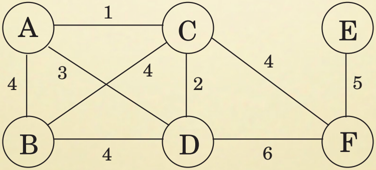
Exponential Search Space

- A solution (path, tree, matching) is searched from among an exponential population of possibilities.
- Be solved in exponential time by checking through all candidate solutions.
- An algorithm with running time 2^n, or worse, is useless in practice.
- Efficient algorithms is about finding clever ways to bypass this process of exhaustive search, dramatically narrowing down the search space.

Minimum Spanning Trees

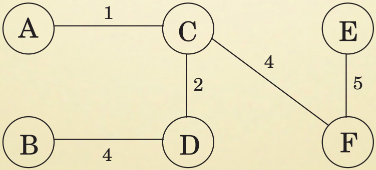
Build a Network

- Suppose you are asked to network a collection of computers by linking selected pairs of them.
- This translates into a graph problem in which
  - nodes are computers,
  - undirected edges are potential links, each with a maintenance cost.



Build a Network

- The goal is to
  - pick enough of these edges that the nodes are connected,
  - the total maintenance cost is minimum.
- One immediate observation is that the optimal set of edges cannot contain a cycle.



A Greedy Approach

- Kruskal's minimum spanning tree algorithm starts with the empty graph and then selects edges from E according to the following rule.
- Repeatedly add the next lightest edge that doesn't produce a cycle.

Example:
Starting with an empty graph and then attempt to add edges in increasing order of weight

$$B-C; C-D; B-D; C-F; D-F; E-F; A-D; A-B; C-E; A-C$$

A General Kruskal's Algorithm

$X = \{\ \}$;
repeat until $|X| = |V| - 1$;
    pick a set $S \subset V$ for which X has no edges between S and $V - S$;
    let $e \in E$ be the minimum-weight edge between S and $V - S$;
    $X = X \cup \{e\}$;

Prim's Algorithm

- A popular alternative to Kruskal's algorithm is Prim's, in which the intermediate set of edges X always forms a subtree, and S is chosen to be the set of this tree's vertices.
- On each iteration, the subtree defined by X grows by one edge, namely, the lightest edge between a vertex in S and a vertex outside S. We can equivalently think of S as growing to include the vertex $v \notin S$ of smallest `cost`:

$$\mathtt{cost}(v) = \min_{u \in S} w(u, v)$$

A Little Change of the MST

What if the tree is not allowed to branch?

Satisfiability Problem

Satisfiability

The instances of Satisfiability or SAT:

Satisfiability

The instances of Satisfiability or SAT:

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

Satisfiability

The instances of Satisfiability or SAT:

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

That is, a Boolean formula in conjunctive normal form (CNF).

Satisfiability

The instances of Satisfiability or SAT:

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

That is, a Boolean formula in conjunctive normal form (CNF).

- It is a collection of clauses (the parentheses),

Satisfiability

The instances of Satisfiability or SAT:

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

That is, a Boolean formula in conjunctive normal form (CNF).

- It is a collection of clauses (the parentheses),
  - each consisting of the disjunction (logical or, denoted $\vee$) of several literals;

Satisfiability

The instances of Satisfiability or SAT:

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

That is, a Boolean formula in conjunctive normal form (CNF).

- It is a collection of clauses (the parentheses),
  - each consisting of the disjunction (logical or, denoted $\vee$) of several literals;
  - a literal is either a Boolean variable (such as x) or the negation of one (such as $\overline{x}$).

Satisfiability

The instances of Satisfiability or SAT:

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

That is, a Boolean formula in conjunctive normal form (CNF).

- It is a collection of clauses (the parentheses),
    - each consisting of the disjunction (logical or, denoted $\vee$) of several literals;
    - a literal is either a Boolean variable (such as x) or the negation of one (such as $\overline{x}$).
- A satisfying truth assignment is an assignment of `false` or `true` to each variable so that every clause contains a literal whose value is `true`.

Satisfiability

The instances of Satisfiability or SAT:

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

That is, a Boolean formula in conjunctive normal form (CNF).

- It is a collection of clauses (the parentheses),
  - each consisting of the disjunction (logical or, denoted $\vee$) of several literals;
  - a literal is either a Boolean variable (such as x) or the negation of one (such as $\overline{x}$).
- A satisfying truth assignment is an assignment of `false` or `true` to each variable so that every clause contains a literal whose value is `true`.
- Given a Boolean formula in conjunctive normal form, either find a satisfying truth assignment or else report that none exists.

2-SAT

Given a set of clauses, where each clause is the disjunction of two literals.

$$(x_1 \vee x_2) \wedge (\overline{x}_1 \vee x_3) \wedge (x_1 \vee \overline{x}_2) \wedge (x_3 \vee x_4) \wedge (\overline{x}_1 \vee \overline{x}_4)$$

2-SAT

Given a set of clauses, where each clause is the disjunction of two literals.

$$(x_1 \vee x_2) \wedge (\overline{x}_1 \vee x_3) \wedge (x_1 \vee \overline{x}_2) \wedge (x_3 \vee x_4) \wedge (\overline{x}_1 \vee \overline{x}_4)$$

Given an instance I of 2-Sat with n variables and m clauses, construct a directed graph $G_I = (V, E)$ as follows.

2-SAT

Given a set of clauses, where each clause is the disjunction of two literals.

$$(x_1 \vee x_2) \wedge (\overline{x}_1 \vee x_3) \wedge (x_1 \vee \overline{x}_2) \wedge (x_3 \vee x_4) \wedge (\overline{x}_1 \vee \overline{x}_4)$$

Given an instance I of 2-Sat with n variables and m clauses, construct a directed graph $G_I = (V, E)$ as follows.

- G_I has $2n$ nodes, one for each variable and its negation.

2-SAT

Given a set of clauses, where each clause is the disjunction of two literals.

$$(x_1 \vee x_2) \wedge (\overline{x}_1 \vee x_3) \wedge (x_1 \vee \overline{x}_2) \wedge (x_3 \vee x_4) \wedge (\overline{x}_1 \vee \overline{x}_4)$$

Given an instance I of 2-Sat with n variables and m clauses, construct a directed graph $G_I = (V, E)$ as follows.

- G_I has $2n$ nodes, one for each variable and its negation.
- G_I has $2m$ edges: for each clause $(\alpha \vee \beta)$ of I, G_I has an edge from the negation of α to β, and one from the negation of β to α.

2-SAT

Show that if G_I has a strongly connected component containing both x and $\overline{x}$ for some variable x, then I has no satisfying assignment.

2-SAT

Show that if G_I has a strongly connected component containing both x and $\overline{x}$ for some variable x, then I has no satisfying assignment.

If none of G_I's strongly connected components contain both a literal and its negation, then the instance I must be satisfiable.

2-SAT

Show that if G_I has a strongly connected component containing both x and $\overline{x}$ for some variable x, then I has no satisfying assignment.

If none of G_I's strongly connected components contain both a literal and its negation, then the instance I must be satisfiable.

Conclude that there is a linear-time algorithm for solving 2-SAT.

A Little Extension of 2-SAT

How about 3-SAT, n-SAT?

Tractable and Intractable Problems

Tractability and Intractability

Tractable Problems: can be solved in polynomial time

Tractability and Intractability

Tractable Problems: can be solved in polynomial time

Intractable Problems: unlikely to be solved in polynomial time

Decision Problem

Decision problems are those whose solutions have only two possible outcomes: Yes or No.

Decision Problem

Decision problems are those whose solutions have only two possible outcomes: Yes or No.

An algorithm that solves a decision problem can be easily modified to solve its corresponding optimization problem.

Element Uniqueness

Decision problem: Element Uniqueness
Input: A sequence of integers
Question: Are there two elements in S that are equal?

Element Uniqueness

Decision problem: Element Uniqueness
Input: A sequence of integers
Question: Are there two elements in S that are equal?

Optimization Problem: Element Count
Input: A sequence of integers
Question: An element in S of highest frequency?

Clique

Decision problem: Clique
Input: An undirected graph $G = (V, E)$ and a positive integer k
Question: Does G have a clique of size k?

Clique

Decision problem: Clique
Input: An undirected graph $G = (V, E)$ and a positive integer k
Question: Does G have a clique of size k?

Optimization Problem: Max-Clique
Input: An undirected graph $G = (V, E)$
Question: The maximum clique size of G?

Coloring

Decision problem: Coloring
Input: An undirected graph $G = (V, E)$ and a positive integer k
Question: Is G k-colorable?

Coloring

Decision problem: Coloring
Input: An undirected graph $G = (V, E)$ and a positive integer k
Question: Is G k-colorable?

Optimization Problem: Chromatic Number
Input: An undirected graph $G = (V, E)$
Question: The chromatic number $\chi(G)$ of G?

The Class P

Let A be an algorithm to solve a problem Π. We say that A is *deterministic* if, when presented with an instance of the problem Π, it has only one choice in each step throughout its execution. Thus, if A is run again and again on the same input instance, its output never changes.

The Class P

The class of decision problems P consists of those decision problems whose *yes/no* solution can be obtained using a deterministic algorithm that runs in polynomial number of steps, i.e., in $O(n^k)$ steps, for some nonnegative integer k, where n is the input size.

The Class P

Some problems in P: 2-Coloring, 2-Sat, 2-DM

The Closure Property of P

The class P is closed under complement.

Nondeterministic Algorithm

On input x, a nondeterministic algorithm consists of two phases:

Nondeterministic Algorithm

On input x, a nondeterministic algorithm consists of two phases:

- The guessing phase. An arbitrary string of characters y is generated in $O(|x|^i)$ time for some positive integer i. The y may or may not be a solution; it may or may not be in proper format of a solution. It may differ from one run to another.

Nondeterministic Algorithm

On input x, a nondeterministic algorithm consists of two phases:

- The guessing phase. An arbitrary string of characters y is generated in $O(|x|^i)$ time for some positive integer i. The y may or may not be a solution; it may or may not be in proper format of a solution. It may differ from one run to another.

- The verification phase. A deterministic algorithm verifies two things in $O(|x|^j)$ time for some positive integer j: It checks if y is in proper format. If not then answer *no*; otherwise it checks if y is a solution to the instance x. If y is a solution to the instance x then answer *yes*, otherwise answer *no*.

Nondeterministic Algorithm

Let A be a nondeterministic algorithm for a problem Π. We say that A accepts an instance I of Π iff on input I there is a guess that leads to a *yes* answer.

Nondeterministic Algorithm

Let A be a nondeterministic algorithm for a problem Π. We say that A accepts an instance I of Π iff on input I there is a guess that leads to a *yes* answer.

Running time: $O(|x|^i) + O(|x|^j)$

The Class NP

The class of decision problem NP consists of those decision problems for which there exists a nondeterministic algorithm that run in polynomial time.

Example

The problem Coloring is in NP.

Argument:
An algorithm A does two things: (i) A guesses a solution by generating an arbitrary assignment of the colors to the vertexes. (ii) A verifies if the guess is a valid assignment.

P and NP

P is the class of decision problems that we can **decide** or **solve** using a deterministic algorithm that runs in polynomial time.

P and NP

P is the class of decision problems that we can **decide** or **solve** using a deterministic algorithm that runs in polynomial time.

NP is the class of decision problems that we can **check** or **verify** their solutions using a deterministic algorithm that runs in polynomial time.

P and NP

P is the class of decision problems that we can **decide** or **solve** using a deterministic algorithm that runs in polynomial time.

NP is the class of decision problems that we can **check** or **verify** their solutions using a deterministic algorithm that runs in polynomial time.

Solution Searching vs. Solution Checking.

Polynomial Time Reduction

Let Π and Π' be two decision problems. We say that Π reduces to Π' in polynomial time, symbolized as $\Pi \propto_{poly} \Pi'$, if there exists a deterministic algorithm A that behaves as follows. When A is presented with an instance I of problem Π, it transforms it into an instance I' of problem Π' such that the answer to I is *yes* if the answer to I' is *yes*. Moreover this transformation must be achieved in polynomial time.

NP-Complete Problem

A decision problem Π is said to be NP-hard if, for every problem Π' in NP, $\Pi' \propto_{poly} \Pi$.

NP-Complete Problem

A decision problem Π is said to be NP-hard if, for every problem Π' in NP, $\Pi' \propto_{poly} \Pi$.

A decision problem Π is said to be NP-complete if the following two properties hold:

1. Π is in NP, and
2. for every problem Π' in NP, $\Pi' \propto_{poly} \Pi$.

The Satisfiability Problem

Conjunctive normal form

$$f = (x_1 \vee x_2) \wedge (\overline{x_1} \vee x_3 \vee x_4 \vee \overline{x_5}) \wedge (x_1 \vee \overline{x_3} \vee x_4)$$

A formula is said to be satisfiable if there is a truth assignment to its variables that makes it true.

The Satisfiability Problem

Decision Problem: Satisfiability
Input: A CNF boolean formula f.
Question: Is f satisfiable?

SAT: the First NP-Complete Problem

Cook's Theorem. Satisfiability is NP-Complete.

Establish NP-Completeness Result

Theorem.

Let Π, Π' and Π'' be three decision problems such that $\Pi \propto_{poly} \Pi'$ and $\Pi' \propto_{poly} \Pi''$. Then $\Pi \propto_{poly} \Pi''$.

Establish NP-Completeness Result

Theorem.

Let Π, Π' and Π'' be three decision problems such that $\Pi \propto_{poly} \Pi'$ and $\Pi' \propto_{poly} \Pi''$. Then $\Pi \propto_{poly} \Pi''$.

Corollary.

If Π and Π' are two problems in NP such that $\Pi' \propto_{poly} \Pi$, and Π' is NP-complete, then Π is NP-complete.

An Example

Hamiltonian Cycle: Given an undirected graph $G = (V, E)$, does it have a Hamiltonian cycle, i.e., a cycle that visits each vertex exactly once?

An Example

Hamiltonian Cycle: Given an undirected graph $G = (V, E)$, does it have a Hamiltonian cycle, i.e., a cycle that visits each vertex exactly once?

Traveling Salesman: Given a set of n cities with their intercity distances, and an integer k, does there exist a *tour* of length at most k? Here a tour is a cycle that visits each city exactly once.

An Example

Hamiltonian Cycle: Given an undirected graph $G = (V, E)$, does it have a Hamiltonian cycle, i.e., a cycle that visits each vertex exactly once?

Traveling Salesman: Given a set of n cities with their intercity distances, and an integer k, does there exist a *tour* of length at most k? Here a tour is a cycle that visits each city exactly once.

Fact: Hamiltonian Cycle $\propto_{poly}$ Traveling Salesman

Vertex Cover, Independence Set, Clique Problems

Clique: Given an undirected graph $G = (V, E)$ and a positive integer k, does G contain a clique of size k?

Vertex Cover, Independence Set, Clique Problems

Clique: Given an undirected graph $G = (V, E)$ and a positive integer k, does G contain a clique of size k?

Vertex Cover: Given an undirected graph $G = (V, E)$ and a positive integer k, is there a subset $C \subseteq V$ of size k such that each edge in E is incident to at least one vertex in C?

Vertex Cover, Independence Set, Clique Problems

Clique: Given an undirected graph $G = (V, E)$ and a positive integer k, does G contain a clique of size k?

Vertex Cover: Given an undirected graph $G = (V, E)$ and a positive integer k, is there a subset $C \subseteq V$ of size k such that each edge in E is incident to at least one vertex in C?

Independence Set: Given an undirected graph $G = (V, E)$ and a positive integer k, is there a subset $S \subseteq V$ of k vertices such that for each pair of vertices $u, w \in S$, $(u, w) \notin E$?

Satisfiability $\propto_{poly}$ Clique

Given an instance of Satisfiability $f = C_1 \wedge \ldots \wedge C_m$ with m clauses and n boolean variables $x_1, \ldots, x_n$, we construct a graph $G = (V, E)$, where V is the set of all **occurrences** of the $2n$ literals, and

$$\{(x_i, x_j) \mid x_i, x_j \text{ are in two different clauses and } x_i \neq \overline{x_j}\}$$

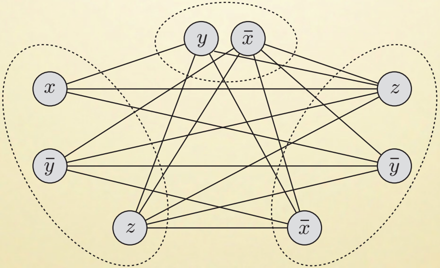
Fact: f is satisfiable iff G has a clique of size m.

Example

$$f = (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee y) \wedge (\overline{x} \vee \overline{y} \vee z)$$

Example

$$f = (x \vee \bar{y} \vee z) \wedge (\bar{x} \vee y) \wedge (\bar{x} \vee \bar{y} \vee z)$$



Satisfiability $\propto_{poly}$ Vertex Cover

Given an instance of Satisfiability $f = C_1 \wedge \ldots \wedge C_m$ with m clauses and n boolean variables $x_1, \ldots, x_n$, we construct I' as follows:

1. For each boolean variable x_i in f, G contains a pair of vertices x_i and $\overline{x_i}$ joined by an edge.
2. For each clause C_j containing n_j literals, G contains a clique C_j of size n_j.
3. For each vertex w in C_j, there is an edge connecting w to its corresponding literal in the vertex pairs $(x_i, \overline{x_i})$ constructed in part (1).
4. Let $k = n + \sum_{j=1}^{m} (n_j - 1)$.

Satisfiability $\propto_{poly}$ Vertex Cover

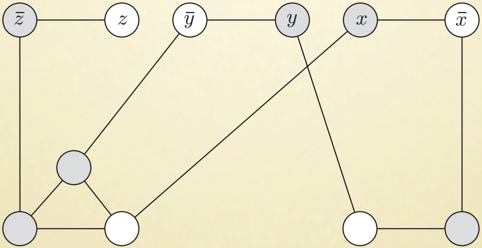
For instance

$$f = (x \vee \bar{y} \vee \bar{z}) \wedge (\bar{x} \vee y)$$

Satisfiability $\propto_{poly}$ Vertex Cover

For instance

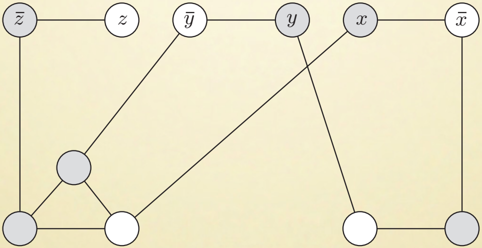
$$f = (x \vee \bar{y} \vee \bar{z}) \wedge (\bar{x} \vee y)$$



Satisfiability $\propto_{poly}$ Vertex Cover

For instance

$$f = (x \vee \bar{y} \vee \bar{z}) \wedge (\bar{x} \vee y)$$



Fact: f is satisfiable iff the constructed graph has a vertex cover of size k.

Vertex Cover $\propto_{poly}$ Independence Set

Fact: Let $G = (V, E)$ be a connected undirected graph. Then $S \subseteq V$ is an independence set iff $V \setminus S$ is a vertex cover in G.

More NP-Complete Problems

1. **3-SAT**. Given a boolean formula f in conjunctive normal form such that each clause consists of three literals, is f satisfiable?

2. **3-Coloring**. Given an undirected graph $G = (V, E)$, can G be colored using three colors?

3. **3-DimensionalMatching**. Let X, Y, Z be pairwise disjoint sets of size k each. Let W be the set of triples

$$\{(x, y, z) \mid x \in X, y \in Y, z \in Z\}$$

Does there exist a *perfect matching* M of W? That is, does there exist a subset $M \subseteq W$ of size k such that no two triples in M agree in any coordinate?

More NP-Complete Problems

4. **Hamiltonian Path**. Given an undirected graph $G = (V, E)$, does it contain a simple open path that visits each vertex exactly once?

5. **Longest Path**. Given a weighted graph $G = (V, E)$, two distinguished vertices $s, t \in V$ and a positive integer c, is there a *simple* path in G from s to t of length c or more?

6. **Partition**. Given a set S of n integers, is it possible to partition S into two subsets S_1 and S_2 so that the sum of the integers in S_1 is equal to the sum of the integers in S_2?

More NP-Complete Problems

7. **BinPacking**. Given n items with sizes $s_1, s_2, \ldots, s_n$, a bin capacity C and a positive integer k, is it possible to pack the n items using at most k bins?

8. **SetCover**. Given a set X, a family $\mathcal{F}$ of subsets of X and an integer k between 1 and $|\mathcal{F}|$, do there exist k subsets in $\mathcal{F}$ whose union is X?

9. **Knapsack**. Given n items with sizes $s_1, s_2, \ldots, s_n$ and values $v_1, v_2, \ldots, v_n$, a knapsack capacity C and a constant integer k, is it possible to fill the knapsack with some of these items whose total size is at most C and whose total value is at least k? This problem can be solved in time $\Theta(nC)$ using dynamic programming.

VertexCover $\propto_{poly}$ SetCover

SetCover. Given a set X, a family $\mathcal{F}$ of subsets of X and an integer k between 1 and $|\mathcal{F}|$, do there exist k subsets in $\mathcal{F}$ whose union is X?

SAT $\propto_{poly}$ 3SAT

From SAT to 3SAT.

The Class co-NP

The class co-NP consists of those problems whose complements are in NP.

The Class co-NP

The class co-NP consists of those problems whose complements are in NP.

It is highly unlikely that co-NP=NP. Consider for example the complement of Traveling Salesman and the complement of Satisfiability.

The Class co-NP

A problem Π is complete for the class co-NP if

1. Π is in co-NP, and
2. for every problem Π' in co-NP, $\Pi' \propto_{poly} \Pi$.

The Class co-NP

Theorem.

A problem Π is NP-complete iff its complement $\overline{\Pi}$ is complete for the class co-NP.

Fact: UnSat, or Tautology, is complete for co-NP.

- Tautology is in P iff co-NP=P
- Tautology is in NP iff co-NP=NP

The Class NPI

Theorem.

If a problem Π and its complement $\overline{\Pi}$ are NP-complete, then co-NP=NP.

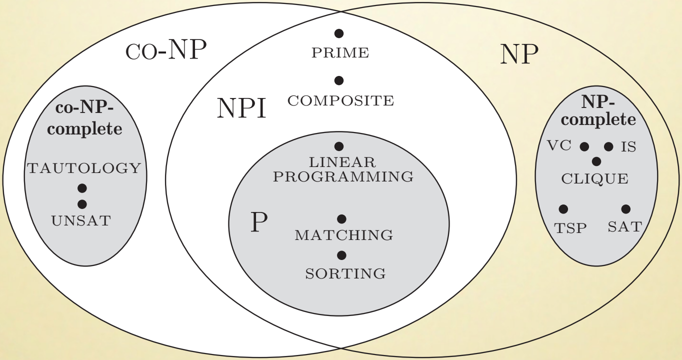
Fact: If co-NP$\neq$NP then NP$\neq$P.

The Class NPI

Let NPI = co-NP $\cap$ NP.

Clearly P $\subseteq$ NPI. It is not known if the inclusion is strict.

A (Problematic) Graph



The Class NPI

Some potential candidates turn out to be in P.

The Class NPI

Some potential candidates turn out to be in P.

Prime Number: Given an integer $k \geq 2$, is k a prime number?

The Class NPI

Some potential candidates turn out to be in P.

Prime Number: Given an integer $k \geq 2$, is k a prime number?

Fact: Prime Number and Composite Number are complement to each other. They are both in P.

The Class NPI

Some potential candidates turn out to be in P.

Prime Number: Given an integer $k \geq 2$, is k a prime number?

Fact: Prime Number and Composite Number are complement to each other. They are both in P.

A related problem is Factorization, which is not known if it is in P.

The Class NPI

A potential candidate:

The Class NPI

A potential candidate:

Graph Isomorphism: Given two graphs, G_1, G_2, are they isomorphism?

The Class NPI

A potential candidate:

Graph Isomorphism: Given two graphs, G_1, G_2, are they isomorphism?

A related problem, Graph Sub-isomorphism, is known to be NP-complete.

Exercise

[DPV07] 8.3, 8.7

[Als99]. 10.3, 10.5, 10.9, 10.19, 10.22