CSL 356: Analysis and Design of Algorithms

Ragesh Jaiswal

CSE, IIT Delhi

Topics

- Greedy Algorithms
- Divide and Conquer
- Dynamic Programming
- Network Flow
- Computational intractability
- <u>Other topics</u>: Randomized algorithms, Computational Geometry, Number-theoretic algorithms etc.

Computational Intractability

Introduction

Computational Intractability

- Efficient Algorithms: All algorithms that run in time *polynomial* in the input size. We will call them polynomial time algorithms.

Computational Intractability

- Efficient Algorithms: All algorithms that run in time *polynomial* in the input size. We will call them polynomial time algorithms.
- Question: Given a problem, does there exist an efficient algorithm to solve the problem?

Computational Intractability

- Efficient Algorithms: All algorithms that run in time *polynomial* in the input size. We will call them polynomial time algorithms.
- Question: Given a problem, does there exist an efficient algorithm to solve the problem?
- There are a lot of problems arising in various fields for which this question is still unresolved.

Computational Intractability

- Efficient Algorithms: All algorithms that run in time *polynomial* in the input size. We will call them polynomial time algorithms.
- Question: Given a problem, does there exist an efficient algorithm to solve the problem?
- There are a lot of problems arising in various fields for which this question is still unresolved.
- Question: Are these problems *related* in some manner? Are there certain aspects that are common to all these problems?

Computational Intractability

- Efficient Algorithms: All algorithms that run in time *polynomial* in the input size. We will call them polynomial time algorithms.
- Question: Given a problem, does there exist an efficient algorithm to solve the problem?
- There are a lot of problems arising in various fields for which this question is still unresolved.
- Question: Are these problems *related* in some manner? Are there certain aspects that are common to all these problems?
- Question: 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.

Computational Intractability

- <u>NP-Complete Problems</u>: This is a large class of problems such that all problems in this class are equivalent in the following sense:
  - *A polynomial time algorithm for any one problem in this class implies the existence of polynomial time algorithm for all of them.*

Computational Intractability

- NP-Complete Problems: This is a large class of problems such that all problems in this class are equivalent in the following sense:
  - *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 correctly.
  - We say that Y is polynomial time reducible to X or $Y \leq_p X$.

Computational Intractability

- 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 or $Y \leq_p X$.

- Is $Bipartite - matching \leq_p Max - flow$?

Computational Intractability

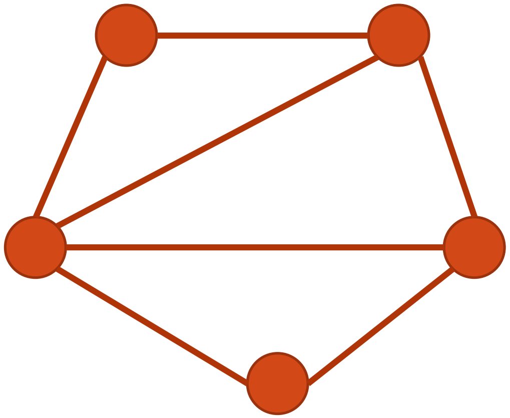
- Polynomial time reduction:
  - Consider two problems X and Y.
  - Suppose there is a *black box* that solves arbitrary instances of problem X.
  - Suppose any arbitrary instance of problem Y can be solved using a polynomial number of standard computational steps and a polynomial number of calls to the black box that solves instance of problem X correctly.
  - We say that Y is polynomial time reducible to X or $Y \leq_p X$.
- <u>Claim 1</u>: Suppose $Y \leq_p X$. If X can be solved in polynomial time, then Y can be solved in polynomial time.
- <u>Claim 2</u>: Suppose $Y \leq_p X$. If Y cannot be solved in polynomial time, then X cannot be solved in polynomial time.

Computational Intractability

Polynomial-time reductions

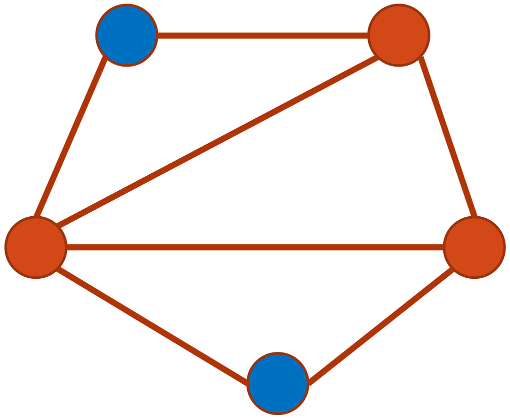
Computational Intractability: Reduction

- 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.
  - *Independent set*: A subset I of vertices is called an independent set if there is no edge between any pair of vertices in I.
- Problem(Maximum independent set): Given a graph $G = (V, E)$, output the size of the *independent set* of G of maximum size.



Computational Intractability: Reduction

- 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.
  - *Independent set*: A subset I of vertices is called an independent set if there is no edge between any pair of vertices in I.
- Problem(Maximum independent set): Given a graph $G = (V, E)$, output the size of the *independent set* of G of maximum size.



Computational Intractability: Reduction

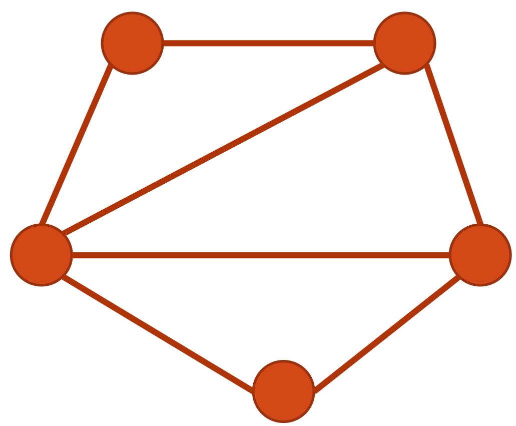
- <u>Problem(Independent set)</u>: Given a graph $G = (V, E)$ and an integer k, check if there is an *independent set* of size at least k in G.
  - *Independent set*: A subset I of vertices is called an independent set if there is no edge between any pair of vertices in I.

- <u>Problem(Maximum independent set)</u>: Given a graph $G = (V, E)$, output the size of the *independent set* of G of maximum size.

- <u>Claim 1</u>: Maximum-independent-set $\leq_p$ Independent-set
- <u>Claim 2</u>: Independent-set $\leq_p$ Maximum-independent-set

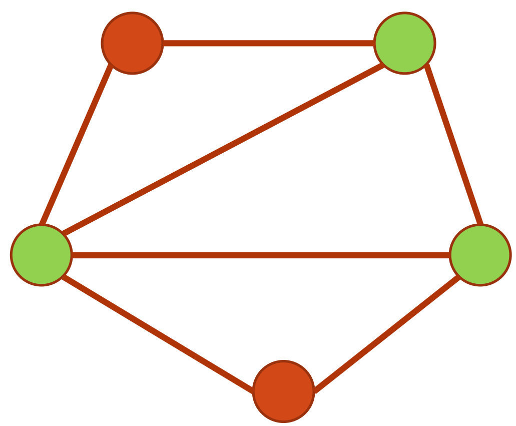
Computational Intractability: Reduction

- 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.
  - *Vertex cover*: A subset S of vertices is called a vertex cover of G if for any edge (u, v) in the graph u is in S or v is in S.
- Problem(Minimum vertex cover): Given a graph $G = (V, E)$, output the size of the *vertex cover* of G of minimum size.



Computational Intractability: Reduction

- 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.
  - *Vertex cover*: A subset S of vertices is called a vertex cover of G if for any edge (u, v) in the graph u is in S or v is in S.
- Problem(Minimum vertex cover): Given a graph $G = (V, E)$, output the size of the *vertex cover* of G of minimum size.



Computational Intractability: Reduction

- 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.
  - *Vertex cover*: a subset S of vertices is called a vertex cover of G if for any edge (u, v) in the graph u is in S or v is in S.

- Problem(Minimum vertex cover): Given a graph $G = (V, E)$, output the size of the *vertex cover* of G of minimum size.

- Claim 1: Minimum-vertex-cover $\leq_p$ Vertex-cover.
- Claim 2: Vertex-cover $\leq_p$ Minimum-vertex-cover

Computational Intractability: Reduction

- Claim: Independent-set $\leq_p$ Vertex-cover.

Computational Intractability: Reduction

- Claim: Independent-set $\leq_p$ Vertex-cover.
- Proof:
  - Claim 1: Let I be an independent set of G, then $V - I$ is a vertex cover of G.

Computational Intractability: Reduction

- Claim: Independent-set $\leq_p$ Vertex-cover.
- Proof:
  - Claim 1: Let I be an independent set of G, then $V - I$ is a vertex cover of G.
  - Claim 2: Let S be a vertex cover of G, then $V - S$ is an independent set of G.

Computational Intractability: Reduction

- Claim: Independent-set $\leq_p$ Vertex-cover.
- Proof:
  - Claim 1: Let I be an independent set of G, then $V - I$ is a vertex cover of G.
  - Claim 2: Let S be a vertex cover of G, then $V - S$ is an independent set of G.
  - Claim 3: G has an independent set of size at least k if and only if G has a vertex cover of size at most $(n - k)$.

Computational Intractability: Reduction

- Claim: Independent-set $\leq_p$ Vertex-cover.
- Proof:
  - Claim 1: Let I be an independent set of G, then $V - I$ is a vertex cover of G.
  - Claim 2: Let S be a vertex cover of G, then $V - S$ is an independent set of G.
  - Claim 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 of the independent set problem (G, k) create an instance of the vertex cover problem $(G, n - k)$, make a query to the black box solving the vertex cover problem and return the answer that is returned by the black box.

Computational Intractability: Reduction

- Claim: Minimum-vertex-cover $\leq_p$ Maximum-independent-set.

Computational Intractability: Reduction

- Claim: Minimum-vertex-cover $\leq_p$ Maximum-independent-set.
- Proof 1: G has an independent set of size at least k if and only if G has a vertex cover of size at most $(n - k)$.

Computational Intractability: Reduction

- Claim: Minimum-vertex-cover $\leq_p$ Maximum-independent-set.
- Proof 1: G has an independent set of size at least k if and only if G has a vertex cover of size at most $(n - k)$.
- Proof 2:
  - Minimum-vertex-cover $\leq_p$ Vertex-cover.
  - Vertex-cover $\leq_p$ Independent-set.
  - Independent-set $\leq_p$ Maximum-independent-set.

Computational Intractability: Reduction

- <u>Claim</u>: Minimum-vertex-cover $\leq_p$ Maximum-independent-set.
- <u>Proof 1</u>: G has an independent set of size at least k if and only if G has a vertex cover of size at most $(n - k)$.
- <u>Proof 2</u>:
  - Minimum-vertex-cover $\leq_p$ Vertex-cover.
  - Vertex-cover $\leq_p$ Independent-set.
  - Independent-set $\leq_p$ Maximum-independent-set.

- <u>Theorem</u>: If $X \leq_p Y$ and $Y \leq_p Z$, then $X \leq_p Z$.

Computational Intractability: Reduction

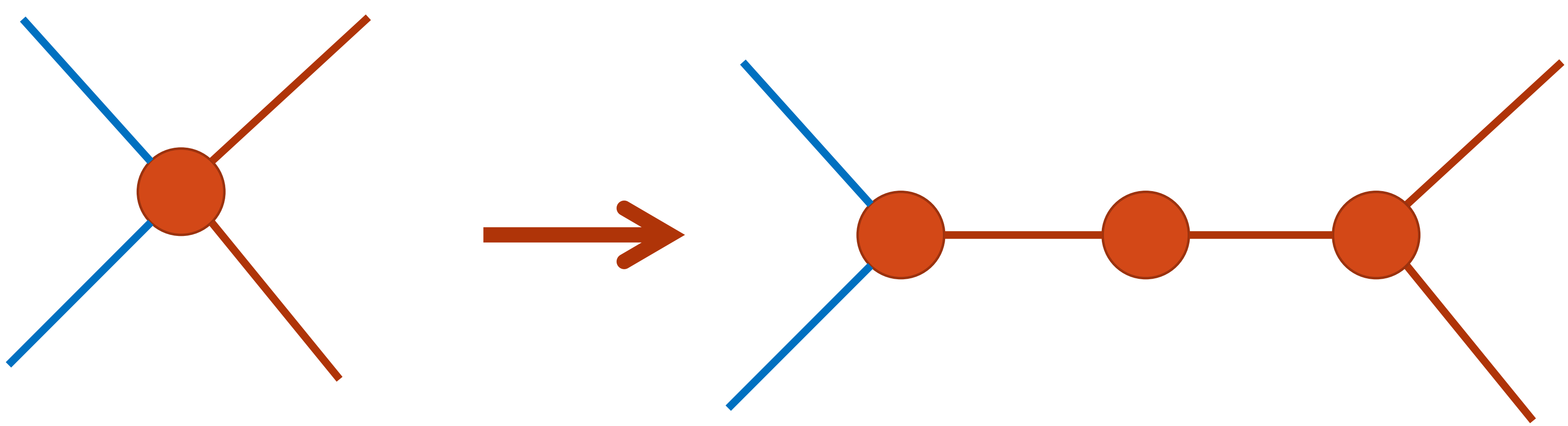
- Problem(Deg-3-Independent set): Given a graph $G = (V, E)$ of *bounded degree* 3 and an integer k, check if there is an *independent set* of size at least k in G.
  - *Graph with bounded degree 3*: A graph is said to have bounded degree 3 if the degrees of all vertices in the graph is at most 3.

- Claim: Independent-set $\leq_p$ Deg-3-Independent-set.

Computational Intractability: Reduction

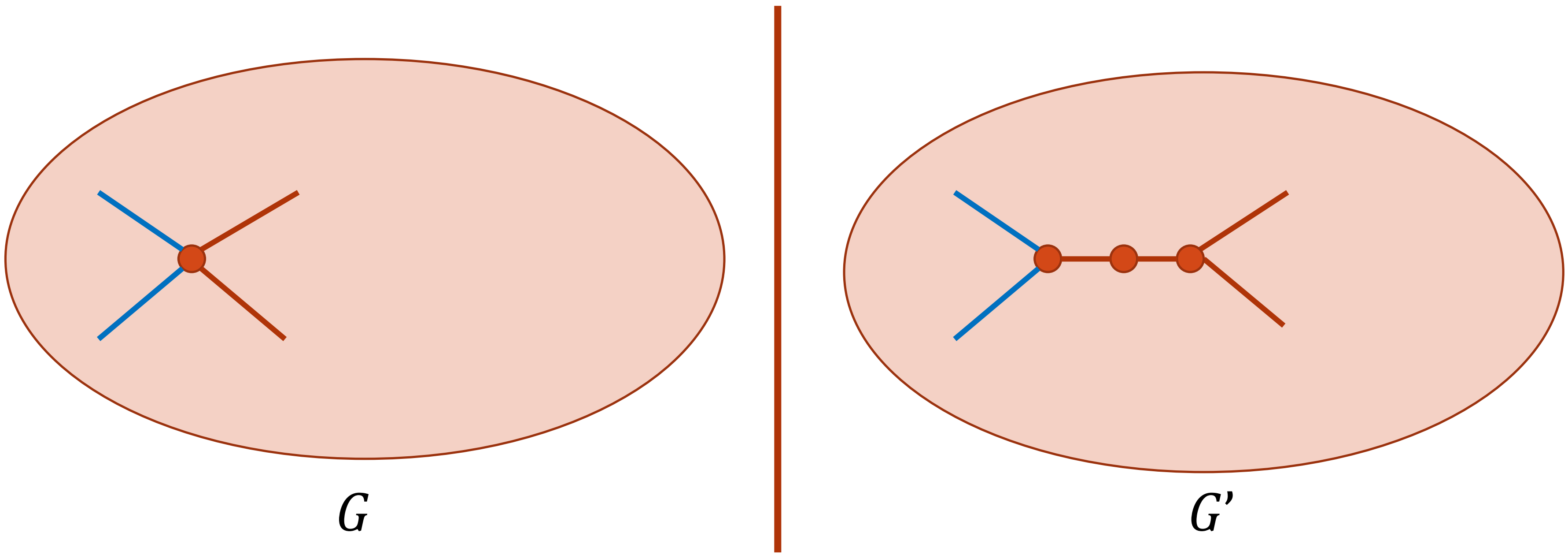
- Problem(Deg-3-Independent set): Given a graph $G = (V, E)$ of *bounded degree* 3 and an integer k, check if there is an *independent set* of size at least k in G.
  - *Graph with bounded degree 3*: A graph is said to have bounded degree 3 if the degrees of all vertices in the graph is at most 3.

- Claim: Independent-set $\leq_p$ Deg-3-Independent-set.



Computational Intractability: Reduction

- Claim: Independent-set $\leq_p$ Deg-3-Independent-set.



- Claim: G has an independent set of size at least k if and only if G' has an independent set of size at least $(k+1)$.

End