CSL 356: Analysis and Design of Algorithms

Ragesh Jaiswal

CSE, IIT Delhi

Polynomial-time reductions: Examples

3-SAT Vs Independent-set

Computational Intractability: Reduction

- Problem(3-SAT): Given a set of clauses $C_1, \ldots, C_m$, each of length at most 3, over a set of variables $x_1, \ldots, x_n$, does there exist 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: 3-SAT $\leq_p$ Independent-set

Computational Intractability: Reduction

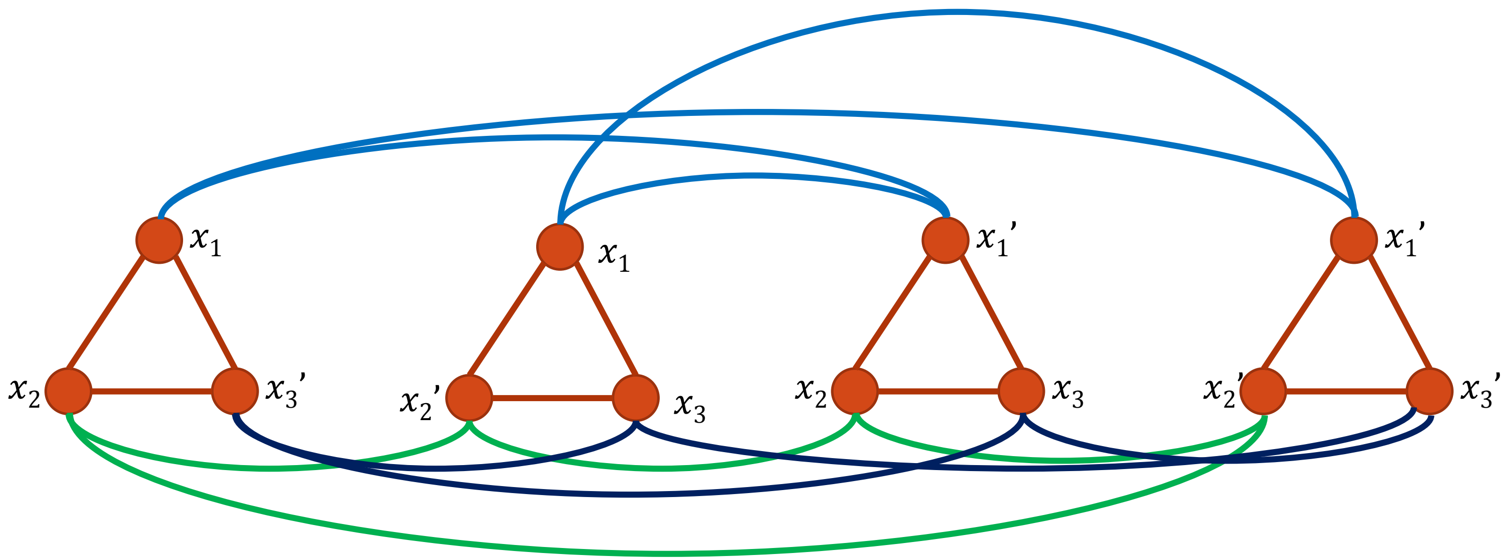
- Claim: 3-SAT $\leq_p$ Independent-set
- Proof idea:
  - Given an instance of the 3-SAT problem $(C_1, \ldots, C_m)$, construct an instance of the Independent-set problem (G, m).
  - Then show that $(C_1, \ldots, C_m)$ has a satisfying assignment if and only if G has an independent set of size at least m.

Computational Intractability: Reduction

- Claim: 3-SAT $\leq_p$ Independent-set
- Proof idea:
  - Given an instance of the 3-SAT problem $(C_1, \ldots, C_m)$, construct an instance of the Independent-set problem (G, m).
  - Then show that $(C_1, \ldots, C_m)$ has a satisfying assignment if and only if G has an independent set of size at least m.
  - Example:
    - 3-SAT instance: $(x_1 \vee x_2 \vee x_3'), (x_1 \vee x_2' \vee x_3), (x_1' \vee x_2 \vee x_3), (x_1' \vee x_2' \vee x_3')$
    - Independent-set instance: (G, m)

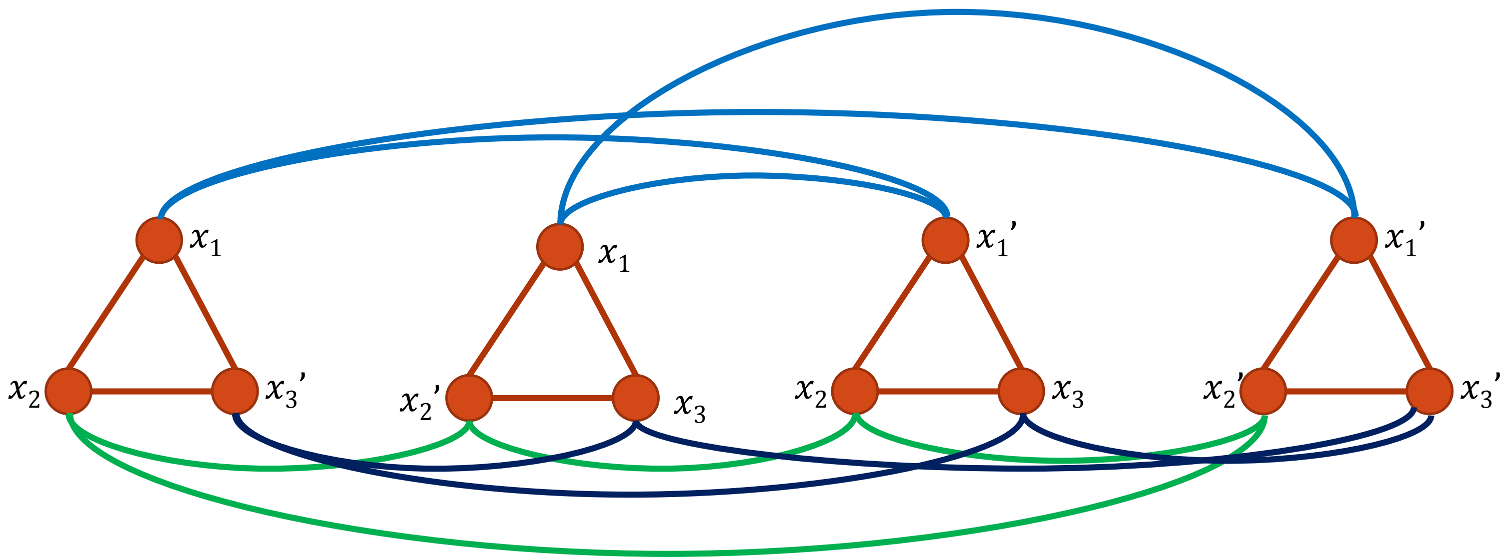
Computational Intractability: Reduction

- Claim: 3-SAT $\leq_p$ Independent-set
- Proof idea:
  - Given an instance of the 3-SAT problem $(C_1, \ldots, Cm)$, construct an instance of the Independent-set problem (G, m).
  - Then show that $(C_1, \ldots, Cm)$ has a satisfying assignment if and only if G has an independent set of size at least m.
  - Example:
    - 3-SAT instance: $(x_1 \vee x_2 \vee x_3'), (x_1 \vee x_2' \vee x_3), (x_1' \vee x_2 \vee x_3), (x_1' \vee x_2' \vee x_3')$
    - Independent-set instance: (G, m)



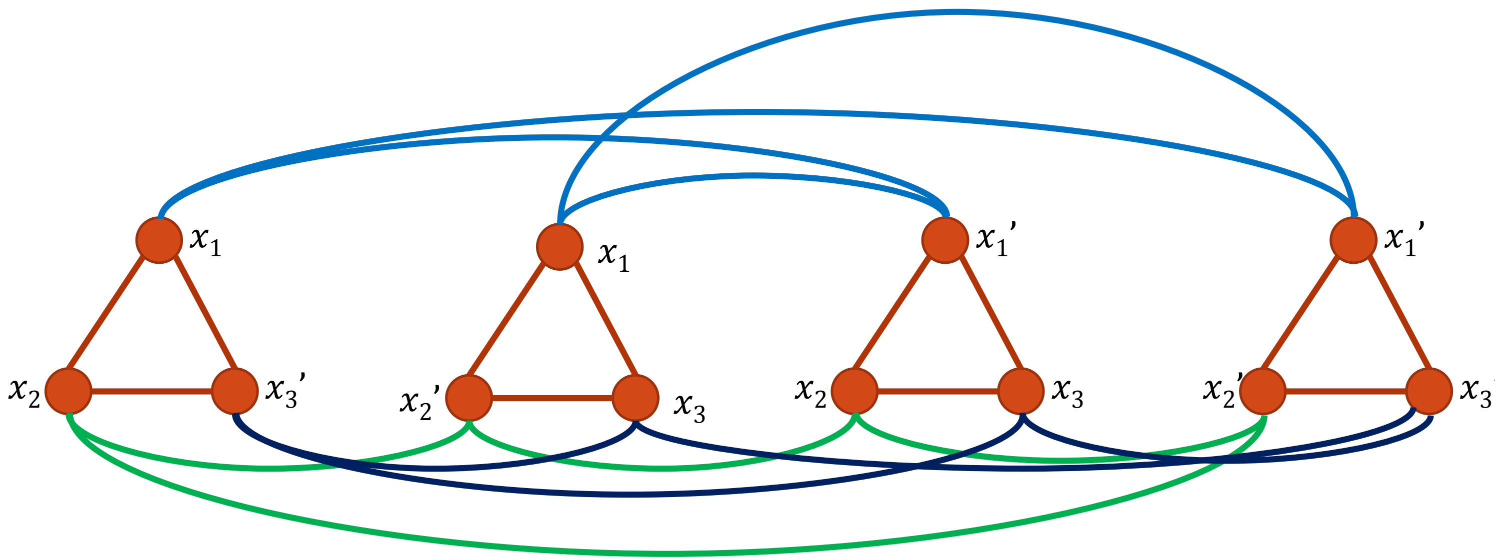
Computational Intractability: Reduction

- Claim: 3-SAT $\leq_p$ Independent-set
- Proof idea:
  - Example:
    - 3-SAT instance: $(x_1 \vee x_2 \vee x_3'), (x_1 \vee x_2' \vee x_3), (x_1' \vee x_2 \vee x_3), (x_1' \vee x_2' \vee x_3')$
    - Independent-set instance: (G, m)
    - Claim 1: If (C_1, C_2, C_3, C_4) has a satisfying assignment, then G has an independent set of size 4.



Computational Intractability: Reduction

- Claim: 3-SAT $\leq_p$ Independent-set
- Proof idea:
  - Example:
    - 3-SAT instance: $(x_1 \vee x_2 \vee x_3'), (x_1 \vee x_2' \vee x_3), (x_1' \vee x_2 \vee x_3), (x_1' \vee x_2' \vee x_3')$
    - Independent-set instance: (G, m)
    - Claim 2: If G has an independent set of size 4, then (C_1, C_2, C_3, C_4) has a satisfying assignment.



Computational Intractability: Reduction

- Claim: SAT $\leq_p$ Independent-set
  - Proof: SAT $\leq_p$ 3-SAT $\leq_p$ Independent-set

Computational Intractability: Reduction

- Claim: SAT $\leq_p$ Independent-set
  - Proof: SAT $\leq_p$ 3-SAT $\leq_p$ Independent-set.

- Claim: SAT $\leq_p$ Set-cover

Computational Intractability: Reduction

- Claim: SAT $\leq_p$ Independent-set
  - Proof: SAT $\leq_p$ 3-SAT $\leq_p$ Independent-set.

- Claim: SAT $\leq_p$ Set-cover
  - Proof:
    - SAT $\leq_p$ 3-SAT
    - 3-SAT $\leq_p$ Independent-set
    - Independent-set $\leq_p$ Vertex-cover
    - Vertex-cover $\leq_p$ Set-cover

Computational Intractability

NP and NP-complete

Computational Intractability: Reductions

- 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 correctly.
  - We say that Y is polynomial time reducible to X ($Y \leq_p X$).
- Examples:
  - Independent-set $\leq_p$ Vertex-cover
  - SAT $\leq_p$ Independent-set

Computational Intractability: Reductions

- We said that the problems Independent-set, Vertex-cover, SAT seem hard.
- Are there some instances of these problems that are easy?

Computational Intractability: Reductions

- We said that the problems Independent-set, Vertex-cover, SAT seem hard.
- Are there some instances of these problems that are easy?
- <u>2-Independent-set</u>: Given (G, k) such that the degree of each vertex in G is at most 2, determine if there is an independent set of size at least k.
- Is Independent-set $\leq_p$ 2-Independent-set?
- Can you solve the 2-Independent-set in polynomial time?

Computational Intractability: Reductions

- We said that the problems Independent-set, Vertex-cover, SAT seem hard.
- Are there some instances of these problems that are easy?
- 2-Independent-set: Given (G, k) such that the degree of each vertex in G is at most 2, determine if there is an independent set of size at least k.
- Is Independent-set $\leq_p$ 2-Independent-set?
- Can you solve the 2-Independent-set in polynomial time?
- 2-Vertex-cover: Given (G, k) such that the degree of each vertex in G is at most 2, determine if there is an independent set of size at most k.
- 2-SAT: Given a Boolean formula in CNF form such that each clause has at most 2 terms. Determine if the formula is satisfiable.

Computational Intractability: Reductions

- Reductions just give pair-wise relationships between problems.
- Is there a common *theme* that binds all these problems into one *computational class*.
- Let us try to extract a theme that is common to the problems we looked at:
  - 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 conjunctive normal form, determine if the formula is satisfiable.

Computational Intractability: Defining NP

- Let us try to extract a theme that is common to the problems we looked at:
  - 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 conjunctive normal form, 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.

Computational Intractability: Defining NP

- Problem encoding and algorithm:
  - An *instance* of a problem can be encoded using a finite bit 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 polynomial running time if there is a polynomial p such that for every string s, A terminates on s in at most $O(p(|s|))$ steps.

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 $B(s, t) = 1$.
- B does not solve the problem but only verifies a proposed solution correctly.
- Can we use B to solve the problem?

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 $B(s,t) = 1$.

- B does not solve the problem but only verifies a proposed solution correctly.
- Can we use B to solve the problem?
- Can we use B to solve the problem efficiently?

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 $B(s,t) = 1$.

- B does not solve the problem but only verifies a proposed solution correctly.
- Can we use B to solve the problem?
- Can we use B to solve the problem efficiently?
- **NP**: This is the set of all problems for which there exists an efficient certification algorithm.

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 $\mathbf{P} = \mathbf{NP}$?

Computational Intractability: NP Completeness

- Is $\mathbf{P} = \mathbf{NP}$?
- 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?

End