COL702: Advanced Data Structures and Algorithms

Ragesh Jaiswal, CSE, IITD

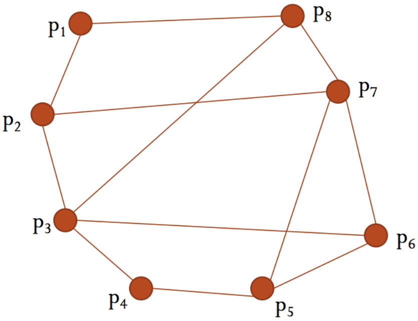
Course Overview

- Basic graph algorithms
- Algorithm Design Techniques:
  - Greedy Algorithms
  - Divide and Conquer
  - Dynamic Programming
  - Network Flow
- Computational Intractability

- Is it always possible to find a fast algorithm for any problem?

Problem

Given a social network, find the largest subset of people such that no two people in the subset are friends.



- The problem in the previous slide is called the Independent Set problem and no one knows if it can be solved in polynomial time (quickly).
- There is a whole class of problems to which Independent Set belongs.
- If you solve one problem in this class quickly, then you can solve all the problems in this class quickly.
- You can also win a million dollars!!
- We will see techniques of how to show that a new problem belongs to this class:
  - Why: *because then you can say to your boss that the new problem belongs to the difficult class of problems and even the most brilliant people in the world have not been able to solve the problem so do not expect me to do it. Also, if I can solve the problem there is no reason for me to work for you!*

Computational Intractability

Definition (Efficient Algorithms)

An algorithm is said to be *efficient* iff it runs in time polynomial in the input size. Such algorithms are also called *polynomial-time* algorithms.

Definition (Efficient Algorithms)

An algorithm is said to be *efficient* iff it runs in time polynomial in the input size. Such algorithms are also called *polynomial-time* algorithms.

- Question 1: Given a problem, does there exist an efficient algorithm to solve the problem?

Definition (Efficient Algorithms)

An algorithm is said to be *efficient* iff it runs in time polynomial in the input size. Such algorithms are also called *polynomial-time* algorithms.

- Question 1: Given a problem, does there exist an efficient algorithm to solve the problem?
- There are lots of problems arising in various fields for which this question is unresolved.
- Question 2: Are these problems related in some manner?

Definition (Efficient Algorithms)

An algorithm is said to be *efficient* iff it runs in time polynomial in the input size. Such algorithms are also called *polynomial-time* algorithms.

- Question 1: Given a problem, does there exist an efficient algorithm to solve the problem?
- There are lots of problems arising in various fields for which this question is unresolved.
- Question 2: Are these problems related in some manner?
- Question 3: If someone discovers an efficient algorithm to one of these difficult problems, then does that mean that there are efficient algorithms for other problems? If so, how do we obtain such an algorithm.

- NP-complete problems: This is a large class of problems such that all problems in this class are equivalent in the following sense:

  > *The existence of a polynomial-time algorithm for any one problem in this class implies the existence of polynomial-time algorithm for all of them.*

- NP-complete problems: This is a large class of problems such that all problems in this class are equivalent in the following sense:

  > *The existence of a polynomial-time algorithm for any one problem in this class implies the existence of polynomial-time algorithm for all of them.*

- Polynomial-time reduction:
  - Consider two problems X and Y.
  - Suppose there is a *black box* that solves arbitrary instances of problem X.
  - Suppose any arbitrary instance of problem Y can be solved using a polynomial number of standard computational steps and a polynomial number of calls to the black box that solves instance of problem X.
  - If the previous statement is true, then we say that Y is polynomial-time reducible to X. A short notation for this is $Y \leq_p X$.

- Polynomial-time reduction:
  - Consider two problems X and Y.
  - Suppose there is a *black box* that solves arbitrary instances of problem X.
  - Suppose any arbitrary instance of problem Y can be solved using a polynomial number of standard computational steps and a polynomial number of calls to the black box that solves instance of problem X.
  - If the previous statement is true, then we say that Y is polynomial-time reducible to X. A short notation for this is $Y \leq_p X$.
- Claim 1: BIPARTITE-MATCHING $\leq_p$ MAX-FLOW.

- Polynomial-time reduction:
  - Consider two problems X and Y.
  - Suppose there is a *black box* that solves arbitrary instances of problem X.
  - Suppose any arbitrary instance of problem Y can be solved using a polynomial number of standard computational steps and a polynomial number of calls to the black box that solves instance of problem X.
  - If the previous statement is true, then we say that Y is polynomial-time reducible to X. A short notation for this is $Y \leq_p X$.
- Claim 2: Suppose $Y \leq_p X$. If X can be solved in polynomial time, then Y can be solved in polynomial time.

- Polynomial-time reduction:
  - Consider two problems X and Y.
  - Suppose there is a *black box* that solves arbitrary instances of problem X.
  - Suppose any arbitrary instance of problem Y can be solved using a polynomial number of standard computational steps and a polynomial number of calls to the black box that solves instance of problem X.
  - If the previous statement is true, then we say that Y is polynomial-time reducible to X. A short notation for this is $Y \leq_p X$.
- Claim 2: Suppose $Y \leq_p X$. If X can be solved in polynomial time, then Y can be solved in polynomial time.
- Claim 3: Suppose $Y \leq_p X$. If Y cannot be solved in polynomial time, then X cannot be solved in polynomial time.

Definition (Independent Set)

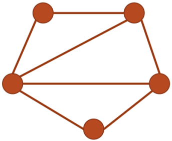
Given a graph $G = (V, E)$, a subset $I \subseteq V$ of vertices is called an independent set of G iff there are no edges between any pair of vertices in I.

Problem

<u>INDEPENDENT-SET</u>: Given a graph $G = (V, E)$ and an integer k, check if there is an independent set of size at least k in G.

Problem

<u>MAXIMUM-INDEPENDENT-SET</u>: Given a graph $G = (V, E)$, output the size of independent set of G of maximum cardinality.



Definition (Independent Set)

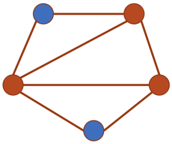
Given a graph $G = (V, E)$, a subset $I \subseteq V$ of vertices is called an independent set of G iff there are no edges between any pair of vertices in I.

Problem

INDEPENDENT-SET: Given a graph $G = (V, E)$ and an integer k, check if there is an independent set of size at least k in G.

Problem

MAXIMUM-INDEPENDENT-SET: Given a graph $G = (V, E)$, output the size of independent set of G of maximum cardinality.



Definition (Independent Set)

Given a graph $G = (V, E)$, a subset $I \subseteq V$ of vertices is called an independent set of G iff there are no edges between any pair of vertices in I.

Problem

INDEPENDENT-SET: Given a graph $G = (V, E)$ and an integer k, check if there is an independent set of size at least k in G.

Problem

MAXIMUM-INDEPENDENT-SET: Given a graph $G = (V, E)$, output the size of independent set of G of maximum cardinality.

- Claim 1: MAXIMUM-INDEPENDENT-SET $\leq_p$ INDEPENDENT-SET.

Definition (Independent Set)

Given a graph $G = (V, E)$, a subset $I \subseteq V$ of vertices is called an independent set of G iff there are no edges between any pair of vertices in I.

Problem

INDEPENDENT-SET: Given a graph $G = (V, E)$ and an integer k, check if there is an independent set of size at least k in G.

Problem

MAXIMUM-INDEPENDENT-SET: Given a graph $G = (V, E)$, output the size of independent set of G of maximum cardinality.

- Claim 1: MAXIMUM-INDEPENDENT-SET $\leq_p$ INDEPENDENT-SET.
- Claim 2: INDEPENDENT-SET $\leq_p$ MAXIMUM-INDEPENDENT-SET.

Definition (Vertex Cover)

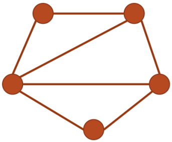
Given a graph $G = (V, E)$, a subset $S \subseteq V$ of vertices is called a vertex cover of G iff for any edge (u, v) in the graph at least one of u, v is in S.

Problem

<u>VERTEX-COVER</u>: Given a graph $G = (V, E)$ and an integer k, check if there is a vertex cover of size at most k in G.

Problem

<u>MINIMUM-VERTEX-COVER</u>: Given a graph $G = (V, E)$, output the size of vertex cover of G of minimum cardinality.



Definition (Vertex Cover)

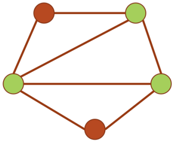
Given a graph $G = (V, E)$, a subset $S \subseteq V$ of vertices is called a vertex cover of G iff for any edge (u, v) in the graph at least one of u, v is in S.

Problem

VERTEX-COVER: Given a graph $G = (V, E)$ and an integer k, check if there is a vertex cover of size at most k in G.

Problem

MINIMUM-VERTEX-COVER: Given a graph $G = (V, E)$, output the size of vertex cover of G of minimum cardinality.



Definition (Vertex Cover)

Given a graph $G = (V, E)$, a subset $S \subseteq V$ of vertices is called a vertex cover of G iff for any edge (u, v) in the graph at least one of u, v is in S.

Problem

VERTEX-COVER: Given a graph $G = (V, E)$ and an integer k, check if there is a vertex cover of size at most k in G.

Problem

MINIMUM-VERTEX-COVER: Given a graph $G = (V, E)$, output the size of vertex cover of G of minimum cardinality.

- Claim 3: MINIMUM-VERTEX-COVER $\leq_p$ VERTEX-COVER.

Definition (Vertex Cover)

Given a graph $G = (V, E)$, a subset $S \subseteq V$ of vertices is called a vertex cover of G iff for any edge (u, v) in the graph at least one of u, v is in S.

Problem

VERTEX-COVER: Given a graph $G = (V, E)$ and an integer k, check if there is a vertex cover of size at most k in G.

Problem

MINIMUM-VERTEX-COVER: Given a graph $G = (V, E)$, output the size of vertex cover of G of minimum cardinality.

- Claim 3: MINIMUM-VERTEX-COVER $\leq_p$ VERTEX-COVER.
- Claim 4: VERTEX-COVER $\leq_p$ MINIMUM-VERTEX-COVER.

- Claim 5: INDEPENDENT-SET $\leq_p$ VERTEX-COVER.

Proof of Claim 5

- Claim 5.1: Let I be an independent set of G, then $V - I$ is a vertex cover of G.

- Claim 5: INDEPENDENT-SET $\leq_p$ VERTEX-COVER.

Proof of Claim 5

- Claim 5.1: Let I be an independent set of G, then $V - I$ is a vertex cover of G.
- Claim 5.2: Let S be a vertex cover of G, then $V - S$ is an independent set of G.

- Claim 5: INDEPENDENT-SET $\leq_p$ VERTEX-COVER.

Proof of Claim 5

- Claim 5.1: Let I be an independent set of G, then $V - I$ is a vertex cover of G.
- Claim 5.2: Let S be a vertex cover of G, then $V - S$ is an independent set of G.
- Claim 5.3: G has an independent set of size at least k if and only if G has a vertex cover of size at most $n - k$.

- Claim 5: INDEPENDENT-SET $\leq_p$ VERTEX-COVER.

Proof of Claim 5

- Claim 5.1: Let I be an independent set of G, then $V - I$ is a vertex cover of G.
- Claim 5.2: Let S be a vertex cover of G, then $V - S$ is an independent set of G.
- Claim 5.3: G has an independent set of size at least k if and only if G has a vertex cover of size at most $n - k$.
- Given an instance (G, k) of the independent set problem, create an instance $(G, n - k)$ of the vertex cover problem, make a single query to the block box for solving the vertex cover problem and return the answer that is returned by the black box. $\square$

- Claim 6: MINIMUM-VERTEX-COVER $\leq_p$ MAXIMUM-INDEPENDENT-SET.

- Claim 6: MINIMUM-VERTEX-COVER $\leq_p$ MAXIMUM-INDEPENDENT-SET.

Proof of Claim 6

- Claim 6.1: G has an independent set of size k if and only if G has a vertex cover of size $n - k$.
- Make a single call to the black box for the maximum independent problem with input G. If the black box returns k, then return $n - k$. □

- Claim 6: MINIMUM-VERTEX-COVER $\leq_p$ MAXIMUM-INDEPENDENT-SET.

Proof of Claim 6

- Claim 6.1: G has an independent set of size k if and only if G has a vertex cover of size $n - k$.
- Make a single call to the black box for the maximum independent problem with input G. If the black box returns k, then return $n - k$. $\square$

Another proof of Claim 6

- MINIMUM-VERTEX-COVER $\leq_p$ VERTEX-COVER
- VERTEX-COVER $\leq_p$ INDEPENDENT-SET
- INDEPENDENT-SET $\leq_p$ MAXIMUM-INDEPENDENT-SET $\square$

Theorem

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

Problem

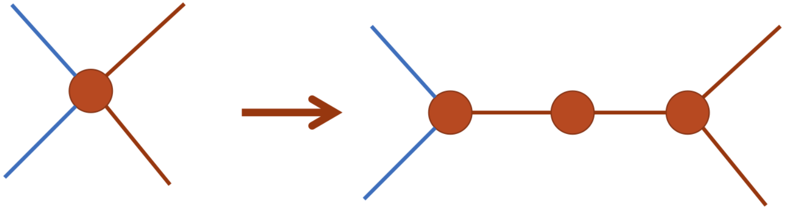
DEG-3-INDEPENDENT-SET: Given a graph $G = (V, E)$ of bounded degree 3 *(i.e., all vertices have degree* ≤ 3*)* and an integer k, check if there is an independent set of size at least k in G.

- Claim 1: INDEPENDENT-SET $\leq_p$ DEG-3-INDEPENDENT-SET

Problem

DEG-3-INDEPENDENT-SET: Given a graph $G = (V, E)$ of bounded degree 3 *(i.e., all vertices have degree* ≤ 3*)* and an integer k, check if there is an independent set of size at least k in G.

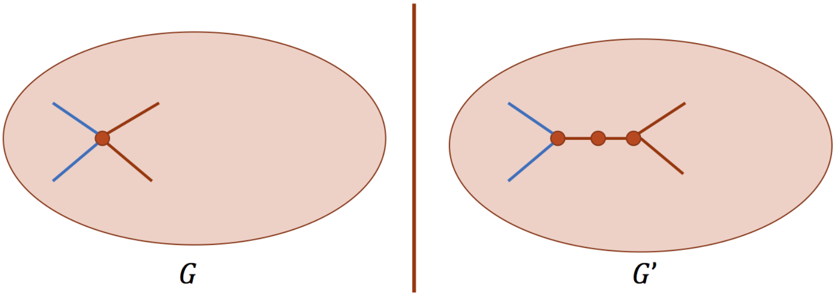
- Claim 1: INDEPENDENT-SET $\leq_p$ DEG-3-INDEPENDENT-SET
  - Idea: "Split" all vertices.



- Claim 1: INDEPENDENT-SET $\leq_p$ DEG-3-INDEPENDENT-SET

Proof of Claim 1

- Consider graph G' constructed by "splitting" a vertex of G.
- Claim 1.1: G has an independent set of size at least k if and only if G' has an independent set of size at least $(k+1)$.



Problem

SET-COVER: Given a set U of n elements, a collection $S_1, ..., S_m$ of subsets of U, and an integer k, determine if there exist a collection of at most k of these sets whose union is equal to U.

Problem

SET-COVER: Given a set U of n elements, a collection $S_1, ..., S_m$ of subsets of U, and an integer k, determine if there exist a collection of at most k of these sets whose union is equal to U.

- Claim 1: VERTEX-COVER $\leq_p$ SET-COVER.

Definition

- Boolean variables: 0-1 (true/false) variables.
- Term: A variable or its negation is called a term.
- Clause: Disjunction of terms (e.g., $(x_1 \vee \bar{x}_2 \vee x_3)$)
- Assignment: Fixing 0-1 values for each variables.
- Satisfying assignment: An assignment of variables is called a satisfying assignment for a collection of clauses if all clauses evaluate to 1 (true).
  - For example, $(x_1 \vee \bar{x}_2), (x_2 \vee \bar{x}_3), (x_3 \vee \bar{x}_1)$

Problem

SAT: Given a set of clauses $C_1, ..., C_m$ over a set of variables $x_1, ..., x_n$ determine if there exists a satisfying assignment.

Problem

SAT: Given a set of clauses $C_1, ..., C_m$ over a set of variables $x_1, ..., x_n$ determine if there exists a satisfying assignment.

Problem

3-SAT: Given a set of clauses $C_1, ..., C_m$ each of length at most 3, over a set of variables $x_1, ..., x_n$ determine if there exists a satisfying assignment.

Problem

SAT: Given a set of clauses $C_1, ..., C_m$ over a set of variables $x_1, ..., x_n$ determine if there exists a satisfying assignment.

Problem

3-SAT: Given a set of clauses $C_1, ..., C_m$ each of length at most 3, over a set of variables $x_1, ..., x_n$ determine if there exists a satisfying assignment.

- Claim 1: SAT $\leq_p$ 3-SAT

Problem

SAT: Given a set of clauses $C_1, ..., C_m$ over a set of variables $x_1, ..., x_n$ determine if there exists a satisfying assignment.

Problem

3-SAT: Given a set of clauses $C_1, ..., C_m$ each of length at most 3, over a set of variables $x_1, ..., x_n$ determine if there exists a satisfying assignment.

- Claim 1: SAT $\leq_p$ 3-SAT
  - Main idea: $(t_1 \vee t_2 \vee t_3 \vee t_4) \equiv ((t_1 \vee t_2 \vee z), (z \equiv t_3 \vee t_4))$

Problem

3-SAT: Given a set of clauses $C_1, ..., C_m$ each of length at most 3, over a set of variables $x_1, ..., x_n$ determine if there exists a satisfying assignment.

Problem

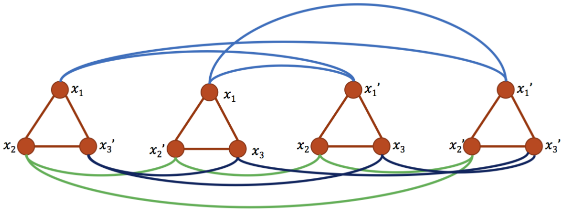
INDEPENDENT-SET: Given a graph $G = (V, E)$ and an integer k, check if there is an independent set of size at least k in G.

- Claim 1: 3-SAT $\leq_p$ INDEPENDENT-SET

- Claim 1: 3-SAT $\leq_p$ INDEPENDENT-SET

Proof sketch of Claim 1

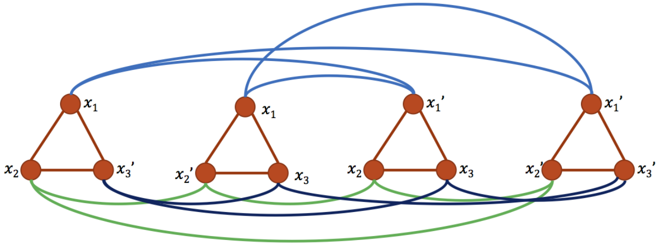
- Given an instance of the 3-SAT problem $(C_1, ..., C_m)$, we will construct an instance (G, m) of the INDEPENDENT-SET problem.
- We will then show that $(C_1, ..., C_m)$ has a satisfying assignment if and only if G has an independent set of size at least m.
- Consider an example construction:
  - 3-SAT instance: $(x_1 \vee x_2 \vee \bar{x}_3), (x_1 \vee \bar{x}_2 \vee x_3), (\bar{x}_1 \vee x_2 \vee x_3), (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)$
  - INDEPENDENT-SET instance (G, m) for the above shown below:



- Claim 1: 3-SAT $\leq_p$ INDEPENDENT-SET

Proof sketch of Claim 1

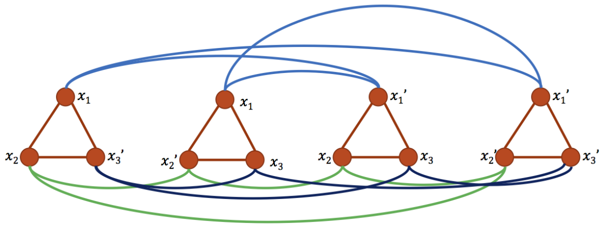
- Consider an example construction:
  - 3-SAT instance:
    $(x_1 \vee x_2 \vee \bar{x}_3), (x_1 \vee \bar{x}_2 \vee x_3), (\bar{x}_1 \vee x_2 \vee x_3), (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)$
  - INDEPENDENT-SET instance (G, m) for the above shown below:
  - Claim 1.1: If (C_1, C_2, C_3, C_4) has a satisfying assignment, then G has an independent set of size 4.



- Claim 1: 3-SAT $\leq_p$ INDEPENDENT-SET

Proof sketch of Claim 1

- Consider an example construction:
  - 3-SAT instance: $(x_1 \vee x_2 \vee \bar{x}_3), (x_1 \vee \bar{x}_2 \vee x_3), (\bar{x}_1 \vee x_2 \vee x_3), (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)$
  - INDEPENDENT-SET instance (G, m) for the above shown below:
  - Claim 1.1: If (C_1, C_2, C_3, C_4) has a satisfying assignment, then G has an independent set of size 4.
  - Claim 1.2: If G has an independent set of size 4, then (C_1, C_2, C_3, C_4) has a satisfying assignment.



- Claim 1: 3-SAT $\leq_p$ INDEPENDENT-SET
- Claim 2: SAT $\leq_p$ INDEPENDENT-SET

- Claim 1: 3-SAT $\leq_p$ INDEPENDENT-SET
- Claim 2: SAT $\leq_p$ INDEPENDENT-SET
  - Since SAT $\leq_p$ 3-SAT $\leq_p$ INDEPENDENT-SET

- Claim 1: 3-SAT $\leq_p$ INDEPENDENT-SET
- Claim 2: SAT $\leq_p$ INDEPENDENT-SET
  - Since SAT $\leq_p$ 3-SAT $\leq_p$ INDEPENDENT-SET
- Claim 3: SAT $\leq_p$ SET-COVER

- Claim 1: 3-SAT $\leq_p$ INDEPENDENT-SET
- Claim 2: SAT $\leq_p$ INDEPENDENT-SET
  - Since SAT $\leq_p$ 3-SAT $\leq_p$ INDEPENDENT-SET
- Claim 3: SAT $\leq_p$ SET-COVER
  - Since SAT $\leq_p$ 3-SAT $\leq_p$ INDEPENDENT-SET $\leq_p$ VERTEX-COVER $\leq_p$ SET-COVER

Computational Intractability: NP and NP-complete

- We said that the problems INDEPENDENT-SET, VERTEX-COVER, SAT seem hard.

- We said that the problems INDEPENDENT-SET, VERTEX-COVER, SAT seem hard.
- Polynomial-time reductions just give pair-wise relationships between problems.
- Is there a common theme that binds all these problems in one computational class?

- We said that the problems INDEPENDENT-SET, VERTEX-COVER, SAT seem hard.
- Polynomial-time reductions just give pair-wise relationships between problems.
- Is there a common theme that binds all these problems in one computational class?
- Let us try to extract a theme that is common to some of the problems we saw:
  - INDEPENDENT-SET: Given (G, k), determine if G has an independent set of size at least k.
  - VERTEX-COVER: Given (G, k), determine if G has a vertex cover of size at most k.
  - SAT: Given a Boolean formula Ω in CNF, determine if the formula is satisfiable.

- Let us try to extract a theme that is common to some of the problems we saw:
  - INDEPENDENT-SET: Given (G, k), determine if G has an independent set of size at least k.
    - Suppose there is an independent set of size at least k and someone gives such a subset as a certificate. Then we can verify this certificate quickly.
  - VERTEX-COVER: Given (G, k), determine if G has a vertex cover of size at most k.
    - Suppose there is a vertex cover of size at most k and someone gives such a subset as a certificate. Then we can verify this certificate quickly.
  - SAT: Given a Boolean formula Ω in CNF, determine if the formula is satisfiable.
    - Suppose the formula Ω is satisfiable and someone gives such a satisfying assignment as a certificate. Then we can verify this certificate quickly.

- Problem encoding and algorithm:
  - An *instance* of a problem can be encoded using a finite string s.
  - A *decision* problem X can be thought of as a set of strings on which the answer is true (or 1).
  - We say that an algorithm A solves a problem X if for all strings s, $A(s) = 1$ if and only if s is in X.
  - We say that an algorithm A has a polynomial running time if there is a polynomial p such that for every string s, A terminates on input s in at most $O(p(|s|))$ steps.

- Efficient Certification:
  - We say that algorithm B is an efficient certifier for a problem X, iff the following holds:
    - B is a polynomial time algorithm that takes two input string s and t.
    - There is a polynomial p such that for every string s, we have $s \in X$ if and only if there exists a string t such that $|t| \leq p(|s|)$ and $B(s, t) = 1$.

- Efficient Certification:
  - We say that algorithm B is an efficient certifier for a problem X, iff the following holds:
    - B is a polynomial time algorithm that takes two input string s and t.
    - There is a polynomial p such that for every string s, we have $s \in X$ if and only if there exists a string t such that $|t| \leq p(|s|)$ and $B(s, t) = 1$.
- Note that B does not solve the problem but only verifies a proposed solution.
- Can we use B to solve the problem?

- Efficient Certification:
  - We say that algorithm B is an efficient certifier for a problem X, if the following holds:
    - B is a polynomial time algorithm that takes two input string s and t.
    - There is a polynomial p such that for every string s, we have $s \in X$ if and only if there exists a string t such that $|t| \leq p(|s|)$ and $B(s, t) = 1$.
- Note that B does not solve the problem but only verifies a proposed solution.
- Can we use B to solve the problem? Yes
- Can we use B to solve the problem efficiently?

- Efficient Certification:
  - We say that algorithm B is an efficient certifier for a problem X, if the following holds:
    - B is a polynomial time algorithm that takes two input string s and t.
    - There is a polynomial p such that for every string s, we have $s \in X$ if and only if there exists a string t such that $|t| \leq p(|s|)$ and $B(s, t) = 1$.
- Note that B does not solve the problem but only verifies a proposed solution.
- Can we use B to solve the problem? Yes
- Can we use B to solve the problem efficiently?

Definition (NP)

A problem is said to be in NP iff there exists an efficient certification algorithm for the problem.

- Efficient Certification:
  - We say that algorithm B is an efficient certifier for a problem X, if the following holds:
    - B is a polynomial time algorithm that takes two input string s and t.
    - There is a polynomial p such that for every string s, we have $s \in X$ if and only if there exists a string t such that $|t| \leq p(|s|)$ and $B(s, t) = 1$.

Definition (NP)

A problem is said to be in NP iff there exists an efficient certification algorithm for the problem.

- NP stands for Non-deterministic Polynomial time.
  - Non-deterministic algorithms are allowed to make non-deterministic choices (guesswork). Such algorithms can guess the certificate t for an instance s.

Definition (NP)

A problem is said to be in NP iff there exists an efficient certification algorithm for the problem.

Definition (P)

A problem is said to be in P iff there exists an efficient algorithm that solves the problem.

- Theorem: P $\subseteq$ NP.

Definition (NP)

A problem is said to be in NP iff there exists an efficient certification algorithm for the problem.

Definition (P)

A problem is said to be in P iff there exists an efficient algorithm that solves the problem.

- Theorem: P $\subseteq$ NP.
- Claim 1: INDEPENDENT-SET $\in$ NP
  - Proof sketch: The certificate is an independent set of size at least k. The certifier checks if the given set if indeed an independent set of size at least k.

Definition (NP)

A problem is said to be in NP iff there exists an efficient certification algorithm for the problem.

Definition (P)

A problem is said to be in P iff there exists an efficient algorithm that solves the problem.

- Theorem: P $\subseteq$ NP.
- Claim 1: INDEPENDENT-SET $\in$ NP
  - Proof sketch: The certificate is an independent set of size at least k. The certifier checks if the given set if indeed an independent set of size at least k.
- Claim 2: SAT $\in$ NP
  - Proof sketch: The certificate is a satisfying assignment.The certifier checks if the assignment makes all clauses true.

Definition (NP)

A problem is said to be in NP iff there exists an efficient certification algorithm for the problem.

Definition (P)

A problem is said to be in P iff there exists an efficient algorithm that solves the problem.

- Theorem: $P \subseteq NP$.
- Is $P = NP$?
- What are the hardest problems in NP?

Definition (NP)

A problem is said to be in NP iff there exists an efficient certification algorithm for the problem.

Definition (P)

A problem is said to be in P iff there exists an efficient algorithm that solves the problem.

- Theorem: P $\subseteq$ NP.
- Is P $=$ NP?
- What are the hardest problems in NP?
- A problem $X \in$ NP is the hardest problem in NP if for all problems $Y \in$ NP, $Y \leq_p X$.
- Such problems are called NP-complete problems.

Definition (NP)

A problem is said to be in NP iff there exists an efficient certification algorithm for the problem.

Definition (P)

A problem is said to be in P iff there exists an efficient algorithm that solves the problem.

Definition (NP-complete)

A problem X is said to be NP-complete iff the following two properties hold:

1. $X \in \text{NP}$.
2. For all $Y \in \text{NP}$, $Y \leq_p X$.

Definition (NP)

A problem is said to be in NP iff there exists an efficient certification algorithm for the problem.

Definition (P)

A problem is said to be in P iff there exists an efficient algorithm that solves the problem.

Definition (NP-complete)

A problem X is said to be NP-complete iff the following two properties hold:

1. $X \in \text{NP}$.
2. For all $Y \in \text{NP}$, $Y \leq_p X$.

- How do we show that there is a problem that is NP-complete?

Definition (NP)

A problem is said to be in NP iff there exists an efficient certification algorithm for the problem.

Definition (P)

A problem is said to be in P iff there exists an efficient algorithm that solves the problem.

Definition (NP-complete)

A problem X is said to be NP-complete iff the following two properties hold:

1. $X \in$ NP.
2. For all $Y \in$ NP, $Y \leq_p X$.

- How do we show that there is a problem that is NP-complete?
- Suppose by some magic we have shown that SAT is NP-complete, does that mean that there are more NP-complete problems?

Definition (NP)

A problem is said to be in NP iff there exists an efficient certification algorithm for the problem.

Definition (P)

A problem is said to be in P iff there exists an efficient algorithm that solves the problem.

Definition (NP-complete)

A problem X is said to be NP-complete iff the following two properties hold:

1. $X \in$ NP.
2. For all $Y \in$ NP, $Y \leq_p X$.

Theorem (Cook-Levin Theorem)

3-SAT is NP*-complete.*

Definition (NP-complete)

A problem X is said to be NP-complete iff the following two properties hold:

1. $X \in \text{NP}$.
2. For all $Y \in \text{NP}$, $Y \leq_p X$.

Theorem (Cook-Levin Theorem)

3-SAT is NP*-complete.*

Proof sketch

- Claim 1: CIRCUIT-SAT is NP-complete.
- Claim 2: CIRCUIT-SAT $\leq_p$ 3-SAT.

Theorem (Cook-Levin Theorem)

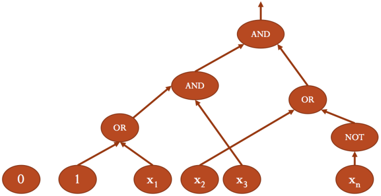
3-*SAT is* NP-*complete.*

Proof sketch

- Claim 1: CIRCUIT-SAT is NP-complete.
- Claim 2: CIRCUIT-SAT $\leq_p$ 3-SAT.

- Circuit: A directed acyclic graph where each node is either:
  - Constant nodes: Labeled 0/1
  - Input nodes: These denote the variables
  - Gates: AND, OR, and NOT

  There is a single output node.



Theorem (Cook-Levin Theorem)

3-SAT is NP*-complete.*

Proof sketch

- Claim 1: CIRCUIT-SAT is NP-complete.
- Claim 2: CIRCUIT-SAT $\leq_p$ 3-SAT.

- Circuit: A directed acyclic graph where each node is either:
  - Constant nodes: Labeled 0/1
  - Input nodes: These denote the variables
  - Gates: AND, OR, and NOT

  There is a single output node.

Problem

CIRCUIT-SAT: Given a circuit, determine if there is an input such that the output of the circuit is 1.

Theorem (Cook-Levin Theorem)

3-SAT is NP-complete.

Proof sketch

- Claim 1: CIRCUIT-SAT is NP-complete.
  - Fact: For every algorithm that runs in time polynomial in the input size n, there is an equivalent circuit of size polynomial in n.
  - Given an input instance s of any NP problem X, consider the equivalent circuit for the efficient certifier of X. The input gates of this circuit has s and t.
  - $s \in X$ if and only if this circuit is satisfiable.
- Claim 2: CIRCUIT-SAT $\leq_p$ 3-SAT.
  - For any circuit, we can write an equivalent 3-SAT formula.

Problem

CIRCUIT-SAT: Given a circuit, determine if there is an input such that the output of the circuit is 1.

End