

NP and Computational Intractability

T. M. Murali

April 23, 28, 2014

Algorithm Design

► Patterns

- Greed.
- Divide-and-conquer.
- Dynamic programming.
- Duality.

$O(n \log n)$ interval scheduling.

$O(n \log n)$ closest pair of points.

$O(n^2)$ edit distance.

$O(n^3)$ maximum flow and minimum cuts.

Algorithm Design

► Patterns

- Greed.
- Divide-and-conquer.
- Dynamic programming.
- Duality.
- Reductions.
- Local search.
- Randomization.

$O(n \log n)$ interval scheduling.

$O(n \log n)$ closest pair of points.

$O(n^2)$ edit distance.

$O(n^3)$ maximum flow and minimum cuts.

Algorithm Design

► Patterns

- Greed.
- Divide-and-conquer.
- Dynamic programming.
- Duality.
- Reductions.
- Local search.
- Randomization.

$O(n \log n)$ interval scheduling.

$O(n \log n)$ closest pair of points.

$O(n^2)$ edit distance.

$O(n^3)$ maximum flow and minimum cuts.

► “Anti-patterns”

- NP-completeness.
- PSPACE-completeness.
- Undecidability.

$O(n^k)$ algorithm unlikely.

$O(n^k)$ certification algorithm unlikely.

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

Shortest path

Matching

Minimum cut

2-SAT

Planar four-colour

Bipartite vertex cover

Primality testing

Probably not

Longest path

3-D matching

Maximum cut

3-SAT

Planar three-colour

Vertex cover

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!

Polynomial-Time Reduction

- ▶ 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

- ▶ 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 any 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 .

Polynomial-Time Reduction

- ▶ 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 any 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 is “finding the largest matching in a bipartite graph.” X is “Compute the maximum flow in a network.”
 - ▶ Y is “Find the best segmentation of an image into foreground and background.” X is “Compute the minimum s - t cut in a network.”

Polynomial-Time Reduction

- ▶ 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 any 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 is “finding the largest matching in a bipartite graph.” X is “Compute the maximum flow in a network.”
 - ▶ Y is “Find the best segmentation of an image into foreground and background.” X is “Compute the minimum s - t cut in a network.”
- ▶ $Y \leq_P X$ 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.
- ▶ Informally: If Y is hard, and we can show that Y reduces to X , then the hardness “spreads” to X .

Reduction Strategies

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

Optimisation versus Decision Problems

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

Optimisation versus Decision Problems

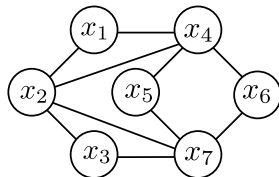
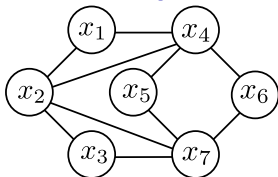
- ▶ So far, we have developed algorithms that solve optimisation problems.
 - ▶ Compute the *largest* flow.
 - ▶ Find the *closest* pair of points.
 - ▶ Find the schedule with the *least* completion time.
- ▶ Now, we will focus on *decision versions* of problems, e.g., is there a flow with value at least k , for a given value of k ?
- ▶ Decision problem: answer to every input is yes or no.

PRIMES

INSTANCE: A natural number n

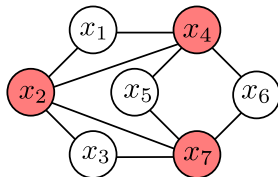
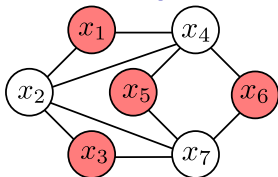
QUESTION: Is n prime?

Independent Set and Vertex Cover



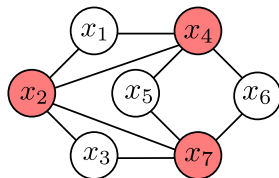
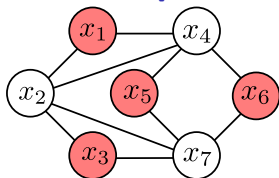
- ▶ Given an undirected graph $G(V, E)$, a subset $S \subseteq V$ is an *independent set* if no two vertices in S are connected by an edge.
- ▶ Given an undirected graph $G(V, E)$, a subset $S \subseteq V$ is a *vertex cover* if every edge in E is incident on at least one vertex in S .

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 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

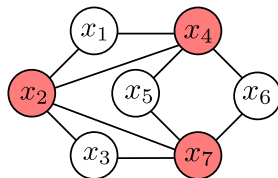
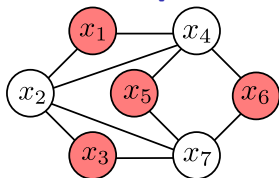
QUESTION: Does G contain an independent set of size $\geq k$?

VERTEX COVER

INSTANCE: Undirected graph G and an integer l

QUESTION: Does G contain a vertex cover of size $\leq l$?

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 $\geq k$?

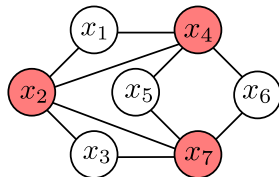
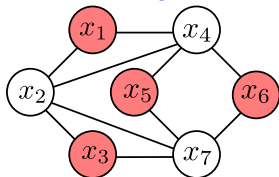
- ▶ Demonstrate simple equivalence between these two problems.

VERTEX COVER

INSTANCE: Undirected graph G and an integer l

QUESTION: Does G contain a vertex cover of size $\leq l$?

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 $\geq k$?

- ▶ Demonstrate simple equivalence between these two problems.
- ▶ Claim: $\text{INDEPENDENT SET} \leq_P \text{VERTEX COVER}$ and $\text{VERTEX COVER} \leq_P \text{INDEPENDENT SET}$.

VERTEX COVER

INSTANCE: Undirected graph G and an integer l

QUESTION: Does G contain a vertex cover of size $\leq l$?

Strategy for Proving Indep. Set \leq_P Vertex Cover

1. Start with an arbitrary instance of INDEPENDENT SET: an undirected graph $G(V, E)$ and an integer k .
2. From $G(V, E)$ and k , create an instance of VERTEX COVER: an undirected graph $G'(V', E')$ and an integer l .
 - ▶ G' related to G in some way.
 - ▶ l can depend upon k and size of G .
3. Prove that $G(V, E)$ has an independent set of size $\geq k$ iff $G'(V', E')$ has a vertex cover of size $\leq l$.

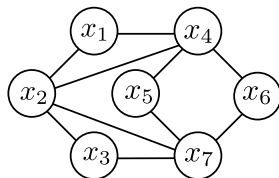
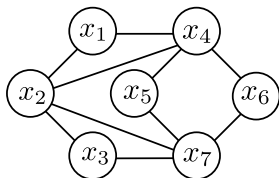
Strategy for Proving Indep. Set \leq_P Vertex Cover

1. Start with an arbitrary instance of INDEPENDENT SET: an undirected graph $G(V, E)$ and an integer k .
2. From $G(V, E)$ and k , create an instance of VERTEX COVER: an undirected graph $G'(V', E')$ and an integer l .
 - ▶ G' related to G in some way.
 - ▶ l can depend upon k and size of G .
3. Prove that $G(V, E)$ has an independent set of size $\geq k$ iff $G'(V', E')$ has a vertex cover of size $\leq l$.
 - ▶ Transformation and proof must be correct for all possible graphs $G(V, E)$ and all possible values of k .
 - ▶ Why is the proof an iff statement?

Strategy for Proving Indep. Set \leq_P Vertex Cover

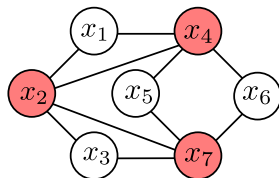
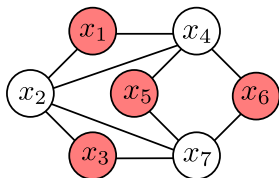
1. Start with an arbitrary instance of INDEPENDENT SET: an undirected graph $G(V, E)$ and an integer k .
2. From $G(V, E)$ and k , create an instance of VERTEX COVER: an undirected graph $G'(V', E')$ and an integer l .
 - ▶ G' related to G in some way.
 - ▶ l can depend upon k and size of G .
3. Prove that $G(V, E)$ has an independent set of size $\geq k$ **iff** $G'(V', E')$ has a vertex cover of size $\leq l$.
 - ▶ Transformation and proof must be correct for all possible graphs $G(V, E)$ and all possible values of k .
 - ▶ Why is the proof an **iff** statement? In the reduction, we are using black box for VERTEX COVER to solve INDEPENDENT SET.
 - (i) If there is an independent set size $\geq k$, we must be sure that there is a vertex cover of size $\leq l$, so that we know that the black box will find this vertex cover.
 - (ii) If the black box finds a vertex cover of size $\leq l$, we must be sure we can construct an independent set of size $\geq k$ from this vertex cover.

Proof that Independent Set \leq_P Vertex Cover



1. Arbitrary instance of INDEPENDENT SET: an undirected graph $G(V, E)$ and an integer k .
2. Let $|V| = n$.
3. Create an instance of VERTEX COVER: same undirected graph $G(V, E)$ and integer $n - k$.

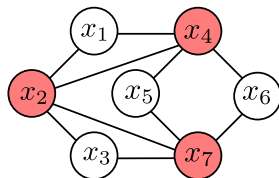
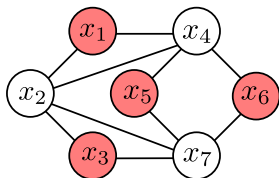
Proof that Independent Set \leq_P Vertex Cover



1. Arbitrary instance of INDEPENDENT SET: an undirected graph $G(V, E)$ and an integer k .
2. Let $|V| = n$.
3. Create an instance of VERTEX COVER: same undirected graph $G(V, E)$ and integer $n - k$.
4. Claim: $G(V, E)$ has an independent set of size $\geq k$ iff $G(V, E)$ has a vertex cover of size $\leq n - k$.

Proof: S is an independent set in G iff $V - S$ is a vertex cover in G .

Proof that Independent Set \leq_P Vertex Cover



1. Arbitrary instance of INDEPENDENT SET: an undirected graph $G(V, E)$ and an integer k .
2. Let $|V| = n$.
3. Create an instance of VERTEX COVER: same undirected graph $G(V, E)$ and integer $n - k$.
4. Claim: $G(V, E)$ has an independent set of size $\geq k$ iff $G(V, E)$ has a vertex cover of size $\leq n - k$.

Proof: S is an independent set in G iff $V - S$ is a vertex cover in G .

- Same idea proves that VERTEX COVER \leq_P INDEPENDENT SET

Vertex Cover and Set Cover

- ▶ INDEPENDENT SET 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, \dots, 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 ?

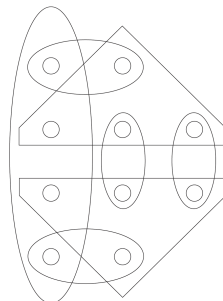
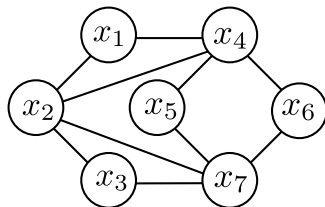


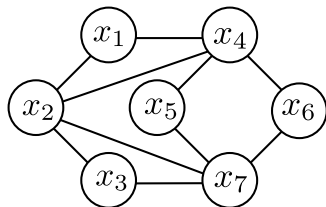
Figure 8.2 An instance of the Set Cover Problem.

Vertex Cover \leq_P Set Cover



- ▶ Input to VERTEX COVER: an undirected graph $G(V, E)$ and an integer k .
- ▶ Let $|V| = n$.
- ▶ Create an instance $\{U, \{S_1, S_2, \dots, S_n\}\}$ of SET COVER where

Vertex Cover \leq_P Set Cover



$$U = \{(x_1, x_2), (x_1, x_4), (x_2, x_3), (x_2, x_4), (x_2, x_7), (x_3, x_7), \\ (x_4, x_5), (x_5, x_6), (x_5, x_7), (x_6, x_7)\}$$

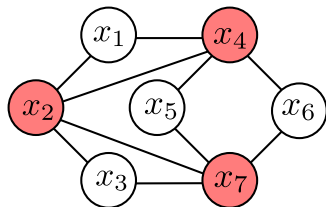
$$S_1 = \{(x_1, x_2), (x_1, x_4)\}$$

$$S_2 = \{(x_1, x_2), (x_2, x_3), (x_2, x_4), (x_2, x_7)\}$$

S_3, S_4, S_5, S_6 , and S_7 defined similarly.

- ▶ Input to VERTEX COVER: an undirected graph $G(V, E)$ and an integer k .
- ▶ Let $|V| = n$.
- ▶ Create an instance $\{U, \{S_1, S_2, \dots, S_n\}\}$ of SET COVER where
 - ▶ $U = E$,
 - ▶ for each vertex $i \in V$, create a set $S_i \subseteq U$ of the edges incident on i .

Vertex Cover \leq_P Set Cover



$$U = \{(x_1, x_2), (x_1, x_4), (x_2, x_3), (x_2, x_4), (x_2, x_7), (x_3, x_7), \\ (x_4, x_5), (x_5, x_6), (x_5, x_7), (x_6, x_7)\}$$

$$S_1 = \{(x_1, x_2), (x_1, x_4)\}$$

$$S_2 = \{(x_1, x_2), (x_2, x_3), (x_2, x_4), (x_2, x_7)\}$$

$S_3, S_4, S_5, S_6,$ and S_7 defined similarly.

- ▶ Input to VERTEX COVER: an undirected graph $G(V, E)$ and an integer k .
- ▶ Let $|V| = n$.
- ▶ Create an instance $\{U, \{S_1, S_2, \dots, S_n\}\}$ of SET COVER where
 - ▶ $U = E$,
 - ▶ for each vertex $i \in V$, create a set $S_i \subseteq U$ of the edges incident on i .
- ▶ Claim: U can be covered with fewer than k subsets iff G has a vertex cover with at most k nodes.
- ▶ Proof strategy:
 1. If $G(V, E)$ has a vertex cover of size at most k , then U can be covered with at most k subsets.
 2. If U can be covered with at most k subsets, then $G(V, E)$ has a vertex cover of size at most k .

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 $X = \{x_1, x_2, \dots, x_n\}$ 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 (or) of l distinct terms $t_1 \vee t_2 \vee \dots \vee t_l$.
- ▶ A *truth assignment* for X is a function $\nu : X \rightarrow \{0, 1\}$.
- ▶ An assignment *satisfies* a clause C if it causes at least one term in C to evaluate to 1 (since C is an or of terms).
- ▶ An assignment *satisfies* a collection of clauses C_1, C_2, \dots, C_k if it causes all clauses to evaluate to 1, i.e., $C_1 \wedge C_2 \wedge \dots \wedge C_k = 1$.
 - ▶ ν is a *satisfying assignment* with respect to C_1, C_2, \dots, C_k .
 - ▶ set of clauses C_1, C_2, \dots, C_k is *satisfiable*.

SAT and 3-SAT

SATISFIABILITY PROBLEM (SAT)

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

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

SAT and 3-SAT

3-SATISFIABILITY PROBLEM (3-SAT)

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

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

SAT and 3-SAT

3-SATISFIABILITY PROBLEM (SAT)

INSTANCE: A set of clauses C_1, C_2, \dots, C_k , each of length three, over a set $X = \{x_1, x_2, \dots, x_n\}$ 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.

Examples of 3-SAT

Example:

- ▶ $C_1 = x_1 \vee 0 \vee 0$
- ▶ $C_2 = x_2 \vee 0 \vee 0$
- ▶ $C_3 = \overline{x_1} \vee \overline{x_2} \vee 0$

Examples of 3-SAT

Example:

- ▶ $C_1 = x_1 \vee 0 \vee 0$
- ▶ $C_2 = x_2 \vee 0 \vee 0$
- ▶ $C_3 = \overline{x_1} \vee \overline{x_2} \vee 0$

1. Is $C_1 \wedge C_2$ satisfiable?

Examples of 3-SAT

Example:

- ▶ $C_1 = x_1 \vee 0 \vee 0$
- ▶ $C_2 = x_2 \vee 0 \vee 0$
- ▶ $C_3 = \overline{x_1} \vee \overline{x_2} \vee 0$

1. Is $C_1 \wedge C_2$ satisfiable? Yes, by $x_1 = 1, x_2 = 1$.

Examples of 3-SAT

Example:

- ▶ $C_1 = x_1 \vee 0 \vee 0$
- ▶ $C_2 = x_2 \vee 0 \vee 0$
- ▶ $C_3 = \overline{x_1} \vee \overline{x_2} \vee 0$

1. Is $C_1 \wedge C_2$ satisfiable? Yes, by $x_1 = 1, x_2 = 1$.
2. Is $C_1 \wedge C_3$ satisfiable?

Examples of 3-SAT

Example:

- ▶ $C_1 = x_1 \vee 0 \vee 0$
- ▶ $C_2 = x_2 \vee 0 \vee 0$
- ▶ $C_3 = \overline{x_1} \vee \overline{x_2} \vee 0$

1. Is $C_1 \wedge C_2$ satisfiable? Yes, by $x_1 = 1, x_2 = 1$.
2. Is $C_1 \wedge C_3$ satisfiable? Yes, by $x_1 = 1, x_2 = 0$.

Examples of 3-SAT

Example:

- ▶ $C_1 = x_1 \vee 0 \vee 0$
- ▶ $C_2 = x_2 \vee 0 \vee 0$
- ▶ $C_3 = \overline{x_1} \vee \overline{x_2} \vee 0$

1. Is $C_1 \wedge C_2$ satisfiable? Yes, by $x_1 = 1, x_2 = 1$.
2. Is $C_1 \wedge C_3$ satisfiable? Yes, by $x_1 = 1, x_2 = 0$.
3. Is $C_2 \wedge C_3$ satisfiable?

Examples of 3-SAT

Example:

- ▶ $C_1 = x_1 \vee 0 \vee 0$
- ▶ $C_2 = x_2 \vee 0 \vee 0$
- ▶ $C_3 = \overline{x_1} \vee \overline{x_2} \vee 0$

1. Is $C_1 \wedge C_2$ satisfiable? Yes, by $x_1 = 1, x_2 = 1$.
2. Is $C_1 \wedge C_3$ satisfiable? Yes, by $x_1 = 1, x_2 = 0$.
3. Is $C_2 \wedge C_3$ satisfiable? Yes, by $x_1 = 0, x_2 = 1$.

Examples of 3-SAT

Example:

- ▶ $C_1 = x_1 \vee 0 \vee 0$
- ▶ $C_2 = x_2 \vee 0 \vee 0$
- ▶ $C_3 = \overline{x_1} \vee \overline{x_2} \vee 0$

1. Is $C_1 \wedge C_2$ satisfiable? Yes, by $x_1 = 1, x_2 = 1$.
2. Is $C_1 \wedge C_3$ satisfiable? Yes, by $x_1 = 1, x_2 = 0$.
3. Is $C_2 \wedge C_3$ satisfiable? Yes, by $x_1 = 0, x_2 = 1$.
4. Is $C_1 \wedge C_2 \wedge C_3$ satisfiable?

Examples of 3-SAT

Example:

- ▶ $C_1 = x_1 \vee 0 \vee 0$
- ▶ $C_2 = x_2 \vee 0 \vee 0$
- ▶ $C_3 = \overline{x_1} \vee \overline{x_2} \vee 0$

1. Is $C_1 \wedge C_2$ satisfiable? Yes, by $x_1 = 1, x_2 = 1$.
2. Is $C_1 \wedge C_3$ satisfiable? Yes, by $x_1 = 1, x_2 = 0$.
3. Is $C_2 \wedge C_3$ satisfiable? Yes, by $x_1 = 0, x_2 = 1$.
4. Is $C_1 \wedge C_2 \wedge C_3$ satisfiable? No.

3-SAT and Independent Set

$$C_1 = x_1 \vee \overline{x_2} \vee \overline{x_3}$$

$$C_2 = \overline{x_1} \vee x_2 \vee x_4$$

$$C_3 = \overline{x_1} \vee x_3 \vee \overline{x_4}$$

- We want to prove $3\text{-SAT} \leq_P \text{INDEPENDENT SET}$.

3-SAT and Independent Set

$$C_1 = \textcolor{red}{x}_1 \vee \overline{x}_2 \vee \overline{x}_3$$

$$C_2 = \overline{x}_1 \vee \textcolor{red}{x}_2 \vee \textcolor{red}{x}_4$$

$$C_3 = \overline{x}_1 \vee \textcolor{red}{x}_3 \vee \overline{x}_4$$

1. Select $x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1$.

- ▶ We want to prove $3\text{-SAT} \leq_P \text{INDEPENDENT SET}$.
- ▶ 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.

3-SAT and Independent Set

$$C_1 = x_1 \vee \overline{x_2} \vee \overline{x_3}$$

$$C_2 = \overline{x_1} \vee x_2 \vee x_4$$

$$C_3 = \overline{x_1} \vee x_3 \vee \overline{x_4}$$

1. Select $x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1$.

2. Choose one literal from each clause to evaluate to true.

► We want to prove $3\text{-SAT} \leq_P \text{INDEPENDENT SET}$.

► 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*, e.g., select $\overline{x_2}$ in C_1 and x_2 in C_2 .

3-SAT and Independent Set

$$C_1 = x_1 \vee \overline{x_2} \vee \overline{x_3}$$

$$C_2 = \overline{x_1} \vee x_2 \vee x_4$$

$$C_3 = \overline{x_1} \vee x_3 \vee \overline{x_4}$$

1. Select $x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1$.

2. Choose one literal from each clause to evaluate to true.

► Choices of selected literals imply $x_1 = 0, x_2 = 0, x_4 = 1$.

► We want to prove $3\text{-SAT} \leq_P \text{INDEPENDENT SET}$.

► 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*, e.g., select $\overline{x_2}$ in C_1 and x_2 in C_2 .

Proving $3\text{-SAT} \leq_P \text{Independent Set}$

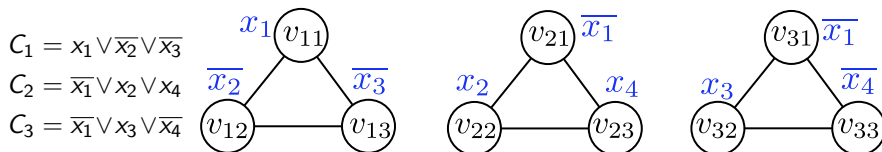
$$C_1 = x_1 \vee \overline{x_2} \vee \overline{x_3}$$

$$C_2 = \overline{x_1} \vee x_2 \vee x_4$$

$$C_3 = \overline{x_1} \vee x_3 \vee \overline{x_4}$$

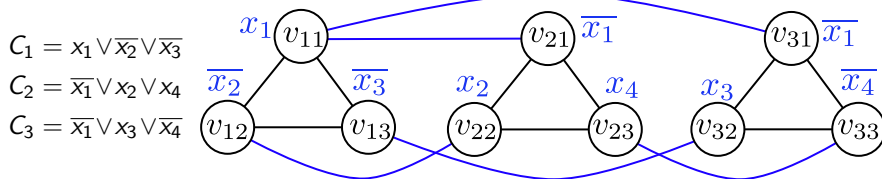
- ▶ We are given an instance of 3-SAT with k clauses of length three over n variables.
- ▶ Construct an instance of independent set: graph $G(V, E)$ with $3k$ nodes.

Proving $3\text{-SAT} \leq_P \text{Independent Set}$



- ▶ We are given an instance of 3-SAT with k clauses of length three over n variables.
- ▶ Construct an instance of independent set: 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 .

Proving $3\text{-SAT} \leq_P \text{Independent Set}$



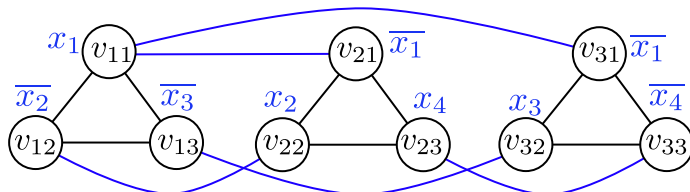
- ▶ We are given an instance of 3-SAT with k clauses of length three over n variables.
- ▶ Construct an instance of independent set: graph $G(V, E)$ with $3k$ nodes.
 - ▶ For each clause $C_i, 1 \leq i \leq k$, add a triangle of three nodes v_{i1}, v_{i2}, v_{i3} and three edges to G .
 - ▶ Label each 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 $3\text{-SAT} \leq_P \text{Independent Set}$

$$C_1 = x_1 \vee \overline{x_2} \vee \overline{x_3}$$

$$C_2 = \overline{x_1} \vee x_2 \vee x_4$$

$$C_3 = \overline{x_1} \vee x_3 \vee \overline{x_4}$$



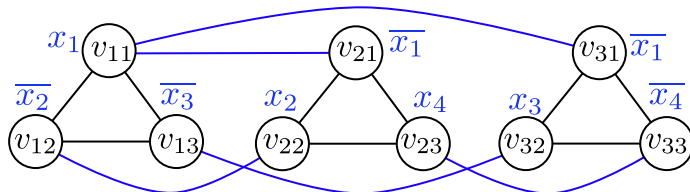
- Claim: 3-SAT instance is satisfiable iff G has an independent set of size k .

Proving $3\text{-SAT} \leq_P \text{Independent Set}$

$$C_1 = x_1 \vee \overline{x_2} \vee \overline{x_3}$$

$$C_2 = \overline{x_1} \vee x_2 \vee x_4$$

$$C_3 = \overline{x_1} \vee x_3 \vee \overline{x_4}$$



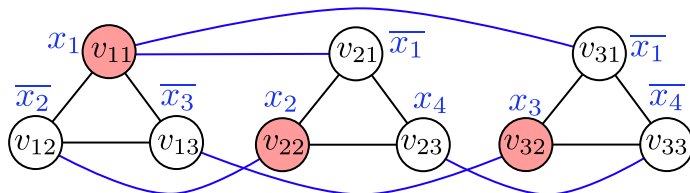
- Claim: 3-SAT instance is satisfiable iff G has an independent set of size k .
- Satisfiable assignment \rightarrow independent set of size k :

Proving $3\text{-SAT} \leq_P \text{Independent Set}$

$$C_1 = x_1 \vee \overline{x_2} \vee \overline{x_3}$$

$$C_2 = \overline{x_1} \vee x_2 \vee x_4$$

$$C_3 = \overline{x_1} \vee x_3 \vee \overline{x_4}$$



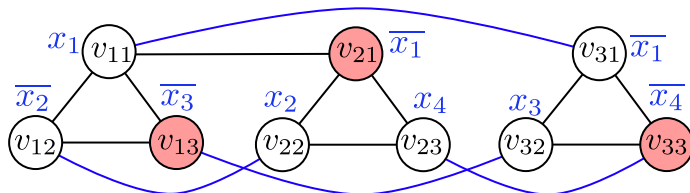
- Claim: 3-SAT instance is satisfiable iff G has an independent set of size k .
- Satisfiable assignment \rightarrow independent set of size k : Each triangle in G has at least one node whose label evaluates to 1. Set S of nodes consisting of one such node from each triangle forms an independent set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

Proving $3\text{-SAT} \leq_P \text{Independent Set}$

$$C_1 = x_1 \vee \overline{x_2} \vee \overline{x_3}$$

$$C_2 = \overline{x_1} \vee x_2 \vee x_4$$

$$C_3 = \overline{x_1} \vee x_3 \vee \overline{x_4}$$



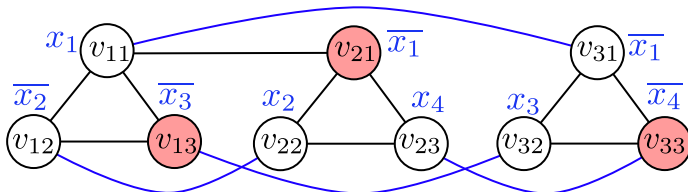
- Claim: 3-SAT instance is satisfiable iff G has an independent set of size k .
- Satisfiable assignment \rightarrow independent set of size k : Each triangle in G has at least one node whose label evaluates to 1. Set S of nodes consisting of one such node from each triangle forms an independent set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

Proving $3\text{-SAT} \leq_P \text{Independent Set}$

$$C_1 = x_1 \vee \overline{x_2} \vee \overline{x_3}$$

$$C_2 = \overline{x_1} \vee x_2 \vee x_4$$

$$C_3 = \overline{x_1} \vee x_3 \vee \overline{x_4}$$



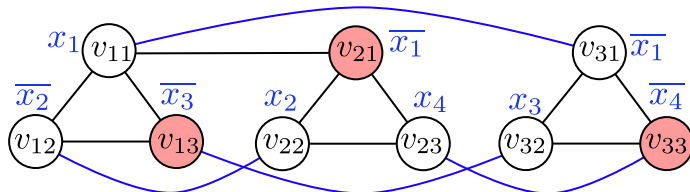
- Claim: 3-SAT instance is satisfiable iff G has an independent set of size k .
- Satisfiable assignment \rightarrow independent set of size k : Each triangle in G has at least one node whose label evaluates to 1. Set S of nodes consisting of one such node from each triangle forms an independent set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

Proving $3\text{-SAT} \leq_P \text{Independent Set}$

$$C_1 = x_1 \vee \overline{x_2} \vee \overline{x_3}$$

$$C_2 = \overline{x_1} \vee x_2 \vee x_4$$

$$C_3 = \overline{x_1} \vee x_3 \vee \overline{x_4}$$



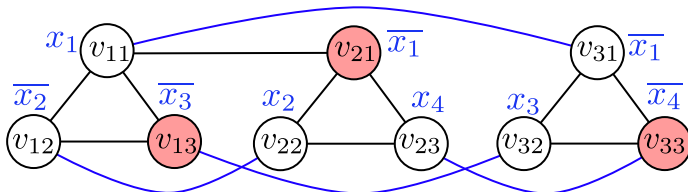
- Claim: 3-SAT instance is satisfiable iff G has an independent set of size k .
- Satisfiable assignment \rightarrow independent set of size k : Each triangle in G has at least one node whose label evaluates to 1. Set S of nodes consisting of one such node from each triangle forms an independent set of size $\geq k$. Why?
- Independent set S of size $k \rightarrow$ satisfiable assignment: the size of this set is k . How do we construct a satisfying truth assignment from the nodes in the independent set?
 - For each variable x_i , only x_i or $\overline{x_i}$ is the label of a node in S . Why?

Proving $3\text{-SAT} \leq_P \text{Independent Set}$

$$C_1 = x_1 \vee \overline{x_2} \vee \overline{x_3}$$

$$C_2 = \overline{x_1} \vee x_2 \vee x_4$$

$$C_3 = \overline{x_1} \vee x_3 \vee \overline{x_4}$$



- ▶ Claim: 3-SAT instance is satisfiable iff G has an independent set of size k .
- ▶ Satisfiable assignment \rightarrow independent set of size k : Each triangle in G has at least one node whose label evaluates to 1. Set S of nodes consisting of one such node from each triangle forms an independent set of size $\geq k$. Why?
- ▶ Independent set S of size $k \rightarrow$ satisfiable assignment: the size of this set is k . How do we construct a satisfying truth assignment from the nodes in the independent set?
 - ▶ For each variable x_i , only x_i or $\overline{x_i}$ is the label of a node in S . Why?
 - ▶ If x_i is the label of a node in S , set $x_i = 1$; else set $x_i = 0$.
 - ▶ Why is each clause satisfied?

Transitivity of Reductions

- ▶ Claim: If $Z \leq_P Y$ and $Y \leq_P X$, then $Z \leq_P X$.

Transitivity of Reductions

► Claim: If $Z \leq_P Y$ and $Y \leq_P X$, then $Z \leq_P X$.

► We have shown

$$3\text{-SAT} \leq_P \text{INDEPENDENT SET} \leq_P \text{VERTEX COVER} \leq_P \text{SET COVER}$$

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).
- ▶ Since 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 and Algorithms

PRIMES

INSTANCE: A natural number n

QUESTION: Is n prime?

- ▶ Decision problem X : for every input s , answer $X(s)$ is yes or no.

Problems and Algorithms

PRIMES

INSTANCE: A natural number n

QUESTION: Is n prime?

- ▶ Decision problem X : for every input s , answer $X(s)$ is yes or no.
- ▶ An algorithm A for a decision problem receives an input s and returns $A(s) \in \{\text{yes}, \text{no}\}$.
- ▶ An algorithm A *solves* the problem X if for every input s ,
 - ▶ if $X(s) = \text{yes}$ then $A(s) = \text{yes}$ and
 - ▶ if $X(s) = \text{no}$ then $A(s) = \text{no}$

Problems and Algorithms

PRIMES

INSTANCE: A natural number n

QUESTION: Is n prime?

- ▶ Decision problem X : for every input s , answer $X(s)$ is yes or no.
- ▶ An algorithm A for a decision problem receives an input s and returns $A(s) \in \{\text{yes}, \text{no}\}$.
- ▶ An algorithm A *solves* the problem X if for every input s ,
 - ▶ if $X(s) = \text{yes}$ then $A(s) = \text{yes}$ and
 - ▶ if $X(s) = \text{no}$ then $A(s) = \text{no}$
- ▶ A has a *polynomial running time* if there is a polynomial function $p(\cdot)$ such that for every input s , A terminates on s in at most $O(p(|s|))$ steps.
 - ▶ There is an algorithm such that $p(|s|) = |s|^8$ for PRIMES (Agarwal, Kayal, Saxena, 2002).

Problems and Algorithms

PRIMES

INSTANCE: A natural number n

QUESTION: Is n prime?

- ▶ Decision problem X : for every input s , answer $X(s)$ is yes or no.
- ▶ An algorithm A for a decision problem receives an input s and returns $A(s) \in \{\text{yes}, \text{no}\}$.
- ▶ An algorithm A *solves* the problem X if for every input s ,
 - ▶ if $X(s) = \text{yes}$ then $A(s) = \text{yes}$ and
 - ▶ if $X(s) = \text{no}$ then $A(s) = \text{no}$
- ▶ A has a *polynomial running time* if there is a polynomial function $p(\cdot)$ such that for every input s , A terminates on s in at most $O(p(|s|))$ steps.
 - ▶ There is an algorithm such that $p(|s|) = |s|^8$ for PRIMES (Agarwal, Kayal, Saxena, 2002).
- ▶ \mathcal{P} : set of problems X for which there is a polynomial time algorithm.

Problems and Algorithms

PRIMES

INSTANCE: A natural number n

QUESTION: Is n prime?

- ▶ Decision problem X : for every input s , answer $X(s)$ is yes or no.
- ▶ An algorithm A for a decision problem receives an input s and returns $A(s) \in \{\text{yes}, \text{no}\}$.
- ▶ An algorithm A *solves* the problem X if for every input s ,
 - ▶ if $X(s) = \text{yes}$ then $A(s) = \text{yes}$ and
 - ▶ if $X(s) = \text{no}$ then $A(s) = \text{no}$
- ▶ A has a *polynomial running time* if there is a polynomial function $p(\cdot)$ such that for every input s , A terminates on s in at most $O(p(|s|))$ steps.
 - ▶ There is an algorithm such that $p(|s|) = |s|^8$ for PRIMES (Agarwal, Kayal, Saxena, 2002).
- ▶ \mathcal{P} : set of problems X for which there is a polynomial time algorithm.
- ▶ A decision problem X is in \mathcal{P} iff there is an algorithm A with polynomial running time that solves 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 $X(s) = \text{yes}$.
- ▶ Checker for INDEPENDENT SET:

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 $X(s) = \text{yes}$.
- ▶ Checker for INDEPENDENT SET: t is a set of at least k vertices; checker verifies that no pair of these vertices are connected by an edge.

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 $X(s) = \text{yes}$.
- ▶ Checker for INDEPENDENT SET: t is a set of at least k vertices; checker verifies that no pair of these vertices are connected by an edge.
- ▶ 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. for all inputs s
 - ▶ if $X(s) = \text{yes}$, then there is a proof t such that $B(s, t) = \text{yes}$ and
 - ▶ if $X(s) = \text{no}$, then for all proofs t , $B(s, t) = \text{no}$.

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 $X(s) = \text{yes}$.
- ▶ Checker for INDEPENDENT SET: t is a set of at least k vertices; checker verifies that no pair of these vertices are connected by an edge.
- ▶ 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. for all inputs s
 - ▶ if $X(s) = \text{yes}$, then there is a proof t such that $B(s, t) = \text{yes}$ and
 - ▶ if $X(s) = \text{no}$, then for all proofs t , $B(s, t) = \text{no}$.
- ▶ Certifier’s job is to take a candidate proof (t) that $s \in X$ and check in polynomial time whether t is a correct proof.
- ▶ Proof t must be “short” so that certifier can run in polynomial time.

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 $X(s) = \text{yes}$.
- ▶ Checker for INDEPENDENT SET: t is a set of at least k vertices; checker verifies that no pair of these vertices are connected by an edge.
- ▶ 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. for all inputs s
 - ▶ if $X(s) = \text{yes}$, then there is a proof t such that $B(s, t) = \text{yes}$ and
 - ▶ if $X(s) = \text{no}$, then for all proofs t , $B(s, t) = \text{no}$.
- ▶ Certifier’s job is to take a candidate proof (t) that $s \in X$ and check in polynomial time whether t is a correct proof.
- ▶ Proof t must be “short” so that certifier can run in polynomial time.
- ▶ Certifier does not care about how to find these proofs.

\mathcal{NP}

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

\mathcal{NP}

- ▶ \mathcal{NP} is the set of all problems for which there exists an efficient certifier.
- ▶ $3\text{-SAT} \in \mathcal{NP}$:
 - ▶ Proof t : .
 - ▶ Certifier B : .

\mathcal{NP}

- ▶ \mathcal{NP} is the set of all problems for which there exists an efficient certifier.
- ▶ $3\text{-SAT} \in \mathcal{NP}$:
 - ▶ Proof t : a truth assignment to the variables.
 - ▶ Certifier B :

\mathcal{NP}

- ▶ \mathcal{NP} is the set of all problems for which there exists an efficient certifier.
- ▶ 3-SAT $\in \mathcal{NP}$:
 - ▶ Proof t : a truth assignment to the variables.
 - ▶ Certifier B : checks whether assignment causes each clause to evaluate to true.

\mathcal{NP}

- ▶ \mathcal{NP} is the set of all problems for which there exists an efficient certifier.
- ▶ $3\text{-SAT} \in \mathcal{NP}$:
 - ▶ Proof t : a truth assignment to the variables.
 - ▶ Certifier B : checks whether assignment causes each clause to evaluate to true.
- ▶ $\text{INDEPENDENT SET} \in \mathcal{NP}$:
 - ▶ Proof t : .
 - ▶ Certifier B : .

\mathcal{NP}

- ▶ \mathcal{NP} is the set of all problems for which there exists an efficient certifier.
- ▶ $3\text{-SAT} \in \mathcal{NP}$:
 - ▶ Proof t : a truth assignment to the variables.
 - ▶ Certifier B : checks whether assignment causes each clause to evaluate to true.
- ▶ $\text{INDEPENDENT SET} \in \mathcal{NP}$:
 - ▶ Proof t : a set of at least k vertices.
 - ▶ Certifier B :

\mathcal{NP}

- ▶ \mathcal{NP} is the set of all problems for which there exists an efficient certifier.
- ▶ $3\text{-SAT} \in \mathcal{NP}$:
 - ▶ Proof t : a truth assignment to the variables.
 - ▶ Certifier B : checks whether assignment causes each clause to evaluate to true.
- ▶ $\text{INDEPENDENT SET} \in \mathcal{NP}$:
 - ▶ Proof t : a set of at least k vertices.
 - ▶ Certifier B : checks that no pair of these vertices are connected by an edge.

\mathcal{NP}

- ▶ \mathcal{NP} is the set of all problems for which there exists an efficient certifier.
- ▶ $3\text{-SAT} \in \mathcal{NP}$:
 - ▶ Proof t : a truth assignment to the variables.
 - ▶ Certifier B : checks whether assignment causes each clause to evaluate to true.
- ▶ $\text{INDEPENDENT SET} \in \mathcal{NP}$:
 - ▶ Proof t : a set of at least k vertices.
 - ▶ Certifier B : checks that no pair of these vertices are connected by an edge.
- ▶ $\text{SET COVER} \in \mathcal{NP}$:
 - ▶ Proof t :
 - ▶ Certifier B :

\mathcal{NP}

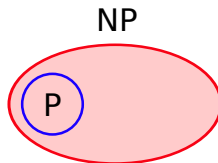
- ▶ \mathcal{NP} is the set of all problems for which there exists an efficient certifier.
- ▶ $3\text{-SAT} \in \mathcal{NP}$:
 - ▶ Proof t : a truth assignment to the variables.
 - ▶ Certifier B : checks whether assignment causes each clause to evaluate to true.
- ▶ $\text{INDEPENDENT SET} \in \mathcal{NP}$:
 - ▶ Proof t : a set of at least k vertices.
 - ▶ Certifier B : checks that no pair of these vertices are connected by an edge.
- ▶ $\text{SET COVER} \in \mathcal{NP}$:
 - ▶ Proof t : a list of k sets from the collection.
 - ▶ Certifier B : .

\mathcal{NP}

- ▶ \mathcal{NP} is the set of all problems for which there exists an efficient certifier.
- ▶ $3\text{-SAT} \in \mathcal{NP}$:
 - ▶ Proof t : a truth assignment to the variables.
 - ▶ Certifier B : checks whether assignment causes each clause to evaluate to true.
- ▶ $\text{INDEPENDENT SET} \in \mathcal{NP}$:
 - ▶ Proof t : a set of at least k vertices.
 - ▶ Certifier B : checks that no pair of these vertices are connected by an edge.
- ▶ $\text{SET COVER} \in \mathcal{NP}$:
 - ▶ Proof t : a list of k sets from the collection.
 - ▶ Certifier B : checks if their union of these sets is U .

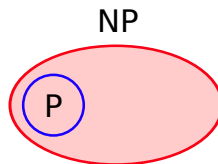
\mathcal{P} vs. \mathcal{NP}

- Claim: $\mathcal{P} \subseteq \mathcal{NP}$.



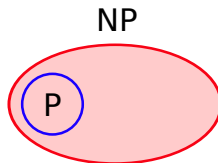
\mathcal{P} vs. \mathcal{NP}

- ▶ Claim: $\mathcal{P} \subseteq \mathcal{NP}$.
 - ▶ If $X \in \mathcal{P}$, 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}

- ▶ Claim: $\mathcal{P} \subseteq \mathcal{NP}$.
 - ▶ If $X \in \mathcal{P}$, 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}

- ▶ Claim: $\mathcal{P} \subseteq \mathcal{NP}$.
 - ▶ If $X \in \mathcal{P}$, 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.



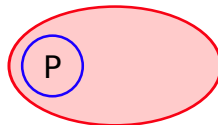
\mathcal{P} vs. \mathcal{NP}

- ▶ Claim: $\mathcal{P} \subseteq \mathcal{NP}$.
 - ▶ If $X \in \mathcal{P}$, 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. \$1M prize offered by Clay Mathematics Institute.



Summary

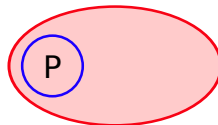
NP



- ▶ $\mathcal{P} \in \mathcal{NP}$
- ▶ 3-SAT, VERTEXCOVER, SETCOVER, INDEPENDENTSET are in \mathcal{NP} .
- ▶ $3\text{-SAT} \leq_P \text{INDEPENDENT SET} \leq_P \text{VERTEX COVER} \leq_P \text{SET COVER}$

Summary

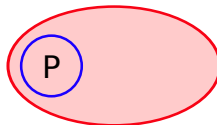
NP



- ▶ $\mathcal{P} \in \mathcal{NP}$
- ▶ 3-SAT, VERTEXCOVER, SETCOVER, INDEPENDENTSET are in \mathcal{NP} .
- ▶ $3\text{-SAT} \leq_P \text{INDEPENDENT SET} \leq_P \text{VERTEX COVER} \leq_P \text{SET COVER}$
- ▶ What is the structure of the problems in \mathcal{NP} ?

Summary

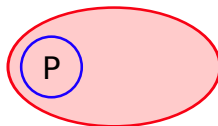
NP



- ▶ $\mathcal{P} \in \mathcal{NP}$
- ▶ 3-SAT, VERTEXCOVER, SETCOVER, INDEPENDENTSET are in \mathcal{NP} .
- ▶ $3\text{-SAT} \leq_P \text{INDEPENDENT SET} \leq_P \text{VERTEX COVER} \leq_P \text{SET COVER}$
- ▶ What is the structure of the problems in \mathcal{NP} ?
 1. Is there a sequence of problems X_1, X_2, X_3, \dots in \mathcal{NP} , such that $X_1 \leq_P X_2 \leq_P X_3 \leq_P \dots$?

Summary

NP



- ▶ $\mathcal{P} \in \mathcal{NP}$
- ▶ 3-SAT, VERTEXCOVER, SETCOVER, INDEPENDENTSET are in \mathcal{NP} .
- ▶ $3\text{-SAT} \leq_P \text{INDEPENDENT SET} \leq_P \text{VERTEX COVER} \leq_P \text{SET COVER}$
- ▶ What is the structure of the problems in \mathcal{NP} ?
 1. Is there a sequence of problems X_1, X_2, X_3, \dots in \mathcal{NP} , such that $X_1 \leq_P X_2 \leq_P X_3 \leq_P \dots$?
 2. Are there 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 $X_2 \leq_P X$?

\mathcal{NP} -Complete and \mathcal{NP} -Hard Problems

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

\mathcal{NP} -Complete and \mathcal{NP} -Hard Problems

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

A problem X is \mathcal{NP} -Complete if

- (i) $X \in \mathcal{NP}$ and
- (ii) for every problem $Y \in \mathcal{NP}$,
 $Y \leq_P X$.

A problem X is \mathcal{NP} -Hard if

- (i) for every problem $Y \in \mathcal{NP}$,
 $Y \leq_P X$.

\mathcal{NP} -Complete and \mathcal{NP} -Hard Problems

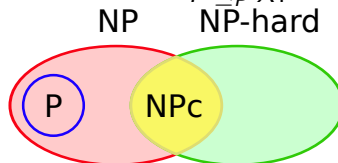
- What are the hardest problems in \mathcal{NP} ?

A problem X is *\mathcal{NP} -Complete* if

- (i) $X \in \mathcal{NP}$ and
- (ii) for **every** problem $Y \in \mathcal{NP}$,
 $Y \leq_P X$.

A problem X is *\mathcal{NP} -Hard* if

- (i) for **every** problem $Y \in \mathcal{NP}$,
 $Y \leq_P X$.



- Claim: Suppose X is \mathcal{NP} -Complete. Then $X \in \mathcal{P}$ iff $\mathcal{P} = \mathcal{NP}$.

\mathcal{NP} -Complete and \mathcal{NP} -Hard Problems

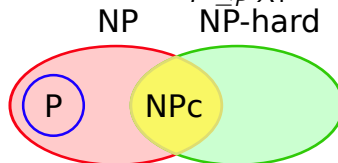
- ▶ What are the hardest problems in \mathcal{NP} ?

A problem X is *\mathcal{NP} -Complete* if

- (i) $X \in \mathcal{NP}$ and
- (ii) for **every** problem $Y \in \mathcal{NP}$,
 $Y \leq_P X$.

A problem X is *\mathcal{NP} -Hard* if

- (i) for **every** problem $Y \in \mathcal{NP}$,
 $Y \leq_P X$.



- ▶ Claim: Suppose X is \mathcal{NP} -Complete. Then $X \in \mathcal{P}$ 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 and \mathcal{NP} -Hard Problems

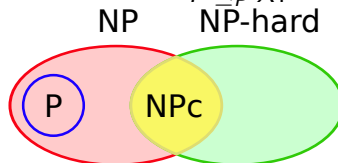
- ▶ What are the hardest problems in \mathcal{NP} ?

A problem X is \mathcal{NP} -Complete if

- (i) $X \in \mathcal{NP}$ and
- (ii) for **every** problem $Y \in \mathcal{NP}$,
 $Y \leq_P X$.

A problem X is \mathcal{NP} -Hard if

- (i) for **every** problem $Y \in \mathcal{NP}$,
 $Y \leq_P X$.



- ▶ Claim: Suppose X is \mathcal{NP} -Complete. Then $X \in \mathcal{P}$ 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.
- ▶ **Does even one \mathcal{NP} -Complete problem exist?!** If it does, how can we prove that every problem in \mathcal{NP} reduces to this problem?

Circuit Satisfiability

- ▶ **Cook-Levin Theorem:** CIRCUIT SATISFIABILITY is \mathcal{NP} -Complete.

Circuit Satisfiability

- ▶ **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 .

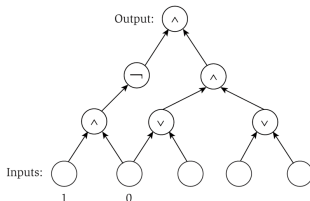


Figure 8.4 A circuit with three inputs, two additional sources that have assigned truth values, and one output.

Circuit Satisfiability

- ▶ **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 .

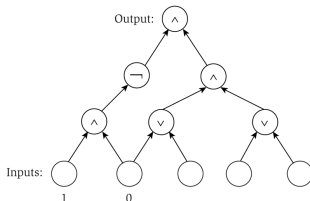


Figure 8.4 A circuit with three inputs, two additional sources that have assigned truth values, and one output.

CIRCUIT SATISFIABILITY

INSTANCE: A circuit K .

QUESTION: Is there a truth assignment to the inputs that causes the output to have value 1?

▶ Skip proof; read textbook or Chapter 2.6 of Garey and Johnson.

Proving Circuit Satisfiability is \mathcal{NP} -Complete

Proving Circuit Satisfiability is \mathcal{NP} -Complete

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

Proving Circuit Satisfiability is \mathcal{NP} -Complete

- ▶ Take an arbitrary problem $X \in \mathcal{NP}$ and show that $X \leq_P \text{CIRCUIT SATISFIABILITY}$.
- ▶ 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 $X \leq_P \text{CIRCUIT SATISFIABILITY}$.
- ▶ 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 $X \leq_P \text{CIRCUIT SATISFIABILITY}$, given an input s of length n , we want to determine whether $s \in X$ using a black box that solves $\text{CIRCUIT SATISFIABILITY}$.

Proving Circuit Satisfiability is \mathcal{NP} -Complete

- ▶ Take an arbitrary problem $X \in \mathcal{NP}$ and show that $X \leq_P \text{CIRCUIT SATISFIABILITY}$.
- ▶ 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 $X \leq_P \text{CIRCUIT SATISFIABILITY}$, given an input s of length n , we want to determine whether $s \in X$ using a black box that solves $\text{CIRCUIT SATISFIABILITY}$.
- ▶ What do we know about X ?

Proving Circuit Satisfiability is \mathcal{NP} -Complete

- ▶ Take an arbitrary problem $X \in \mathcal{NP}$ and show that $X \leq_P \text{CIRCUIT SATISFIABILITY}$.
- ▶ 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 $X \leq_P \text{CIRCUIT SATISFIABILITY}$, given an input s of length n , we want to determine whether $s \in X$ using a black box that solves $\text{CIRCUIT SATISFIABILITY}$.
- ▶ What do we know about X ? It has an efficient certifier $B(\cdot, \cdot)$.

Proving Circuit Satisfiability is \mathcal{NP} -Complete

- ▶ Take an arbitrary problem $X \in \mathcal{NP}$ and show that $X \leq_P \text{CIRCUIT SATISFIABILITY}$.
- ▶ 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 $X \leq_P \text{CIRCUIT SATISFIABILITY}$, given an input s of length n , we want to determine whether $s \in X$ using a black box that solves $\text{CIRCUIT SATISFIABILITY}$.
- ▶ 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 $B(s, t) = \text{yes?}$ ”

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 $B(s, t) = \text{yes?}$ ”

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 .

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 .
- ▶ $s \in X$ iff there is an assignment of the input bits of K that makes K satisfiable.

Example of Transformation to Circuit Satisfiability

- 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).

Example of Transformation to Circuit Satisfiability

- ▶ Suppose G contains three nodes u , v , and w with v connected to u and w .

Example of Transformation to Circuit Satisfiability

- Suppose G contains three nodes u , v , and w with v connected to u and w .

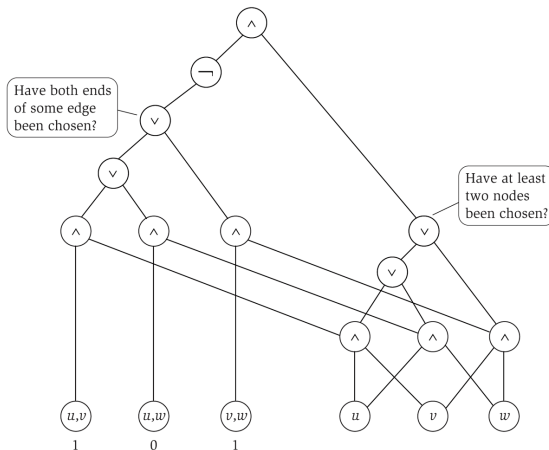


Figure 8.5 A circuit to verify whether a 3-node graph contains a 2-node independent set.

Asymmetry of Certification

- ▶ Definition of efficient certification and \mathcal{NP} is fundamentally asymmetric:
 - ▶ An input string s is a “yes” instance iff there exists a short string t such that $B(s, t) = \text{yes}$.
 - ▶ An input string s is a “no” instance iff *for all* short strings t , $B(s, t) = \text{no}$.

Asymmetry of Certification

- ▶ Definition of efficient certification and \mathcal{NP} is fundamentally asymmetric:
 - ▶ An input string s is a “yes” instance iff there exists a short string t such that $B(s, t) = \text{yes}$.
 - ▶ An input string s is a “no” instance iff *for all* short strings t , $B(s, t) = \text{no}$.
The definition of \mathcal{NP} does not guarantee a short proof for “no” instances.

$\text{co-}\mathcal{NP}$

- For a decision problem X , its *complementary problem* \overline{X} is the set of strings s such that $s \in \overline{X}$ iff $s \notin X$.

$\text{co-}\mathcal{NP}$

- ▶ For a decision problem X , its *complementary problem* \overline{X} is the set of strings s such that $s \in \overline{X}$ iff $s \notin X$.
- ▶ If $X \in \mathcal{P}$,

$\text{co-}\mathcal{NP}$

- ▶ For a decision problem X , its *complementary problem* \overline{X} is the set of strings s such that $s \in \overline{X}$ iff $s \notin X$.
- ▶ If $X \in \mathcal{P}$, then $\overline{X} \in \mathcal{P}$.

$\text{co-}\mathcal{NP}$

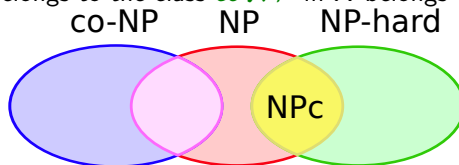
- ▶ For a decision problem X , its *complementary problem* \overline{X} is the set of strings s such that $s \in \overline{X}$ iff $s \notin X$.
- ▶ If $X \in \mathcal{P}$, then $\overline{X} \in \mathcal{P}$.
- ▶ If $X \in \mathcal{NP}$, then is $\overline{X} \in \mathcal{NP}$?

$\text{co-}\mathcal{NP}$

- ▶ For a decision problem X , its *complementary problem* \overline{X} is the set of strings s such that $s \in \overline{X}$ iff $s \notin X$.
- ▶ If $X \in \mathcal{P}$, then $\overline{X} \in \mathcal{P}$.
- ▶ If $X \in \mathcal{NP}$, then is $\overline{X} \in \mathcal{NP}$? Unclear in general.
- ▶ A problem X belongs to the class *co- \mathcal{NP}* iff \overline{X} belongs to \mathcal{NP} .

$\text{co-}\mathcal{NP}$

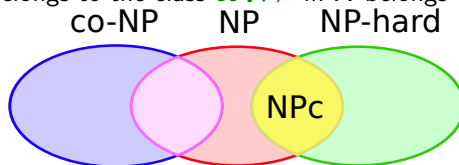
- ▶ For a decision problem X , its *complementary problem* \bar{X} is the set of strings s such that $s \in \bar{X}$ iff $s \notin X$.
- ▶ If $X \in \mathcal{P}$, then $\bar{X} \in \mathcal{P}$.
- ▶ If $X \in \mathcal{NP}$, then is $\bar{X} \in \mathcal{NP}$? Unclear in general.
- ▶ A problem X belongs to the class $\text{co-}\mathcal{NP}$ iff \bar{X} belongs to \mathcal{NP} .



- ▶ Open problem: Is $\mathcal{NP} = \text{co-}\mathcal{NP}$?

$\text{co-}\mathcal{NP}$

- ▶ For a decision problem X , its *complementary problem* \bar{X} is the set of strings s such that $s \in \bar{X}$ iff $s \notin X$.
- ▶ If $X \in \mathcal{P}$, then $\bar{X} \in \mathcal{P}$.
- ▶ If $X \in \mathcal{NP}$, then is $\bar{X} \in \mathcal{NP}$? Unclear in general.
- ▶ A problem X belongs to the class $\text{co-}\mathcal{NP}$ iff \bar{X} belongs to \mathcal{NP} .



- ▶ Open problem: Is $\mathcal{NP} = \text{co-}\mathcal{NP}$?
- ▶ Claim: If $\mathcal{NP} \neq \text{co-}\mathcal{NP}$ then $\mathcal{P} \neq \mathcal{NP}$.

Good Characterisations: the Class $\mathcal{NP} \cap \text{co-}\mathcal{NP}$

- ▶ If a problem belongs to both \mathcal{NP} and $\text{co-}\mathcal{NP}$, then
 - ▶ When the answer is yes, there is a short proof.
 - ▶ When the answer is no, there is a short proof.

Good Characterisations: the Class $\mathcal{NP} \cap \text{co-}\mathcal{NP}$

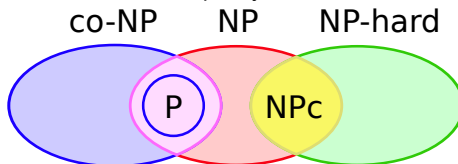
- ▶ If a problem belongs to both \mathcal{NP} and $\text{co-}\mathcal{NP}$, then
 - ▶ When the answer is yes, there is a short proof.
 - ▶ When the answer is no, there is a short proof.
- ▶ Problems in $\mathcal{NP} \cap \text{co-}\mathcal{NP}$ have a *good characterisation*.

Good Characterisations: the Class $\mathcal{NP} \cap \text{co-}\mathcal{NP}$

- ▶ If a problem belongs to both \mathcal{NP} and $\text{co-}\mathcal{NP}$, then
 - ▶ When the answer is yes, there is a short proof.
 - ▶ When the answer is no, there is a short proof.
- ▶ Problems in $\mathcal{NP} \cap \text{co-}\mathcal{NP}$ have a *good characterisation*.
- ▶ Example is the problem of determining if a flow network contains a flow of value at least ν , for some given value of ν .
 - ▶ Yes: construct a flow of value at least ν .
 - ▶ No: demonstrate a cut with capacity less than ν .

Good Characterisations: the Class $\mathcal{NP} \cap \text{co-}\mathcal{NP}$

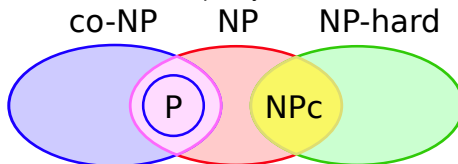
- ▶ If a problem belongs to both \mathcal{NP} and $\text{co-}\mathcal{NP}$, then
 - ▶ When the answer is yes, there is a short proof.
 - ▶ When the answer is no, there is a short proof.
- ▶ Problems in $\mathcal{NP} \cap \text{co-}\mathcal{NP}$ have a *good characterisation*.
- ▶ Example is the problem of determining if a flow network contains a flow of value at least ν , for some given value of ν .
 - ▶ Yes: construct a flow of value at least ν .
 - ▶ No: demonstrate a cut with capacity less than ν .



- ▶ Claim: $\mathcal{P} \subseteq \mathcal{NP} \cap \text{co-}\mathcal{NP}$.

Good Characterisations: the Class $\mathcal{NP} \cap \text{co-}\mathcal{NP}$

- ▶ If a problem belongs to both \mathcal{NP} and $\text{co-}\mathcal{NP}$, then
 - ▶ When the answer is yes, there is a short proof.
 - ▶ When the answer is no, there is a short proof.
- ▶ Problems in $\mathcal{NP} \cap \text{co-}\mathcal{NP}$ have a *good characterisation*.
- ▶ Example is the problem of determining if a flow network contains a flow of value at least ν , for some given value of ν .
 - ▶ Yes: construct a flow of value at least ν .
 - ▶ No: demonstrate a cut with capacity less than ν .



- ▶ Claim: $\mathcal{P} \subseteq \mathcal{NP} \cap \text{co-}\mathcal{NP}$.
- ▶ Open problem: Is $\mathcal{P} = \mathcal{NP} \cap \text{co-}\mathcal{NP}$?