Teoria dos Grafos e Computabilidade

— Tractability and Intractability —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Laboratory of Image and Multimedia Data Science – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Feb 2025

Teoria dos Grafos e Computabilidade

— Intractability —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Laboratory of Image and Multimedia Data Science – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Feb 2025

Algorithm Design

- Patterns
  - Greed. $O(n \log n)$ interval scheduling.
  - Divide-and-conquer. $O(n \log n)$ closest pair of points.
  - Dynamic programming. $O(n^2)$ edit distance.
  - Duality. $O(n^3)$ maximum flow and minimum cuts.

Algorithm Design

- Patterns
  - Greed. $O(n \log n)$ interval scheduling.
  - Divide-and-conquer. $O(n \log n)$ closest pair of points.
  - Dynamic programming. $O(n^2)$ edit distance.
  - Duality. $O(n^3)$ maximum flow and minimum cuts.
  - Reductions.
  - Local search.
  - Randomization.

Algorithm Design

- Patterns
  - Greed. $O(n \log n)$ interval scheduling.
  - Divide-and-conquer. $O(n \log n)$ closest pair of points.
  - Dynamic programming. $O(n^2)$ edit distance.
  - Duality. $O(n^3)$ maximum flow and minimum cuts.
  - Reductions.
  - Local search.
  - Randomization.
- Anti-patterns
  - NP-completeness. $O(n^k)$ algorithm unlikely.
  - PSPACE-completeness. $O(n^k)$ certification algorithm unlikely.
  - Undecidability. No algorithm possible.

Computational Tractability

- When is an algorithm an efficient solution to a problem?

Computational Tractability

- When is an algorithm an efficient solution to a problem?

  When its running time is polynomial in the size of the input.

Computational Tractability

- When is an algorithm an efficient solution to a problem?

  When its running time is polynomial in the size of the input.

- A problem is computationally tractable

  if it has a polynomial-time algorithm.

Computational Tractability

- When is an algorithm an efficient solution to a problem?
  When its running time is polynomial in the size of the input.
- A problem is computationally tractable
  if it has a polynomial-time algorithm.

Polynomial time	Probably not
Shortest path	Longest path
Matching	3-D matching
Minimum cut	Maximum cut
2-SAT	3-SAT
Planar four-colour	Planar three-colour
Bipartite vertex cover	Vertex cover
Primality testing	Factoring

Problem Classification

- Classify problems based on whether they admit efficient solutions or not.
- Some extremely hard problems cannot be solved efficiently (e.g., chess on an *n*-by-*n* board).

Problem Classification

- Classify problems based on whether they admit efficient solutions or not.
- Some extremely hard problems cannot be solved efficiently (e.g., chess on an n-by-n board).
- However, classification is unclear for a very large number of discrete computational problems.

Problem Classification

- Classify problems based on whether they admit efficient solutions or not.
- Some extremely hard problems cannot be solved efficiently (e.g., chess on an n-by-n board).
- However, classification is unclear for a very large number of discrete computational problems.
- We can prove that these problems are fundamentally equivalent and are manifestations of the same problem!

Teoria dos Grafos e Computabilidade

— Reductions —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Laboratory of Image and Multimedia Data Science – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Feb 2025

Polynomial-Time Reduction

- The goal is to express statements of the type

  Problem X is at least as hard as problem Y.

- Use the notion of reductions.

Y is polynomial-time reducible to X ($Y \leq_P X$)

Polynomial-Time Reduction

- The goal is to express statements of the type

  Problem X is at least as hard as problem Y.

- Use the notion of reductions.

Y is polynomial-time reducible to X ($Y \leq_P X$)

if an arbitrary instance of Y can be solved using a polynomial number of standard operations, plus a polynomial number of calls to a black box that solves problem X.

- $Y \leq_P X$ implies that X is at least as hard as Y.
- Such reductions are Cook reductions. Karp reductions allow only one call to the black box that solves X.

Usefulness of Reductions

Claim: *If $Y \leq_P X$ and X can be solved in polynomial time, then Y can be solved in polynomial time.*

Usefulness of Reductions

Claim: *If $Y \leq_P X$ and X can be solved in polynomial time, then Y can be solved in polynomial time.*

Contrapositive: *If $Y \leq_P X$ and Y cannot be solved in polynomial time, then X cannot be solved in polynomial time.*

Usefulness of Reductions

Claim: *If $Y \leq_P X$ and X can be solved in polynomial time, then Y can be solved in polynomial time.*

Contrapositive: *If $Y \leq_P X$ and Y cannot be solved in polynomial time, then X cannot be solved in polynomial time.*

Informally: *If Y is hard, and we can show that Y reduces to X, then the hardness spreads to X.*

Usefulness of Reductions

Claim: *If $Y \leq_P X$ and X can be solved in polynomial time, then Y can be solved in polynomial time.*

Contrapositive: *If $Y \leq_P X$ and Y cannot be solved in polynomial time, then X cannot be solved in polynomial time.*

Informally: *If Y is hard, and we can show that Y reduces to X, then the hardness spreads to X.*

Purpose. Classify problems according to relative difficulty.

Usefulness of Reductions

Claim: *If $Y \leq_P X$ and X can be solved in polynomial time, then Y can be solved in polynomial time.*

Contrapositive: *If $Y \leq_P X$ and Y cannot be solved in polynomial time, then X cannot be solved in polynomial time.*

Informally: *If Y is hard, and we can show that Y reduces to X, then the hardness spreads to X.*

Purpose. Classify problems according to relative difficulty.

- If $Y \leq_P X$ and X can be solved in polynomial-time, then Y can also be solved in polynomial time. Design algorithms

Usefulness of Reductions

Claim: *If $Y \leq_P X$ and X can be solved in polynomial time, then Y can be solved in polynomial time.*

Contrapositive: *If $Y \leq_P X$ and Y cannot be solved in polynomial time, then X cannot be solved in polynomial time.*

Informally: *If Y is hard, and we can show that Y reduces to X, then the hardness spreads to X.*

Purpose. Classify problems according to relative difficulty.

- If $Y \leq_P X$ and X can be solved in polynomial-time, then Y can also be solved in polynomial time. **Design algorithms**
- If $Y \leq_P X$ and Y cannot be solved in polynomial-time, then X cannot be solved in polynomial time. **Establish intractability**

Usefulness of Reductions

Claim: *If $Y \leq_P X$ and X can be solved in* polynomial time*, then Y can be solved in* polynomial time.

Contrapositive: *If $Y \leq_P X$ and Y* cannot *be solved in polynomial time, then X* cannot *be solved in polynomial time.*

Informally: *If Y is hard, and we can show that Y reduces to X, then the hardness spreads to X.*

Purpose. Classify problems according to relative difficulty.

- If $Y \leq_P X$ and X can be solved in polynomial-time, then Y can also be solved in polynomial time. Design algorithms
- If $Y \leq_P X$ and Y cannot be solved in polynomial-time, then X cannot be solved in polynomial time. Establish intractability
- If $X \leq_P Y$ and $Y \leq_P X$, we use notation $X \equiv_P Y$ in order to express the equivalance. Establish equivalence

Polynomial Transformation

Problem X polynomial reduces (Cook) to problem Y if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to oracle that solves problem Y.

Polynomial Transformation

Problem X polynomial reduces (Cook) to problem Y if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to oracle that solves problem Y.

Problem X polynomial transforms (Karp) to problem Y if given any input x to X, we can construct an input y such that x is a yes instance of X iff y is a yes instance of Y.

Polynomial Transformation

Problem X polynomial reduces (Cook) to problem Y if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to oracle that solves problem Y.

Problem X polynomial transforms (Karp) to problem Y if given any input x to X, we can construct an input y such that x is a yes instance of X iff y is a yes instance of Y.

Polynomial transformation is polynomial reduction with just one call to oracle for Y, exactly at the end of the algorithm for X. Almost all previous reductions were of this form.

Reductions

Reduction Design a fast algorithm for one computational problem, using a supposedly fast algorithm for another problem as a subroutine.

Reductions

Reduction Design a fast algorithm for one computational problem, using a supposedly fast algorithm for another problem as a subroutine.

- Use to compare the two problems.
- Even if we don't know whether they can be solved in polynomial time or not,
- We can learn that either they both can or neither can.
- We can also learn that they have a **similar structure**.

Reductions

Reduction Design a fast algorithm for one computational problem, using a supposedly fast algorithm for another problem as a subroutine.

- Use to compare the two problems.
- Even if we don't know whether they can be solved in polynomial time or not,
- We can learn that either they both can or neither can.
- We can also learn that they have a **similar structure**.

Design a fast algorithm for P_{alg} using a supposed fast algorithm for P_{oracle} as a subroutine.

Cook vs Karp Reductions

$$\boxed{P_{alg} \leq_P P_{oracle}}$$

Cook Reduction *Design any fast algorithm for P_{alg} using a supposed fast algorithm for P_{oracle} as a subroutine*

Cook vs Karp Reductions

$$P_{alg} \leq_P P_{oracle}$$

Cook Reduction *Design any fast algorithm for P_{alg} using a supposed fast algorithm for P_{oracle} as a subroutine*

Karp Reduction *The algorithm for P_{alg} calls that for P_{oracle} only once:* *Yes$\Rightarrow$Yes & No$\Rightarrow$No*

Cook vs Karp Reductions

$$\boxed{P_{alg} \leq_P P_{oracle}}$$

Cook Reduction *Design any fast algorithm for P_{alg} using a supposed fast algorithm for P_{oracle} as a subroutine*

Karp Reduction *The algorithm for P_{alg} calls that for P_{oracle} only once: Yes$\Rightarrow$Yes & No$\Rightarrow$No*

We give a fast algorithm for P_{alg} using a supposed fast algorithm for P_{oracle} as a subroutine.

Cook vs Karp Reductions

$$P_{alg} \leq_P P_{oracle}$$

Cook Reduction *Design any fast algorithm for P_{alg} using a supposed fast algorithm for P_{oracle} as a subroutine*

Karp Reduction *The algorithm for P_{alg} calls that for P_{oracle} only once: Yes$\Rightarrow$Yes & No$\Rightarrow$No*

We give a fast algorithm for P_{alg} using a supposed fast algorithm for P_{oracle} as a subroutine.

Is there a fast algorithm for P_{alg}?

Cook vs Karp Reductions

$$\boxed{P_{alg} \leq_P P_{oracle}}$$

Cook Reduction *Design any fast algorithm for P_{alg} using a supposed fast algorithm for P_{oracle} as a subroutine*

Karp Reduction *The algorithm for P_{alg} calls that for P_{oracle} only once: Yes$\Rightarrow$Yes & No$\Rightarrow$No*

We give a fast algorithm for P_{alg} using a supposed fast algorithm for P_{oracle} as a subroutine.

Is there a fast algorithm for P_{alg}?
Is there a fast algorithm for P_{oracle}?

Reductions

We give a fast algorithm for P_{alg} using a supposed fast algorithm for P_{oracle} as a subroutine.

If there is a fast algorithm for P_{alg}?

If there is not a fast algorithm for P_{alg}?

If there is a fast algorithm for P_{oracle}?

If there is not a fast algorithm for P_{oracle}?

Reductions

We give a fast algorithm for P_{alg} using a supposed fast algorithm for P_{oracle} as a subroutine.

If there is a fast algorithm for P_{alg}?

If there is not a fast algorithm for P_{alg}?

If there is a fast algorithm for P_{oracle}?

then there is a fast algorithm for P_{alg}

If there is not a fast algorithm for P_{oracle}?

Reductions

We give a fast algorithm for P_{alg} using a supposed fast algorithm for P_{oracle} as a subroutine.

If there is a fast algorithm for P_{alg}?

If there is not a fast algorithm for P_{alg}?

then there is not a fast algorithm for P_{oracle}

If there is a fast algorithm for P_{oracle}?

then there is a fast algorithm for P_{alg}

If there is not a fast algorithm for P_{oracle}?

Reductions

We give a fast algorithm for P_{alg} using a supposed fast algorithm for P_{oracle} as a subroutine.

If there is a fast algorithm for P_{alg}?

???

If there is not a fast algorithm for P_{alg}?

then there is not a fast algorithm for P_{oracle}

If there is a fast algorithm for P_{oracle}?

then there is a fast algorithm for P_{alg}

If there is not a fast algorithm for P_{oracle}?

Reductions

We give a fast algorithm for P_{alg} using a supposed fast algorithm for P_{oracle} as a subroutine.

If there is a fast algorithm for P_{alg}?

???

If there is not a fast algorithm for P_{alg}?

then there is not a fast algorithm for P_{oracle}

If there is a fast algorithm for P_{oracle}?

then there is a fast algorithm for P_{alg}

If there is not a fast algorithm for P_{oracle}?

???

Reductions

We give a fast algorithm for P_{alg} using a supposed fast algorithm for P_{oracle} as a subroutine.

Reductions

We give a fast algorithm for P_{alg} using a supposed fast algorithm for P_{oracle} as a subroutine.

P_{alg} is **at least as easy as** P_{oracle}

(Modulo polynomial terms.)

Reductions

We give a fast algorithm for P_{alg} using a supposed fast algorithm for P_{oracle} as a subroutine.

P_{alg} is at least as easy as P_{oracle}

(Modulo polynomial terms.)

P_{oracle} is at least as hard as P_{alg}

(Modulo polynomial terms.)

Reductions

We give a fast algorithm for P_{alg} using a supposed fast algorithm for P_{oracle} as a subroutine.

P_{alg} is at least as easy as P_{oracle}

(Modulo polynomial terms.)

P_{oracle} is at least as hard as P_{alg}

(Modulo polynomial terms.)

The problems have a similar underling structure and it is used to design new Algorithms

Reduction Strategies

- Simple equivalence.
- Special case to general case.
- Encoding with gadgets.

Optimization versus Decision Problems

- So far, we have developed algorithms that solve optimization problems.
  - Compute the largest flow.
  - Find the closest pair of points.
  - Find the schedule with the least completion time.

Optimization versus Decision Problems

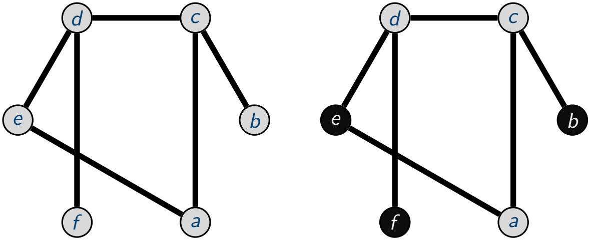
- So far, we have developed algorithms that solve optimization problems.
  - Compute the largest flow.
  - Find the closest pair of points.
  - Find the schedule with the least completion time.
- Now, we will focus on decision versions of problems, e.g..,

  Is there a flow with value at least k, for a given value of k?

Independent sets

Let $G = (V, E)$ be an undirected connected graph.

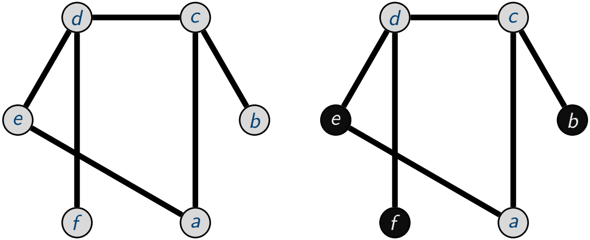
- A subset $S \subseteq V$ is an **independent set** if $\forall u, v \in S$ there exist an edge $(u, v) \in E$.
- Given G and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \geq k$, and for each edge at most one of its endpoints is in S?



Independent sets

Let $G = (V, E)$ be an undirected connected graph.

- A subset $S \subseteq V$ is an **independent set** if $\forall u, v \in S$ there exist an edge $(u, v) \in E$.
- Given G and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \geq k$, and for each edge at most one of its endpoints is in S?

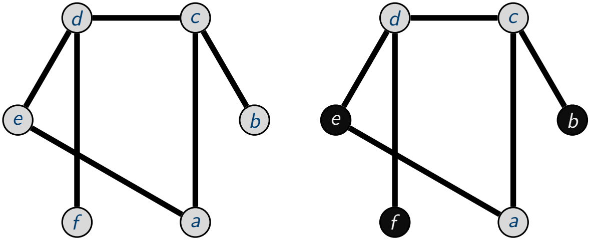


Is there an independent set of size ≥ 3? Yes.

Independent sets

Let $G = (V, E)$ be an undirected connected graph.

- A subset $S \subseteq V$ is an **independent set** if $\forall u, v \in S$ there exist an edge $(u, v) \in E$.
- Given G and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \geq k$, and for each edge at most one of its endpoints is in S?

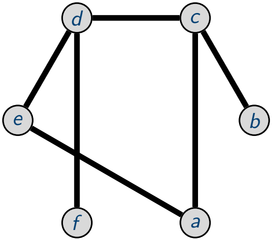


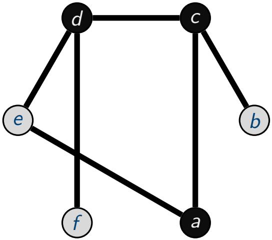
Is there an independent set of size ≥ 4? No.

Vertex cover

Let $G = (V, E)$ be an undirected connected graph.

- A subset $S \subseteq V$ is an vertex cover if $\forall (u, v) \in E$, either $u \in S$ or $v \in S$.
- Given a graph G and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \leq k$, and for each edge, at least one of its endpoints is in S?

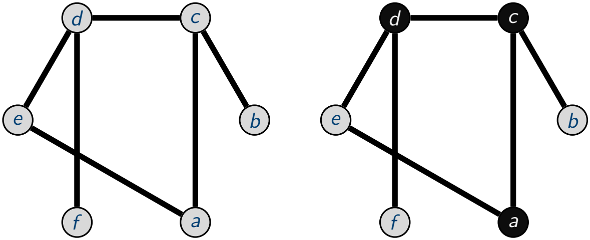




Vertex cover

Let $G = (V, E)$ be an undirected connected graph.

- A subset $S \subseteq V$ is an vertex cover if $\forall(u, v) \in E$, either $u \in S$ or $v \in S$.
- Given a graph G and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \leq k$, and for each edge, at least one of its endpoints is in S?

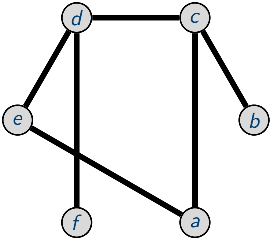


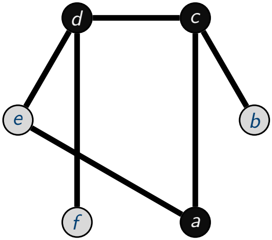
Is there a vertex cover of size ≤ 3? Yes.

Vertex cover

Let $G = (V, E)$ be an undirected connected graph.

- A subset $S \subseteq V$ is an vertex cover if $\forall (u, v) \in E$, either $u \in S$ or $v \in S$.
- Given a graph G and an integer k, is there a subset of vertices $S \subseteq V$ such that $|S| \leq k$, and for each edge, at least one of its endpoints is in S?



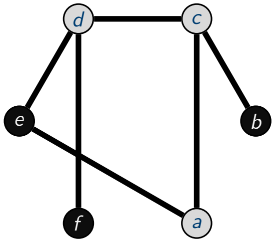


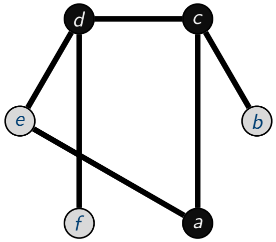
Is there a vertex cover of size ≤ 2? No.

Vertex cover

Let $G = (V, E)$ be an undirected connected graph, and S a vertex cover of G

As S is a vertex cover of G, then V-S is an independent set.





Independent Set and Vertex Cover

- Given an undirected graph $G(V, E)$, a subset $S \subseteq V$ is an independent set if no two vertices in S are connected by an edge.
- Given an undirected graph $G(V, E)$, a subset $S \subseteq V$ is a vertex cover if every edge in E is incident on at least one vertex in S.

Independent Set and Vertex Cover

- Given an undirected graph $G(V, E)$, a subset $S \subseteq V$ is an independent set if no two vertices in S are connected by an edge.
- Given an undirected graph $G(V, E)$, a subset $S \subseteq V$ is a vertex cover if every edge in E is incident on at least one vertex in S.

INDEPENDENT SET

INSTANCE	Undirected graph G and an integer k
QUESTION	Does G contain an independent set of size

VERTEX COVER

INSTANCE	Undirected graph G and an integer k
QUESTION	Does G contain a vertex cover of size

Independent Set and Vertex Cover

- Given an undirected graph $G(V, E)$, a subset $S \subseteq V$ is an independent set if no two vertices in S are connected by an edge.
- Given an undirected graph $G(V, E)$, a subset $S \subseteq V$ is a vertex cover if every edge in E is incident on at least one vertex in S.

Independent Set

INSTANCE Undirected graph G and an integer k

QUESTION Does G contain an independent set of size at least k?

Vertex cover

INSTANCE Undirected graph G and an integer k

QUESTION Does G contain a vertex cover of size at most k?

Independent Set and Vertex Cover

- Given an undirected graph $G(V, E)$, a subset $S \subseteq V$ is an independent set if no two vertices in S are connected by an edge.
- Given an undirected graph $G(V, E)$, a subset $S \subseteq V$ is a vertex cover if every edge in E is incident on at least one vertex in S.

INDEPENDENT SET

INSTANCE Undirected graph G and an integer k

QUESTION Does G contain an independent set of size at least k?

VERTEX COVER

INSTANCE Undirected graph G and an integer k

QUESTION Does G contain a vertex cover of size at most k?

- Demonstrate simple equivalence between these two problems.

Independent Set and Vertex Cover

- Given an undirected graph $G(V, E)$, a subset $S \subseteq V$ is an independent set if no two vertices in S are connected by an edge.
- Given an undirected graph $G(V, E)$, a subset $S \subseteq V$ is a vertex cover if every edge in E is incident on at least one vertex in S.

INDEPENDENT SET

INSTANCE	Undirected graph G and an integer k
QUESTION	Does G contain an independent set of size at least k?

VERTEX COVER

INSTANCE	Undirected graph G and an integer k
QUESTION	Does G contain a vertex cover of size at most k?

- Demonstrate simple equivalence between these two problems.
- S is an independent set in G iff $V - S$ is a vertex cover in G.

Independent Set and Vertex Cover

- Given an undirected graph $G(V, E)$, a subset $S \subseteq V$ is an independent set if no two vertices in S are connected by an edge.
- Given an undirected graph $G(V, E)$, a subset $S \subseteq V$ is a vertex cover if every edge in E is incident on at least one vertex in S.

Independent Set

INSTANCE Undirected graph G and an integer k

QUESTION Does G contain an independent set of size at least k?

Vertex cover

INSTANCE Undirected graph G and an integer k

QUESTION Does G contain a vertex cover of size at most k?

- Demonstrate simple equivalence between these two problems.
- S is an independent set in G iff $V - S$ is a vertex cover in G.
- Independent Set $\leq_P$ Vertex Cover and Vertex Cover $\leq_P$ Independent Set.

Independent Set and Vertex Cover

INDEPENDENT SET $\equiv_P$ VERTEX COVER

We show S is an independent set iff V - S is a vertex cover

Independent Set and Vertex Cover

INDEPENDENT SET $\equiv_P$ VERTEX COVER

We show S is an independent set iff V - S is a vertex cover

- Let S be any independent set.
- Consider an arbitrary edge (u, v).
- S independent $\Rightarrow u \notin S$ or $v \notin S \Rightarrow u \in V - S$ or $v \in V - S$.
- Thus, V - S covers (u, v).

Independent Set and Vertex Cover

INDEPENDENT SET $\equiv_P$ VERTEX COVER

We show S is an independent set iff V - S is a vertex cover

- Let S be any independent set.
- Consider an arbitrary edge (u, v).
- S independent $\Rightarrow u \notin S$ or $v \notin S \Rightarrow u \in V - S$ or $v \in V - S$.
- Thus, V - S covers (u, v).

- Let V - S be any vertex cover.
- Consider two nodes $u \in S$ and $v \in S$.
- Observe that $(u, v) \notin E$ since V - S is a vertex cover.
- Thus, no two nodes in S are joined by an edge $\Rightarrow$ S independent set

Set Cover

Given a set U of elements, a collection $S = \{S_1, S_2, \cdots, S_m\}$ of subsets of U.

- A subset $C \subseteq S$ is a **set cover** if the union of elements of C is equal to U.
- Given U, S, and an integer k, does there exist a collection of $\leq$ k of these sets whose union is equal to U?

Set Cover

Given a set U of elements, a collection $S = \{S_1, S_2, \cdots, S_m\}$ of subsets of U.

- A subset $C \subseteq S$ is a set cover if the union of elements of C is equal to U.
- Given U, S, and an integer k, does there exist a collection of $\leq$ k of these sets whose union is equal to U?

Sample application:

- m available pieces of software
- Set U of n capabilities that we would like our system to have
- The i^{th} piece of software provides the set $S_i \subseteq U$ of capabilities.
- The goal is to achieve all n capabilities using fewest pieces of software.

Set Cover

Given a set U of elements, a collection $S = \{S_1, S_2, \cdots, S_m\}$ of subsets of U.

- A subset $C \subseteq S$ is a **set cover** if the union of elements of C is equal to U.
- Given U, S, and an integer k, does there exist a collection of $\leq$ k of these sets whose union is equal to U?

Sample application:

- $U = \{1, 2, 3, 4, 5, 6, 7\}$ and $k = 2$

$$\begin{array}{ll} S_1 = \{3, 7\} & S_4 = \{2, 4\} \\ S_2 = \{3, 4, 5, 6\} & S_5 = \{5\} \\ S_3 = \{1\} & S_6 = \{1, 2, 6, 7\} \end{array}$$

Set Cover

Given a set U of elements, a collection $S = \{S_1, S_2, \cdots, S_m\}$ of subsets of U.

- A subset $C \subseteq S$ is a **set cover** if the union of elements of C is equal to U.
- Given U, S, and an integer k, does there exist a collection of $\leq$ k of these sets whose union is equal to U?

Sample application:

- $U = \{1, 2, 3, 4, 5, 6, 7\}$ and $k = 2$

$$S_1 = \{3, 7\} \qquad S_4 = \{2, 4\}$$
$$\boxed{S_2 = \{3, 4, 5, 6\}} \qquad S_5 = \{5\}$$
$$S_3 = \{1\} \qquad \boxed{S_6 = \{1, 2, 6, 7\}}$$

Vertex Cover and Set Cover

- Set cover is a packing problem: pack as many vertices as possible, subject to constraints (the edges).
- Vertex Cover is a covering problem: cover all edges in the graph with as few vertices as possible.
- There are more general covering problems.

Vertex Cover and Set Cover

- Set cover is a packing problem: pack as many vertices as possible, subject to constraints (the edges).
- Vertex Cover is a covering problem: cover all edges in the graph with as few vertices as possible.
- There are more general covering problems.

SET COVER

INSTANCE A set U of n elements, a collection $S_1, S_2, \ldots, S_m$ of subsets of U, and an integer k.

QUESTION Is there a collection of $\leq k$ sets in the collection whose union is U?

VERTEX COVER

INSTANCE Undirected graph G and an integer k

QUESTION Does G contain a vertex cover of size

Vertex Cover and Set Cover

- Set cover is a packing problem: pack as many vertices as possible, subject to constraints (the edges).
- Vertex Cover is a covering problem: cover all edges in the graph with as few vertices as possible.
- There are more general covering problems.

SET COVER

INSTANCE A set U of n elements, a collection $S_1, S_2, \ldots, S_m$ of subsets of U, and an integer k.

QUESTION Is there a collection of $\leq k$ sets in the collection whose union is U?

VERTEX COVER

INSTANCE Undirected graph G and an integer k

QUESTION Does G contain a vertex cover of size at most k?

Reducing Vertex Cover to Set Cover

VERTEX COVER $\leq_P$ SET COVER

Reducing Vertex Cover to Set Cover

VERTEX COVER $\leq_P$ SET COVER

- Input to Vertex Cover is an undirected graph $G = (V, E)$ with n vertices.
- Create an instance of Set Cover in which
  - $k = k$, $U = E$, $S_v = \{e \in E : e \textit{ incident to } v\}$

Reducing Vertex Cover to Set Cover

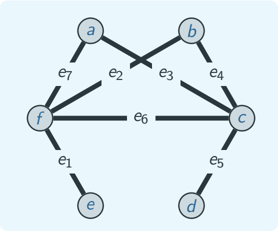
Vertex Cover $\leq_P$ Set Cover

- Input to Vertex Cover is an undirected graph $G = (V, E)$ with n vertices.
- Create an instance of Set Cover in which
  - $k = k$, $U = E$, $S_v = \{e \in E : e \text{ incident to } v\}$
- U can be covered with fewer than k subsets iff G has a vertex cover with at most k nodes.

Reducing Vertex Cover to Set Cover

VERTEX COVER $\leq_P$ SET COVER

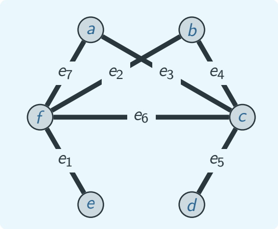
- Input to Vertex Cover is an undirected graph $G = (V, E)$ with n vertices.
- Create an instance of Set Cover in which
  - $k = k$, $U = E$, $S_v = \{e \in E : e \text{ incident to } v\}$
- U can be covered with fewer than k subsets iff G has a vertex cover with at most k nodes.



Reducing Vertex Cover to Set Cover

VERTEX COVER $\leq_P$ SET COVER

- Input to Vertex Cover is an undirected graph $G = (V, E)$ with n vertices.
- Create an instance of Set Cover in which
  - $k = k$, $U = E$, $S_v = \{e \in E : e \text{ incident to } v\}$
- U can be covered with fewer than k subsets iff G has a vertex cover with at most k nodes.



$U = \{1, 2, 3, 4, 5, 6, 7\}$ and $k = 2$

$S_1 = \{3, 7\}$ $\quad$ $S_4 = \{2, 4\}$

$S_2 = \{3, 4, 5, 6\}$ $\quad$ $S_5 = \{5\}$

$S_3 = \{1\}$ $\quad$ $S_6 = \{1, 2, 6, 7\}$

Boolean Satisfiability

- Abstract problems formulated in Boolean notation.
- Often used to specify problems, e.g., in AI.

Boolean Satisfiability

- Abstract problems formulated in Boolean notation.
- Often used to specify problems, e.g., in AI.

- We are given a set $X = \{x_1, x_2, \ldots, x_n\}$ of n Boolean variables.
- Each variable can take the value 0 or 1.
- A **term** is a variable x_i or its negation $\overline{x_i}$.
- A **clause** of **length** l is a disjunction of l distinct terms $t_1 \vee t_2 \vee \cdots t_l$.
- A **truth assignment** for X is a function $\nu : X \rightarrow \{0, 1\}$.
- An assignment **satisfies** a clause C if it causes C to evaluate to 1 under the rules of Boolean logic.
- An assignment **satisfies** a collection of clauses $C_1, C_2, \ldots C_k$ if it causes $C_1 \wedge C_2 \wedge \cdots C_k$ to evaluate to 1.
  - ν is a **satisfying assignment** with respect to $C_1, C_2, \ldots C_k$.
  - set of clauses $C_1, C_2, \ldots C_k$ is **satisfiable**.

Satisfiability Problem (SAT)

INSTANCE A set of clauses $C_1, C_2, \ldots C_k$ over a set $X = \{x_1, x_2, \ldots x_n\}$ of n variables.

QUESTION Is there a satisfying truth assignment for X with respect to C?

SAT and 3-SAT

Satisfiability Problem (SAT)

INSTANCE A set of clauses $C_1, C_2, \ldots C_k$ over a set $X = \{x_1, x_2, \ldots x_n\}$ of n variables.

QUESTION Is there a satisfying truth assignment for X with respect to C?

3-Satisfiability Problem (3-SAT)

INSTANCE A set of clauses $C_1, C_2, \ldots C_k$ each of length 3 over a set $X = \{x_1, x_2, \ldots x_n\}$ of n variables.

QUESTION Is there a satisfying truth assignment for X with respect to C?

SAT and 3-SAT

SATISFIABILITY PROBLEM (SAT)

INSTANCE A set of clauses $C_1, C_2, \ldots C_k$ over a set $X = \{x_1, x_2, \ldots x_n\}$ of n variables.

QUESTION Is there a satisfying truth assignment for X with respect to C?

3-SATISFIABILITY PROBLEM (3-SAT)

INSTANCE A set of clauses $C_1, C_2, \ldots C_k$ each of length 3 over a set $X = \{x_1, x_2, \ldots x_n\}$ of n variables.

QUESTION Is there a satisfying truth assignment for X with respect to C?

- SAT and 3-SAT are fundamental combinatorial search problems.
- We have to make n independent decisions (the assignments for each variable) while satisfying a set of constraints.
- Satisfying each constraint in isolation is easy, but we have to make our decisions so that all constraints are satisfied simultaneously.

3-SAT and Independent Set

- We want to prove $\boxed{\text{3-SAT} \leq_P \textsc{Independent Set}}$.

3-SAT and Independent Set

- We want to prove 3-SAT $\leq_P$ INDEPENDENT SET.
- Two ways to think about 3-SAT:
  1. Make an independent 0/1 decision on each variable and succeed if we achieve one of three ways in which to satisfy each clause.
  2. Choose (at least) one term from each clause. Find a truth assignment that causes each chosen term to evaluate to 1. Ensure that no two terms selected **conflict**, i.e., select x_i and $\overline{x_i}$.

Proving 3-SAT $\leq_P$ Independent Set

3-SAT $\leq_P$ Independent set

Given an instance Φ of 3-SAT, we construct an instance (G, k) of independent set that has an independent set of size k iff Φ is satisfiable.

Proving 3-SAT $\leq_P$ Independent Set

3-SAT $\leq_P$ Independent set

Given an instance Φ of 3-SAT, we construct an instance (G, k) of independent set that has an independent set of size k iff Φ is satisfiable.

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

Proving 3-SAT $\leq_P$ Independent Set

3-SAT $\leq_P$ Independent set

Given an instance Φ of 3-SAT, we construct an instance (G, k) of independent set that has an independent set of size k iff Φ is satisfiable.

Construction.

- G contains 3 nodes for each clause (k=3), one for each literal.
- Connect 3 literals in a clause in a triangle.
- Connect literal to each of its negations.

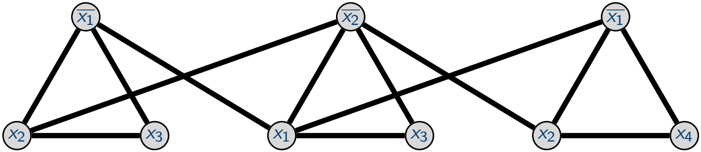
$$\Phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_2} \vee x_1 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$$

3-SAT $\leq_P$ INDEPENDENT SET

Given an instance Φ of 3-SAT, we construct an instance (G, k) of independent set that has an independent set of size k iff Φ is satisfiable.

Construction.

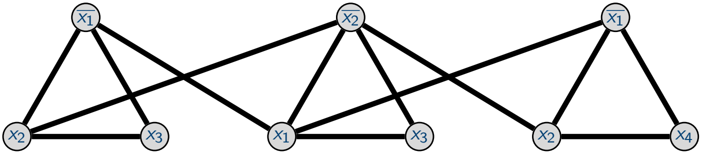
- G contains 3 nodes for each clause (k=3), one for each literal.
- Connect 3 literals in a clause in a triangle.
- Connect literal to each of its negations.



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

3-SAT $\leq_P$ INDEPENDENT SET

G contains independent set of size $k = |\Phi|$ iff Φ is satisfiable.



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

Proving 3-SAT $\leq_P$ Independent Set

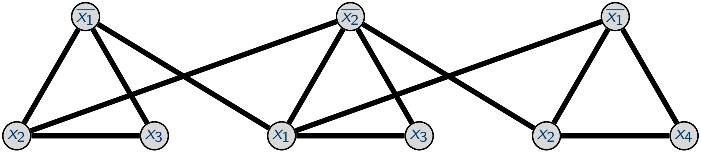
3-SAT $\leq_P$ Independent set

G contains independent set of size $k = |\Phi|$ iff Φ is satisfiable.

$\Rightarrow$ Let S be independent set of size k.

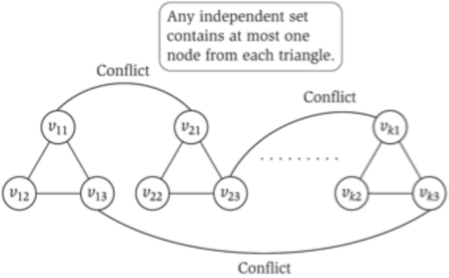
- S must contain exactly one vertex in each triangle.
- Set these literals to true.
- Truth assignment is consistent and all clauses are satisfied.

$\Leftarrow$ Given satisfying assignment, select one true literal from each triangle. This is an independent set of size k.



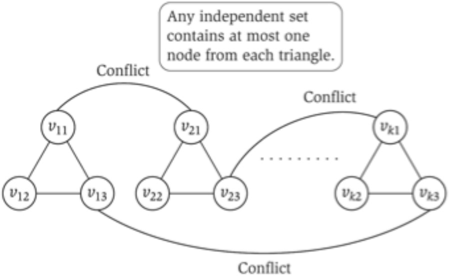
$$\Phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_2} \vee x_1 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$$

Proving 3-SAT $\leq_P$ Independent Set



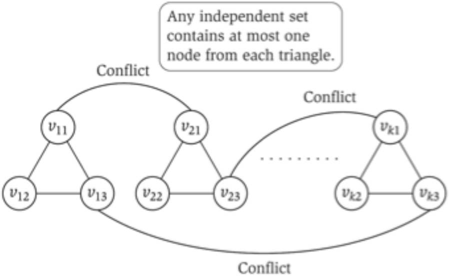
- We are given an instance of 3-SAT with k clauses of length three over n variables.
- Construct a graph $G = (V, E)$ with $3k$ nodes.
  - For each clause $C_i, 1 \leq i \leq k$, add a triangle of three nodes v_{i1}, v_{i2}, v_{i3} and three edges to G.
  - Label each node $v_{ij}, 1 \leq j \leq 3$ with the j-th term in C_i.
  - Add an edge between each pair of nodes whose labels correspond to terms that conflict.

Proving 3-SAT $\leq_P$ Independent Set



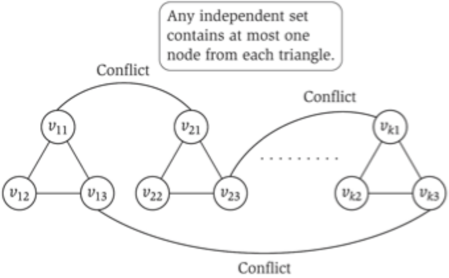
- Claim: 3-SAT instance is satisfiable iff G has an independent set of size at least k.

Proving 3-SAT $\leq_P$ Independent Set



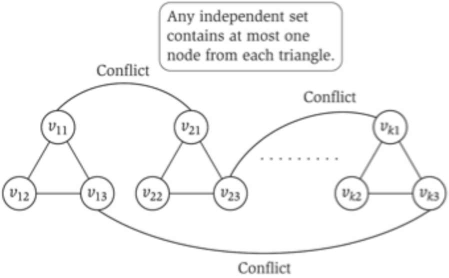
- Claim: 3-SAT instance is satisfiable iff G has an independent set of size at least k.
- Satisfiable assignment $\rightarrow$ independent set of size $\geq k$

Proving 3-SAT $\leq_P$ Independent Set



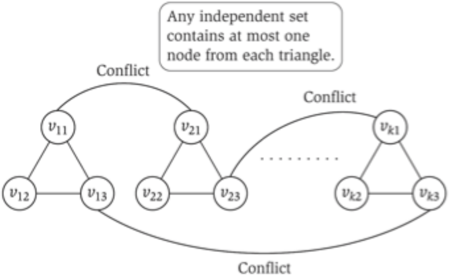
- Claim: 3-SAT instance is satisfiable iff G has an independent set of size at least k.
- Satisfiable assignment $\rightarrow$ independent set of size $\geq k$ Each triangle in G has at least one node whose label evaluates to 1. These nodes form an independent set of size k. Why?

Proving 3-SAT $\leq_P$ Independent Set



- Claim: 3-SAT instance is satisfiable iff G has an independent set of size at least k.
- Satisfiable assignment $\rightarrow$ independent set of size $\geq k$ Each triangle in G has at least one node whose label evaluates to 1. These nodes form an independent set of size k. Why?
- Independent set of size $\geq k$ $\rightarrow$ satisfiable assignment

Proving 3-SAT $\leq_P$ Independent Set



- Claim: 3-SAT instance is satisfiable iff G has an independent set of size at least k.
- Satisfiable assignment $\rightarrow$ independent set of size $\geq k$ Each triangle in G has at least one node whose label evaluates to 1. These nodes form an independent set of size k. Why?
- Independent set of size $\geq k \rightarrow$ satisfiable assignment the size of this set is k. How do we construct a satisfying truth assignment from the nodes in the independent set?

Transitivity of Reductions

Basic reduction strategies.

- Simple equivalence: Independent set $\equiv_P$ Vertex cover.
- Special case to general case: Vertex cover $\leq_P$ Set cover.
- Encoding with gadgets: 3-SAT $\leq_P$ Independent set.

Transitivity of Reductions

Basic reduction strategies.

- Simple equivalence: INDEPENDENT SET $\equiv_P$ VERTEX COVER.
- Special case to general case: VERTEX COVER $\leq_P$ SET COVER.
- Encoding with gadgets: 3-SAT $\leq_P$ INDEPENDENT SET.

If $\mathrm{Z} \leq_P \mathrm{Y}$ and $\mathrm{Y} \leq_P \mathrm{X}$, then $\mathrm{Z} \leq_P \mathrm{X}$.

Transitivity of Reductions

Basic reduction strategies.

- Simple equivalence: INDEPENDENT SET $\equiv_P$ VERTEX COVER.
- Special case to general case: VERTEX COVER $\leq_P$ SET COVER.
- Encoding with gadgets: 3-SAT $\leq_P$ INDEPENDENT SET.

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

3-SAT $\leq_P$ INDEPENDENT SET $\leq_P$ VERTEX COVER $\leq_P$ SET COVER

Teoria dos Grafos e Computabilidade

— $\mathcal{NP}$ —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Laboratory of Image and Multimedia Data Science – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Finding vs. Certifying

- Is it easy to check if a given set of vertices in an undirected graph forms an independent set of size at least k?
- Is it easy to check if a particular truth assignment satisfies a set of clauses?

Finding vs. Certifying

- Is it easy to check if a given set of vertices in an undirected graph forms an independent set of size at least k?
- Is it easy to check if a particular truth assignment satisfies a set of clauses?
- We draw a contrast between finding a solution and checking a solution (in polynomial time).

We have not been able to develop efficient algorithms to solve many decision problems, let us turn our attention to whether we can check if a proposed solution is correct.

Problems, Algorithms, and Strings

- Encode input to a computational problem as a finite binary string s of length $|s|$.
- Identify a decision problem X with the set of strings for which the answer is yes,

Problems, Algorithms, and Strings

- Encode input to a computational problem as a finite binary string s of length $|s|$.
- Identify a decision problem X with the set of strings for which the answer is yes, e.g., $\mathrm{PRIMES} = \{2, 3, 5, 7, 11, \ldots\}$.

Problems, Algorithms, and Strings

- Encode input to a computational problem as a finite binary string s of length $|s|$.
- Identify a decision problem X with the set of strings for which the answer is yes, e.g., $\mathrm{PRIMES} = \{2, 3, 5, 7, 11, \ldots\}$.
- An algorithm A for a decision problem receives an input string s and returns $A(s) \in \{\mathtt{yes}, \mathtt{no}\}$.
- A solves the problem X if for every string s, $A(s) = \mathrm{yes}$ iff $s \in X$.

Problems, Algorithms, and Strings

- Encode input to a computational problem as a finite binary string s of length $|s|$.
- Identify a decision problem X with the set of strings for which the answer is yes, e.g., $\mathrm{PRIMES} = \{2, 3, 5, 7, 11, \ldots\}$.
- An algorithm A for a decision problem receives an input string s and returns $A(s) \in \{\texttt{yes}, \texttt{no}\}$.
- A solves the problem X if for every string s, $A(s) = \texttt{yes}$ iff $s \in X$.
- A has a polynomial running time if there is a polynomial function $p(\cdot)$ such that for every input string s, A terminates on s in at most $O(p(|s|))$ steps,

Problems, Algorithms, and Strings

- Encode input to a computational problem as a finite binary string s of length $|s|$.
- Identify a decision problem X with the set of strings for which the answer is yes, e.g., $\mathrm{PRIMES} = \{2, 3, 5, 7, 11, \ldots\}$.
- An algorithm A for a decision problem receives an input string s and returns $A(s) \in \{\mathtt{yes}, \mathtt{no}\}$.
- A solves the problem X if for every string s, $A(s) = \mathtt{yes}$ iff $s \in X$.
- A has a polynomial running time if there is a polynomial function $p(\cdot)$ such that for every input string s, A terminates on s in at most $O(p(|s|))$ steps, e.g., there is an algorithm such that $p(|s|) = |s|^8$ for PRIMES
- $\mathcal{P}$: set of problems X for which there is a polynomial time algorithm.

Efficient Certification

- A **checking algorithm** for a decision problem X has a different structure from an algorithm that solves X.
- Checking algorithm needs input string s as well as a separate **certificate** string t that contains evidence that $s \in X$.

Efficient Certification

- A checking algorithm for a decision problem X has a different structure from an algorithm that solves X.
- Checking algorithm needs input string s as well as a separate certificate string t that contains evidence that $s \in X$.
- An algorithm B is an efficient certifier for a problem X if
  1. B is a polynomial time algorithm that takes two inputs s and t and
  2. there is a polynomial function p so that for every string s, we have $s \in X$ iff there exists a string t such that $|t| \leq p(|s|)$ and $B(s, t) = \text{yes}$.

Efficient Certification

- A checking algorithm for a decision problem X has a different structure from an algorithm that solves X.
- Checking algorithm needs input string s as well as a separate certificate string t that contains evidence that $s \in X$.
- An algorithm B is an efficient certifier for a problem X if
  1. B is a polynomial time algorithm that takes two inputs s and t and
  2. there is a polynomial function p so that for every string s, we have $s \in X$ iff there exists a string t such that $|t| \leq p(|s|)$ and $B(s, t) = \text{yes}$.
- Certifier's job is to take a candidate short proof (t) that $s \in X$ and check in polynomial time whether t is a correct proof.

Certifier does not care about how to find these proofs.

$\mathcal{NP}$ is the set of all problems for which there exists an efficient certifier.

$\mathcal{NP}$ is the set of all problems for which there exists an efficient certifier.

3-SAT $\in \mathcal{NP}$

$\mathcal{NP}$ is the set of all problems for which there exists an efficient certifier.

3-SAT $\in \mathcal{NP}$

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

Certificate $x_1 = 1$, $x_2 = 1$, $x_3 = 0$ and $x_4 = 1$

$\mathcal{NP}$

$\mathcal{NP}$ is the set of all problems for which there exists an efficient certifier.

3-SAT $\in \mathcal{NP}$ *t is a truth assignment; B evaluates the clauses with respect to the assignment.*

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

Certificate $x_1 = 1$, $x_2 = 1$, $x_3 = 0$ and $x_4 = 1$

$\mathcal{NP}$

$\mathcal{NP}$ is the set of all problems for which there exists an efficient certifier.

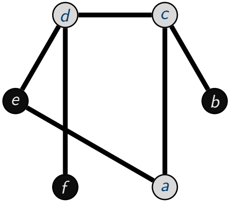
3-SAT $\in \mathcal{NP}$ *t is a truth assignment; B evaluates the clauses with respect to the assignment.*

Independent Set $\in \mathcal{NP}$

$\mathcal{NP}$ is the set of all problems for which there exists an efficient certifier.

3-SAT $\in \mathcal{NP}$ *t is a truth assignment; B evaluates the clauses with respect to the assignment.*

INDEPENDENT SET $\in \mathcal{NP}$

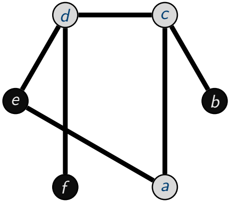


$\mathcal{NP}$

$\mathcal{NP}$ is the set of all problems for which there exists an efficient certifier.

3-SAT $\in \mathcal{NP}$ *t is a truth assignment; B evaluates the clauses with respect to the assignment.*

Independent Set $\in \mathcal{NP}$ *t is a set of at least k vertices; B checks that no pair of these vertices are connected by an edge.*



$\mathcal{NP}$ is the set of all problems for which there exists an efficient certifier.

3-SAT $\in \mathcal{NP}$ *t is a truth assignment; B evaluates the clauses with respect to the assignment.*

Independent Set $\in \mathcal{NP}$ *t is a set of at least k vertices; B checks that no pair of these vertices are connected by an edge.*

Set Cover $\in \mathcal{NP}$

$\mathcal{NP}$

$\mathcal{NP}$ is the set of all problems for which there exists an efficient certifier.

3-SAT $\in \mathcal{NP}$ *t is a truth assignment; B evaluates the clauses with respect to the assignment.*

Independent Set $\in \mathcal{NP}$ *t is a set of at least k vertices; B checks that no pair of these vertices are connected by an edge.*

Set Cover $\in \mathcal{NP}$

- $U = \{1, 2, 3, 4, 5, 6, 7\}$ and $k = 2$

$S_1 = \{3, 7\}$	$S_4 = \{2, 4\}$
$S_2 = \{3, 4, 5, 6\}$	$S_5 = \{5\}$
$S_3 = \{1\}$	$S_6 = \{1, 2, 6, 7\}$

$\mathcal{NP}$

$\mathcal{NP}$ is the set of all problems for which there exists an efficient certifier.

3-SAT $\in \mathcal{NP}$ *t is a truth assignment; B evaluates the clauses with respect to the assignment.*

Independent Set $\in \mathcal{NP}$ *t is a set of at least k vertices; B checks that no pair of these vertices are connected by an edge.*

Set Cover $\in \mathcal{NP}$ *t is a list of k sets from the collection; B checks if their union is U.*

- $U = \{1, 2, 3, 4, 5, 6, 7\}$ and $k = 2$

$S_1 = \{3, 7\}$ $\quad S_4 = \{2, 4\}$

$\boxed{S_2 = \{3, 4, 5, 6\}}$ $\quad S_5 = \{5\}$

$S_3 = \{1\}$ $\quad \boxed{S_6 = \{1, 2, 6, 7\}}$

$\mathcal{P}$ vs. $\mathcal{NP}$

$\mathcal{P}$ *Decision problems for which there is a* *poly-time algorithm*

EXP *Decision problems for which there is an* *exponential-time algorithm*

$\mathcal{NP}$ *Decision problems for which there is a* *poly-time certifier*

$\mathcal{P}$ vs. $\mathcal{NP}$

$\mathcal{P}$ *Decision problems for which there is a* *poly-time algorithm*

EXP *Decision problems for which there is an* *exponential-time algorithm*

$\mathcal{NP}$ *Decision problems for which there is a* *poly-time certifier*

- $\mathcal{P} \subseteq \mathcal{NP}$

$\mathcal{P}$ vs. $\mathcal{NP}$

$\mathcal{P}$ *Decision problems for which there is a* *poly-time algorithm*

EXP *Decision problems for which there is an* *exponential-time algorithm*

$\mathcal{NP}$ *Decision problems for which there is a* *poly-time certifier*

- $\mathcal{P} \subseteq \mathcal{NP}$ If $X \in P$, then there is a polynomial time algorithm A that solves X. B ignores t and returns $A(s)$. Why is B an efficient certifier?

$\mathcal{P}$ vs. $\mathcal{NP}$

$\mathcal{P}$ *Decision problems for which there is a* *poly-time algorithm*

EXP *Decision problems for which there is an* *exponential-time algorithm*

$\mathcal{NP}$ *Decision problems for which there is a* *poly-time certifier*

- $\mathcal{P} \subseteq \mathcal{NP}$ If $X \in P$, then there is a polynomial time algorithm A that solves X. B ignores t and returns $A(s)$. Why is B an efficient certifier?
- Is $\mathcal{P} = \mathcal{NP}$ or is $\mathcal{NP} - \mathcal{P} \neq \emptyset$.

$\mathcal{P}$ vs. $\mathcal{NP}$

$\mathcal{P}$ *Decision problems for which there is a* *poly-time algorithm*

EXP *Decision problems for which there is an* *exponential-time algorithm*

$\mathcal{NP}$ *Decision problems for which there is a* *poly-time certifier*

- $\mathcal{P} \subseteq \mathcal{NP}$ If $X \in P$, then there is a polynomial time algorithm A that solves X. B ignores t and returns $A(s)$. Why is B an efficient certifier?
- Is $\mathcal{P} = \mathcal{NP}$ or is $\mathcal{NP} - \mathcal{P} \neq \emptyset$.

One of the major unsolved problems in computer science.

$\mathcal{P}$ vs. $\mathcal{NP}$

$\mathcal{P}$ *Decision problems for which there is a* *poly-time algorithm*

EXP *Decision problems for which there is an* *exponential-time algorithm*

$\mathcal{NP}$ *Decision problems for which there is a* *poly-time certifier*

- $\mathcal{P} \subseteq \mathcal{NP}$ If $X \in P$, then there is a polynomial time algorithm A that solves X. B ignores t and returns $A(s)$. Why is B an efficient certifier?
- Is $\mathcal{P} = \mathcal{NP}$ or is $\mathcal{NP} - \mathcal{P} \neq \emptyset$.

  One of the major unsolved problems in computer science.
- $\mathcal{NP} \subseteq EXP$.

$\mathcal{P}$ vs. $\mathcal{NP}$

$\mathcal{P}$ *Decision problems for which there is a* *poly-time algorithm*

EXP *Decision problems for which there is an* *exponential-time algorithm*

$\mathcal{NP}$ *Decision problems for which there is a* *poly-time certifier*

- $\mathcal{P} \subseteq \mathcal{NP}$ If $X \in P$, then there is a polynomial time algorithm A that solves X. B ignores t and returns $A(s)$. Why is B an efficient certifier?
- Is $\mathcal{P} = \mathcal{NP}$ or is $\mathcal{NP} - \mathcal{P} \neq \emptyset$.

  One of the major unsolved problems in computer science.
- $\mathcal{NP} \subseteq EXP$. Consider any problem X in $\mathcal{NP}$.
  - By definition, there exists a poly-time certifier C(s, t) for X.
  - To solve input s, run C(s, t) on all strings t with $|t| \leq p(|s|)$.
  - Return yes, if C(s, t) returns yes for any of these.

$\mathcal{P}$ vs. $\mathcal{NP}$

A decision problem belongs to the class $\mathcal{P}$ if there is a solution algorithm with a running time that is polynomial in the input size

$\mathcal{P}$ vs. $\mathcal{NP}$

A decision problem belongs to the class $\mathcal{P}$ if there is a solution algorithm with a running time that is polynomial in the input size

A decision problem belongs to the class $\mathcal{NP}$ if we can check whether a given solution leads to 'yes' can be done in polynomial time with respect to the size of (x,y).

$\mathcal{P}$ vs. $\mathcal{NP}$

A decision problem belongs to the class $\mathcal{P}$ if there is a solution algorithm with a running time that is polynomial in the input size

Class $\mathcal{P}$ – Class of decision problems, for which there exists a Deterministic Turing Machine that can solve any instance in polynomial time.

A decision problem belongs to the class $\mathcal{NP}$ if we can check whether a given solution leads to 'yes' can be done in polynomial time with respect to the size of (x,y).

$\mathcal{P}$ vs. $\mathcal{NP}$

A decision problem belongs to the class $\mathcal{P}$ if there is a solution algorithm with a running time that is polynomial in the input size

Class $\mathcal{P}$ – Class of decision problems, for which there exists a Deterministic Turing Machine that can solve any instance in polynomial time.

A decision problem belongs to the class $\mathcal{NP}$ if we can check whether a given solution leads to 'yes' can be done in polynomial time with respect to the size of (x,y).

Class $\mathcal{NP}$ – Class of decision problems, for which there exists a Non- Deterministic Turing Machine that can solve any yes instance in polynomial time. The machine guesses a yes solution and then verifies that it is a yes solution

Never tell to an expert in *Computational Complexity – tractability –* that you think that $\mathcal{NP}$ stands for Non Polynomial

$\mathcal{NP}$ STANDS for Non-deterministic Polynomial

$\mathcal{P}$ vs. $\mathcal{NP}$

Does $\mathcal{P} = \mathcal{NP}$? [Cook 1971, Edmonds, Levin, Yablonski, Gödel]

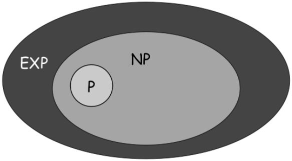
$\mathcal{P}$ vs. $\mathcal{NP}$

Does $\mathcal{P} = \mathcal{NP}$? [Cook 1971, Edmonds, Levin, Yablonski, Gödel]

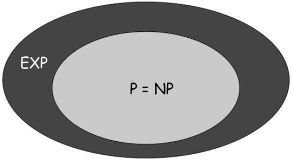
Is the decision problem as easy as the certification problem?

Does $\mathcal{P} = \mathcal{NP}$? [Cook 1971, Edmonds, Levin, Yablonski, Gödel]

Is the decision problem as easy as the certification problem?



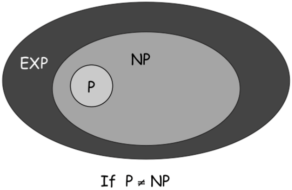
If P ≠ NP



If P = NP

Does $\mathcal{P} = \mathcal{NP}$? [Cook 1971, Edmonds, Levin, Yablonski, Gödel]

Is the decision problem as easy as the certification problem?



If yes Efficient algorithms for 3-COLOR, TSP, FACTOR, SAT, $\cdots$.

If no No efficient algorithms possible for 3-COLOR, TSP, SAT, $\cdots$.

Does $\mathcal{P} = \mathcal{NP}$? [Cook 1971, Edmonds, Levin, Yablonski, Gödel]

Is the decision problem as easy as the certification problem?

If yes Efficient algorithms for 3-COLOR, TSP, FACTOR, SAT, $\cdots$.

If no No efficient algorithms possible for 3-COLOR, TSP, SAT, $\cdots$.

Consensus opinion on P = NP? Probably no.

$\mathcal{P}$ vs. $\mathcal{NP}$

The Simpson's: P = NP?

Copyright © 1990, Matt Groening

$\mathcal{P}$ vs. $\mathcal{NP}$

Futurama: P = NP?

Teoria dos Grafos e Computabilidade

— $\mathcal{NP}$-Complete —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Laboratory of Image and Multimedia Data Science – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Feb 2025

$\mathcal{NP}$-Complete Problems

- What are the hardest problems in $\mathcal{NP}$?

$\mathcal{NP}$-Complete Problems

- What are the hardest problems in $\mathcal{NP}$?
- A problem X is $\mathcal{NP}$-Complete if
  1. $X \in \mathcal{NP}$ and
  2. for *every* problem $Y \in \mathcal{NP}$, $Y \leq_P X$.

$\mathcal{NP}$-Complete Problems

- What are the hardest problems in $\mathcal{NP}$?
- A problem X is **$\mathcal{NP}$-Complete** if
  1. $X \in \mathcal{NP}$ and
  2. for *every* problem $Y \in \mathcal{NP}$, $Y \leq_P X$.
- Suppose X is $\mathcal{NP}$-Complete. Then X can be solved in polynomial-time **iff $\mathcal{P} = \mathcal{NP}$**.

$\mathcal{NP}$-Complete Problems

- What are the hardest problems in $\mathcal{NP}$?
- A problem X is $\mathcal{NP}$-Complete if
  1. $X \in \mathcal{NP}$ and
  2. for *every* problem $Y \in \mathcal{NP}$, $Y \leq_P X$.
- Suppose X is $\mathcal{NP}$-Complete. Then X can be solved in polynomial-time iff $\mathcal{P} = \mathcal{NP}$.

Corollary: If there is any problem in $\mathcal{NP}$ that cannot be solved in polynomial time, then no $\mathcal{NP}$-Complete problem can be solved in polynomial time.

$\mathcal{NP}$-Complete Problems

- What are the hardest problems in $\mathcal{NP}$?
- A problem X is $\mathcal{NP}$-Complete if
  1. $X \in \mathcal{NP}$ and
  2. for *every* problem $Y \in \mathcal{NP}$, $Y \leq_P X$.
- Suppose X is $\mathcal{NP}$-Complete. Then X can be solved in polynomial-time iff $\mathcal{P} = \mathcal{NP}$.

Corollary: If there is any problem in $\mathcal{NP}$ that cannot be solved in polynomial time, then no $\mathcal{NP}$-Complete problem can be solved in polynomial time.

- Are there any $\mathcal{NP}$-Complete problems?
  1. Perhaps there are two problems X_1 and X_2 in $\mathcal{NP}$ such that there is no problem $X \in \mathcal{NP}$ where $X_1 \leq_P X$ and $X_2 \leq_P X$.
  2. Perhaps there is a sequence of problems $X_1, X_2, X_3, \ldots$ in $\mathcal{NP}$, each strictly harder than the previous one.

$\mathcal{NP}$-Complete Problems

- A problem X is **$\mathcal{NP}$-Hard** if
  1. for *every* problem $Y \in \mathcal{NP}$, $Y \leq_P X$.

$\mathcal{NP}$-Complete Problems

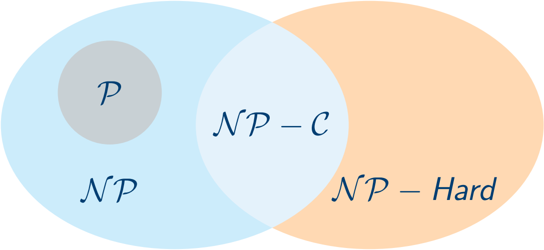
- A problem X is $\mathcal{NP}$-Hard if
  1. for *every* problem $Y \in \mathcal{NP}$, $Y \leq_P X$.

- A problem X is $\mathcal{NP}$-Complete if
  1. $X \in \mathcal{NP}$ and
  2. for *every* problem $Y \in \mathcal{NP}$, $Y \leq_P X$.

$\mathcal{NP}$-Complete Problems

- A problem X is $\mathcal{NP}$-Hard if
  1. for *every* problem $Y \in \mathcal{NP}$, $Y \leq_P X$.

- A problem X is $\mathcal{NP}$-Complete if
  1. $X \in \mathcal{NP}$ and
  2. for *every* problem $Y \in \mathcal{NP}$, $Y \leq_P X$.



- Cook-Levin Theorem Circuit Satisfiability is $\mathcal{NP}$-Complete.

- Cook-Levin Theorem Circuit Satisfiability is $\mathcal{NP}$-Complete.
- A **circuit** K is a labelled, directed acyclic graph such that
  1. the **sources** in K are labelled with constants (0 or 1) or the name of a distinct variable (the **inputs** to the circuit).
  2. every other node is labelled with one Boolean operator $\wedge$, $\vee$, or $\neg$.
  3. a single node with no outgoing edges represents the **output** of K.

Circuit Satisfiability

INSTANCE	A circuit K.
QUESTION	Is there a truth assignment to the inputs that causes the output to have value 1?

Proving CIRCUIT SATISFIABILITY is $\mathcal{NP}$-Complete

- Take an arbitrary problem $X \in \mathcal{NP}$ and show that
  $\boxed{X \leq_P \text{CIRCUIT SATISFIABILITY}}$.

Proving CIRCUIT SATISFIABILITY is $\mathcal{NP}$-Complete

- Take an arbitrary problem $X \in \mathcal{NP}$ and show that $\boxed{X \leq_P \text{CIRCUIT SATISFIABILITY}}$.
- Claim we will not prove: any algorithm that takes a fixed number n of bits as input and produces a yes/no answer
  1. can be represented by an equivalent circuit and
  2. if the running time of the algorithm is polynomial in n, the size of the circuit is a polynomial in n.

Proving CIRCUIT SATISFIABILITY is $\mathcal{NP}$-Complete

- Take an arbitrary problem $X \in \mathcal{NP}$ and show that
  $\boxed{X \leq_P \text{CIRCUIT SATISFIABILITY}}$.
- Claim we will not prove: any algorithm that takes a fixed number n of bits as input and produces a yes/no answer
  1. can be represented by an equivalent circuit and
  2. if the running time of the algorithm is polynomial in n, the size of the circuit is a polynomial in n.
- To show $X \leq_P$ CIRCUIT SATISFIABILITY, given an input s of length n, we want to determine whether $s \in X$ using a black box that solves CIRCUIT SATISFIABILITY.

Proving CIRCUIT SATISFIABILITY is $\mathcal{NP}$-Complete

- Take an arbitrary problem $X \in \mathcal{NP}$ and show that $\boxed{X \leq_P \text{CIRCUIT SATISFIABILITY}}$.
- Claim we will not prove: any algorithm that takes a fixed number n of bits as input and produces a yes/no answer
  1. can be represented by an equivalent circuit and
  2. if the running time of the algorithm is polynomial in n, the size of the circuit is a polynomial in n.
- To show $X \leq_P$ CIRCUIT SATISFIABILITY, given an input s of length n, we want to determine whether $s \in X$ using a black box that solves CIRCUIT SATISFIABILITY.
- What do we know about X?

- Take an arbitrary problem $X \in \mathcal{NP}$ and show that $\boxed{X \leq_P \text{CIRCUIT SATISFIABILITY}}$.
- Claim we will not prove: any algorithm that takes a fixed number n of bits as input and produces a yes/no answer
  1. can be represented by an equivalent circuit and
  2. if the running time of the algorithm is polynomial in n, the size of the circuit is a polynomial in n.
- To show $X \leq_P$ CIRCUIT SATISFIABILITY, given an input s of length n, we want to determine whether $s \in X$ using a black box that solves CIRCUIT SATISFIABILITY.
- What do we know about X? It has an efficient certifier $B(\cdot, \cdot)$.

- Take an arbitrary problem $X \in \mathcal{NP}$ and show that $X \leq_P$ Circuit Satisfiability.
- Claim we will not prove: any algorithm that takes a fixed number n of bits as input and produces a yes/no answer
  1. can be represented by an equivalent circuit and
  2. if the running time of the algorithm is polynomial in n, the size of the circuit is a polynomial in n.
- To show $X \leq_P$ Circuit Satisfiability, given an input s of length n, we want to determine whether $s \in X$ using a black box that solves Circuit Satisfiability.
- What do we know about X? It has an efficient certifier $B(\cdot, \cdot)$.
- To determine whether $s \in X$, we ask

  Is there a string t of length $p(n)$ such that $B(s, t) =$ yes?'

- To determine whether $s \in X$, we ask

  Is there a string t of length $p(|s|)$ such that $B(s, t) = \text{yes}$?

Proving CIRCUIT SATISFIABILITY is $\mathcal{NP}$-Complete

- To determine whether $s \in X$, we ask

  Is there a string t of length $p(|s|)$ such that $B(s, t) = \text{yes}$?

- View $B(\cdot, \cdot)$ as an algorithm on $n + p(n)$ bits.
- Convert B to a polynomial-sized circuit K with $n + p(n)$ sources.
  1. First n sources are hard-coded with the bits of s.
  2. The remaining $p(n)$ sources labelled with variables representing the bits of t.

- To determine whether $s \in X$, we ask

  Is there a string t of length $p(|s|)$ such that $B(s, t) = \text{yes}$?
- View $B(\cdot, \cdot)$ as an algorithm on $n + p(n)$ bits.
- Convert B to a polynomial-sized circuit K with $n + p(n)$ sources.
  1. First n sources are hard-coded with the bits of s.
  2. The remaining $p(n)$ sources labelled with variables representing the bits of t.
- $s \in X$ iff there is an assignment of the input bits of K that makes K satisfiable.

- Does a graph G on n nodes have a two-node independent set?

Example of Transformation to Circuit Satisfiability

- Does a graph G on n nodes have a two-node independent set?
- s encodes the graph G with $\binom{n}{2}$ bits.
- t encodes the independent set with n bits.
- Certifier needs to check if
  1. at least two bits in t are set to 1 and
  2. no two bits in t are set to 1 if they form the ends of an edge (the corresponding bit in s is set to 1).

Proving Other Problems $\mathcal{NP}$-Complete

If Y is $\mathcal{NP}$-Complete and $X \in \mathcal{NP}$ such that $Y \leq_P X$, then X is $\mathcal{NP}$-Complete.

Proving Other Problems $\mathcal{NP}$-Complete

If Y is $\mathcal{NP}$-Complete and $X \in \mathcal{NP}$ such that $Y \leq_P X$, then X is $\mathcal{NP}$-Complete.

- Given a new problem X, a general strategy for proving that X is $\mathcal{NP}$-Complete can be defined as follows

Proving Other Problems $\mathcal{NP}$-Complete

If Y is $\mathcal{NP}$-Complete and $X \in \mathcal{NP}$ such that $Y \leq_P X$, then X is $\mathcal{NP}$-Complete.

- Given a new problem X, a general strategy for proving that X is $\mathcal{NP}$-Complete can be defined as follows
  1. Prove that $X \in \mathcal{NP}$.
  2. Select a problem Y known to be $\mathcal{NP}$-Complete.
  3. Prove that $Y \leq_P X$.

Proving Other Problems $\mathcal{NP}$-Complete

If Y is $\mathcal{NP}$-Complete and $X \in \mathcal{NP}$ such that $Y \leq_P X$, then X is $\mathcal{NP}$-Complete.

- Given a new problem X, a general strategy for proving that X is $\mathcal{NP}$-Complete can be defined as follows
  1. Prove that $X \in \mathcal{NP}$.
  2. Select a problem Y known to be $\mathcal{NP}$-Complete.
  3. Prove that $Y \leq_P X$.
- If we use Karp reductions, we can refine the strategy:
  1. Prove that $X \in \mathcal{NP}$.
  2. Select a problem Y known to be $\mathcal{NP}$-Complete.
  3. Consider an arbitrary instance s_Y of problem Y. Show how to construct, in polynomial time, an instance s_X of problem X such that
     (a) If $s_Y \in Y$, then $s_X \in X$ and
     (b) If $s_X \in X$, then $s_Y \in Y$.

$\mathcal{NP}$-Completeness