CSL 356: Analysis and Design of Algorithms

Ragesh Jaiswal

CSE, IIT Delhi

Computational Intractability: Defining NP

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

- **NP**: This is the set of all problems for which there exists an efficient certification algorithm.
- **NP** stands for **N**on-deterministic **P**olynomial time:
  - Non-deterministic algorithms are allowed to make non-deterministic choices (guesswork). Such algorithms can guess the certificate t for an instance s.

Computational Intractability: Defining NP

- **NP**: This is the set of all problems for which there exists an efficient certification algorithm.
- **P**: This is the set of all problems for which there exists an efficient algorithm that solves the problem. **P** stands for polynomial time.
- Theorem: **P** is contained in **NP**.
- Claim 1: Independent-set is in **NP**.

Computational Intractability: Defining NP

- **NP**: This is the set of all problems for which there exists an efficient certification algorithm.
- **P**: This is the set of all problems for which there exists an efficient algorithm that solves the problem. **P** stands for polynomial time.
- Theorem: **P** is contained in **NP**.
- Claim 1: Independent-set is in **NP**.
  - Proof: The certificate is an independent set of size at least k. The certifier just checks if the given set is indeed an independent set.

Computational Intractability: Defining NP

- **NP**: This is the set of all problems for which there exists an efficient certification algorithm.
- **P**: This is the set of all problems for which there exists an efficient algorithm that solves the problem. **P** stands for polynomial time.
- Theorem: **P** is contained in **NP**.
- Claim 1: Independent-set is in **NP**.
  - Proof: The certificate is an independent set of size at least k. The certifier just checks if the given set is indeed an independent set.
- Claim 2: SAT is in **NP**.

Computational Intractability: Defining NP

- **NP**: This is the set of all problems for which there exists an efficient certification algorithm.
- **P**: This is the set of all problems for which there exists an efficient algorithm that solves the problem. **P** stands for polynomial time.
- Theorem: **P** is contained in **NP**.
- Claim 1: Independent-set is in **NP**.
  - Proof: The certificate is an independent set of size at least k. The certifier just checks if the given set is indeed an independent set.
- Claim 2: SAT is in **NP**.
  - Proof: The certificate is an assignment. The certifier checks if this assignment makes all clauses true.

Computational Intractability: NP Completeness

- Is **P** = **NP**?

Computational Intractability: NP Completeness

- 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.

Computational Intractability: NP Completeness

- 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.
- NP-complete: These are all problems X with the following properties:
  1. X is in **NP**.
  2. For all Y in **NP**, $Y \leq_p X$.

Computational Intractability: NP Completeness

- 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.
- NP-complete: These are all problems X with the following properties:
  1. X is 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?

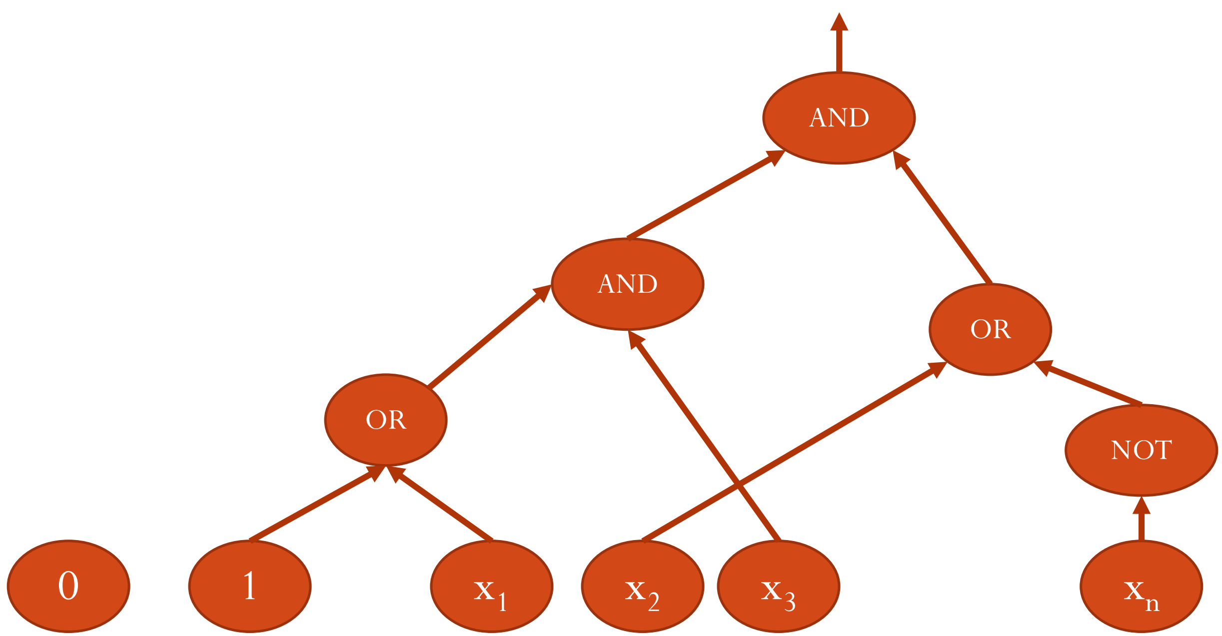
Computational Intractability: NP & NP-complete

- Theorem(Cook-Levin): 3-SAT is **NP**-complete.
  - Proof idea:
    - Claim 1: Circuit-SAT is **NP**-complete.
    - Claim 2: Circuit-SAT $\leq_p$ 3-SAT.

Computational Intractability: NP Completeness

- 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.



Computational Intractability: NP Completeness

- 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.

Computational Intractability: NP Completeness

- 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.
- Claim 1: Circuit-SAT is **NP**-complete.
- Fact: For every algorithm that runs in time polynomial in the inputs size n, there is a circuit of size polynomial in n such that the output of both are the same.

Computational Intractability: NP Completeness

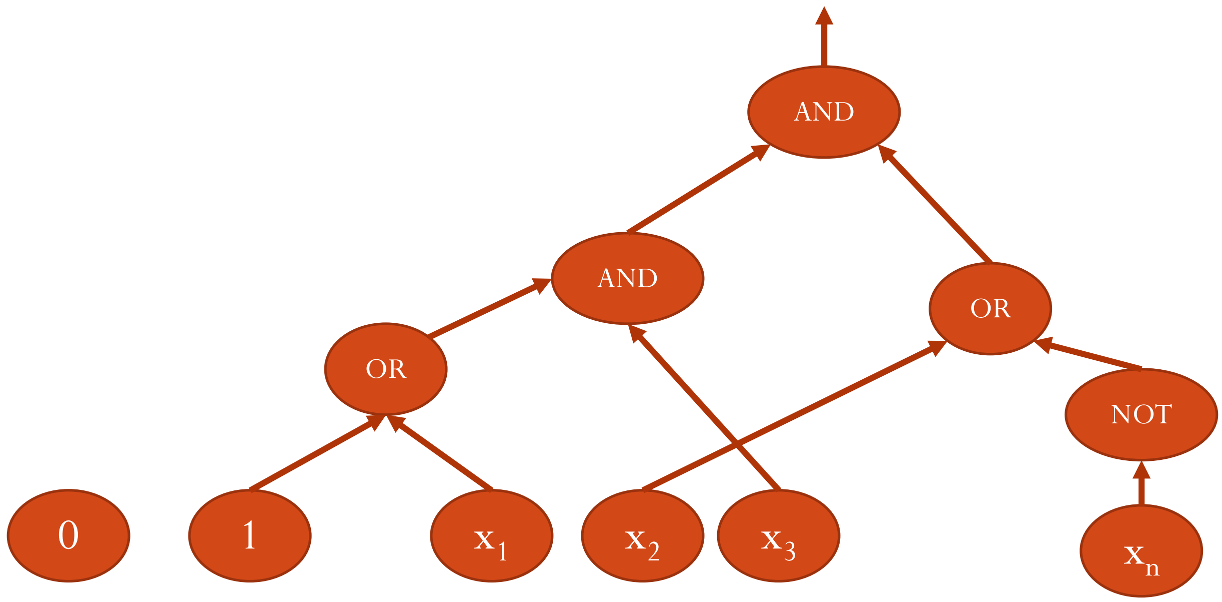
- Problem(Circuit-satisfiability): Given a circuit determine if there is an input such that the output of the circuit is 1.
- Claim 1: Circuit-SAT is **NP**-complete.
- Fact: For every algorithm that runs in time polynomial in the inputs size n, there is an equivalent circuit of size polynomial in n.
- Proof of Claim 1: ?

Computational Intractability: NP Completeness

- Problem(Circuit-satisfiability): Given a circuit determine if there is an input such that the output of the circuit is 1.
- Claim 1: Circuit-SAT is **NP**-complete.
- Fact: For every algorithm that runs in time polynomial in the inputs size n, there is an equivalent circuit of size polynomial in n.
- Proof of Claim 1: 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.
  - Claim 1.1: If s is in X, the circuit is satisfiable.
  - Claim 1.2: If s is not in X, then the circuit is not satisfiable.

Computational Intractability: NP Completeness

- Problem(Circuit-satisfiability): Given a circuit determine if there is an input such that the output of the circuit is 1.
- Claim 1: Circuit-SAT is **NP**-complete.
- Claim 2: Circuit-SAT $\leq_p$ 3-SAT.
  - Proof: For any circuit, we can write an equivalent 3-SAT formula.



Computational Intractability

NP and NP-completeness

Computational Intractability: NP & NP-complete

- **<u>NP</u>**: A problem X is in **NP** if and only if there is an *efficient certifier* for X.
- **<u>NP</u>**<u>-complete</u>: These are all problems X with the following properties:
  1. X is in **NP**.
  2. For all Y in **NP**, $Y \leq_p X$.
- <u>Theorem (Cook-Levin)</u>: 3-SAT is **NP**-complete.
- **<u>NP</u>**<u>-hard</u>: These are all problems X with the following property:
  1. For all Y in **NP**, $Y \leq_p X$.
  - <u>Example</u>: Given a graph G, find the maximum independent set in G.

Computational Intractability

NP-complete problems

Computational Intractability: NP-complete problems

- We now know that 3-SAT is **NP**-complete.
- <u>Claim</u>: Independent-set, Vertex-cover, Set-cover are also **NP**-complete.

Computational Intractability: NP-complete problems

- We now know that 3-SAT is **NP**-complete.
- Claim: Independent-set, Vertex-cover, Set-cover are also **NP**-complete.
  - Proof: These problem are in **NP** and

    3-SAT $\leq_p$ Independent-set $\leq_p$ Vertex-cover $\leq_p$ Set-cover.

Computational Intractability: NP-complete problems

- Problem (TSP): Given a *complete,* weighted, directed graph G and a number k, determine if there is a tour in the graph of total length at most k.
- Claim 1: TSP is in **NP**.
  - Proof: A tour of length at most k is a certificate.

Computational Intractability: NP-complete problems

- Problem (TSP): Given a *complete*, weighted, directed graph G and a number k, determine if there is a tour in the graph of total length at most k.
- Claim 1: TSP is in **NP**.
  - Proof: A tour of length at most k is a certificate.
- Claim 2: 3-SAT $\leq_p$ TSP
  - Proof:
    - Claim 2.1: 3-SAT $\leq_p$ Hamiltonian-cycle.
    - Claim 2.2: Hamiltonian-cycle $\leq_p$ TSP.

Computational Intractability: NP-complete problems

- Problem (TSP): Given a *complete*, weighted, directed graph G and a number k, determine if there is a tour in the graph of total length at most k.
- Claim 1: TSP is in **NP**.
  - Proof: A tour of length at most k is a certificate.
- Claim 2: 3-SAT $\leq_p$ TSP
  - Proof:
    - Claim 2.1: 3-SAT $\leq_p$ Hamiltonian-cycle.
    - Claim 2.2: Hamiltonian-cycle $\leq_p$ TSP.
- Problem (Hamiltonian-cycle): Given an unweighted, directed graph, determine if there is a Hamiltonian cycle in the graph.
  - Hamiltonian cycle: A cycle that visits each vertex exactly once.

Computational Intractability: NP-complete problems

- Claim 2.2: Hamiltonian-cycle $\leq_p$ TSP.
  - Proof: Given a unweighted, directed graph G, construct the following complete, directed, weighted graph G:
    - For each edge (u, v) in G give the weight of 1 to edge (u, v) in G'.
    - For each pair (u, v) such that there is no edge from u to v in G, add an edge (u, v) with weight 2 in G'.
    - Claim2.2.1: G has a Hamiltonian cycle if and only if G' has a tour of size at most n.

End