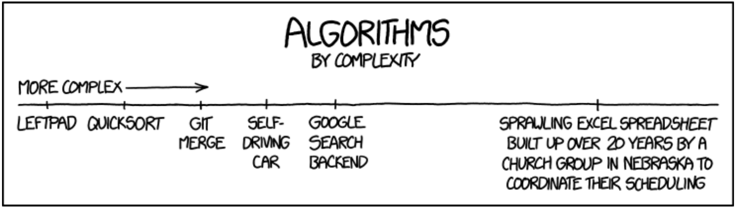
NP and Computational Intractability

T. M. Murali

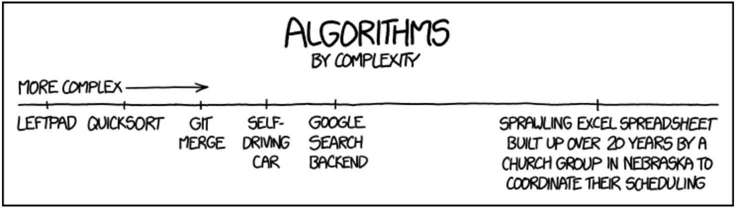
November 16, 18, 30, 2021

Algorithm Design



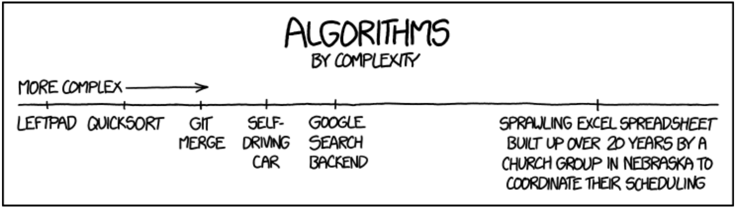
- Patterns
  - Greed. $O(n \log n)$ interval scheduling.
  - Divide-and-conquer. $O(n \log n)$ counting inversions.
  - Dynamic programming. $O(n^3)$ RNA folding.
  - Duality. $O(n^2 m)$ maximum flow and minimum cuts.

Algorithm Design



- Patterns
  - Greed. — $O(n \log n)$ interval scheduling.
  - Divide-and-conquer. — $O(n \log n)$ counting inversions.
  - Dynamic programming. — $O(n^3)$ RNA folding.
  - Duality. — $O(n^2 m)$ maximum flow and minimum cuts.
  - Reductions. — IMAGE SEGMENTATION $\leq_P$ MINIMUM s-t CUT
  - Local search.
  - Randomization.

Algorithm Design



- Patterns
  - Greed. $O(n \log n)$ interval scheduling.
  - Divide-and-conquer. $O(n \log n)$ counting inversions.
  - Dynamic programming. $O(n^3)$ RNA folding.
  - Duality. $O(n^2 m)$ maximum flow and minimum cuts.
  - Reductions. IMAGE SEGMENTATION $\leq_P$ MINIMUM s-t CUT
  - 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!

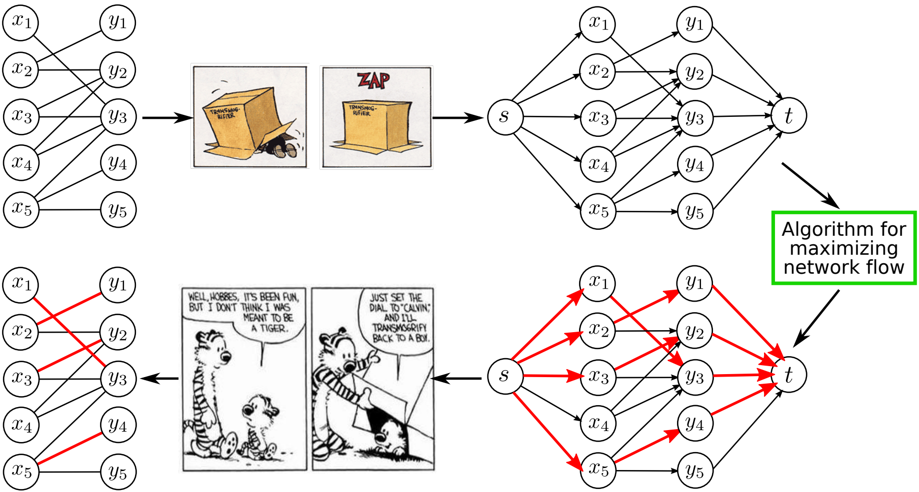
Polynomial-Time Reduction

- 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 input to Y can be solved using a polynomial number of standard operations, plus one call to a black box that solves problem X.

Polynomial-Time Reduction



Maximum Bipartite Matching $\leq_P$ Maximum s-t Flow

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 input to Y can be solved using a polynomial number of standard operations, plus one call 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 input to Y can be solved using a polynomial number of standard operations, plus one call to a black box that solves problem X.
  - MAXIMUM BIPARTITE MATCHING $\leq_P$ MAXIMUM s-t FLOW
  - IMAGE SEGMENTATION $\leq_P$ MINIMUM s-t CUT

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 input to Y can be solved using a polynomial number of standard operations, plus one call to a black box that solves problem X.
  - MAXIMUM BIPARTITE MATCHING $\leq_P$ MAXIMUM s-t FLOW
  - IMAGE SEGMENTATION $\leq_P$ MINIMUM s-t CUT
- $Y \leq_P X$ implies that "X is at least as hard as Y."
  - It is possible to solve Y using (potentially unknown) algorithm that solves X.
  - Not the reverse: we can solve X using an algorithm for Y.
- Such reductions are *Karp reductions*. *Cook reductions* allow a polynomial number of calls 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

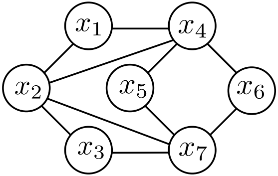
- So far, we have developed algorithms that solve optimisation problems.
  - Compute the *largest* flow.
  - Find the *closest* pair of points.
  - Find the schedule with the *least* completion time.
- Now, we will focus on *decision versions* of problems, e.g., is there a flow with value at least k,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

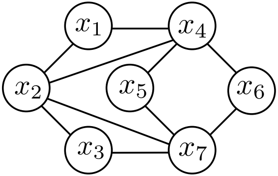
PRIMES

INSTANCE: A natural number n

QUESTION: Is n prime?

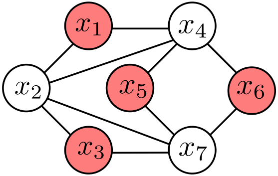
Independent Set and Vertex Cover

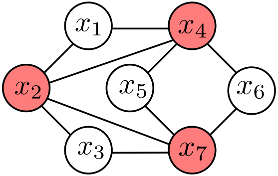




- Given an undirected graph $G(V, E)$, a subset $S \subseteq V$ is an *independent set* if no two vertices in S are connected by an edge.
- Given an undirected graph $G(V, E)$, a subset $S \subseteq V$ is a *vertex cover* if every edge in E is incident on at least one vertex in S. Poll

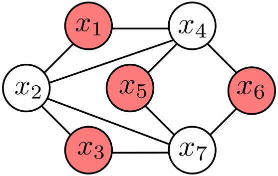
Independent Set and Vertex Cover

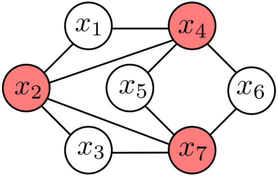




- Given an undirected graph $G(V, E)$, a subset $S \subseteq V$ is an *independent set* if no two vertices in S are connected by an edge.
- Given an undirected graph $G(V, E)$, a subset $S \subseteq V$ is a *vertex cover* if every edge in E is incident on at least one vertex in S.

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

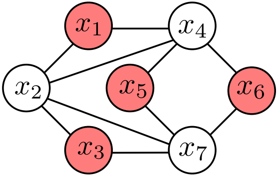
QUESTION: Does G contain an independent set of size $\geq k$?

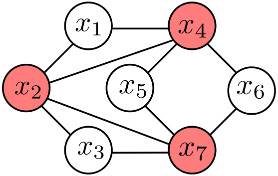
Vertex cover

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 $\geq k$?

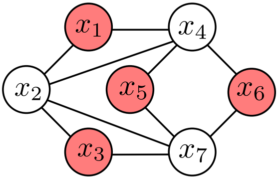
Vertex cover

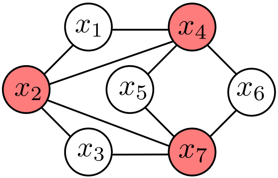
INSTANCE: Undirected graph G and an integer l

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

- 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 $\geq k$?

VERTEX COVER

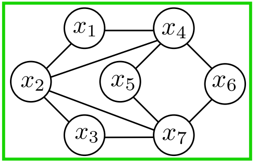
INSTANCE: Undirected graph G and an integer l

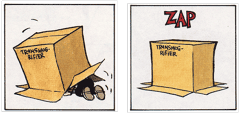
QUESTION: Does G contain a vertex cover of size $\leq l$?

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

Strategy for Proving Indep. Set $\leq_P$ Vertex Cover

$k = 3$





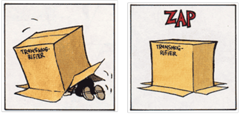
$l = ?$

Input graph for the vertex cover problem

Black box algorithm for solving vertex cover

Yes

No

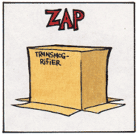


Yes, there is an independent set of size at least 3

No, every independent set is of size 3 or less

Strategy for Proving Indep. Set $\leq_P$ Vertex Cover

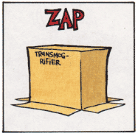
1. Start with an arbitrary input to INDEPENDENT SET: an undirected graph $G(V, E)$ and an integer k.
2. From $G(V, E)$ and k, create an input to VERTEX COVER: an undirected graph $G'(V', E')$ and an integer l.
   - G' related to G in some way.
   - l can depend upon k and size of G.



3. Prove that $G(V, E)$ has an independent set of size $\geq k$ if and only if $G'(V', E')$ has a vertex cover of size $\leq l$.

Strategy for Proving Indep. Set $\leq_P$ Vertex Cover

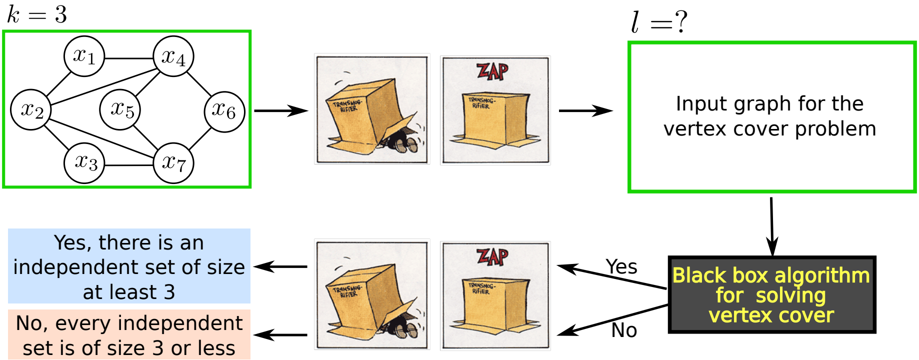
1. Start with an arbitrary input to INDEPENDENT SET: an undirected graph $G(V, E)$ and an integer k.
2. From $G(V, E)$ and k, create an input to VERTEX COVER: an undirected graph $G'(V', E')$ and an integer l.
   - G' related to G in some way.
   - l can depend upon k and size of G.



3. Prove that $G(V, E)$ has an independent set of size $\geq k$ if and only if $G'(V', E')$ has a vertex cover of size $\leq l$.

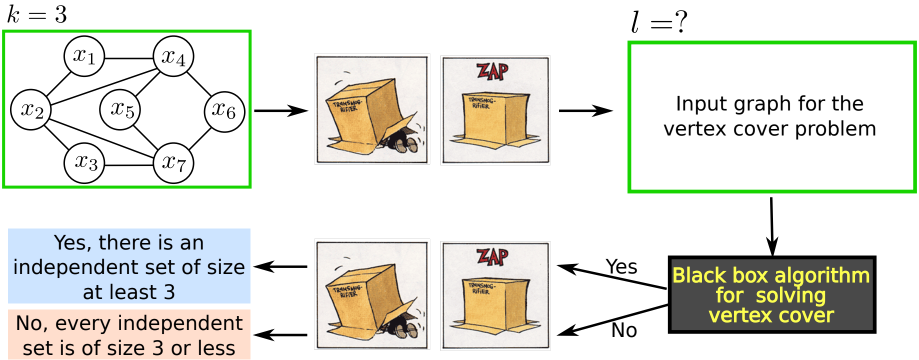
- Transformation and proof must be correct for all possible graphs $G(V, E)$ and all possible values of k.
- Why is the proof an iff statement?

Reason for Two-Way Proof



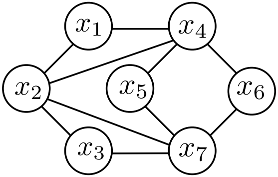
- Why is the proof an iff statement?

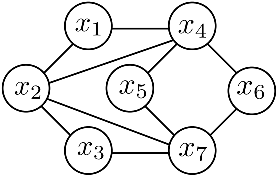
Reason for Two-Way Proof



- Why is the proof an iff statement? In the reduction, we are using black box for Vertex Cover to solve Independent Set.
  1. 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.
  2. 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.

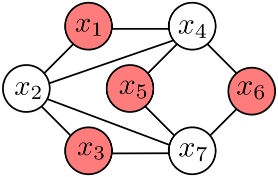
Proof that Independent Set $\leq_P$ Vertex Cover

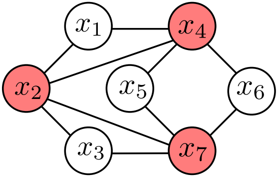




1. Arbitrary input to INDEPENDENT SET: an undirected graph $G(V, E)$ and an integer k.
2. Let $|V| = n$.
3. Create an input to VERTEX COVER: same undirected graph $G(V, E)$ and integer $l = n - k$.

Proof that Independent Set $\leq_P$ Vertex Cover

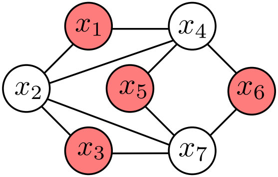


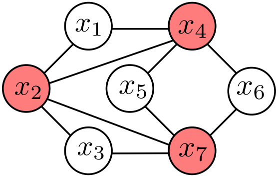


1. Arbitrary input to INDEPENDENT SET: an undirected graph $G(V, E)$ and an integer k.
2. Let $|V| = n$.
3. Create an input to VERTEX COVER: same undirected graph $G(V, E)$ and integer $l = n - k$.
4. Claim: $G(V, E)$ has an independent set of size $\geq k$ iff $G(V, E)$ has a vertex cover of size $\leq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

   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 input to Independent Set: an undirected graph $G(V, E)$ and an integer k.
2. Let $|V| = n$.
3. Create an input to Vertex Cover: same undirected graph $G(V, E)$ and integer $l = n - k$.
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.

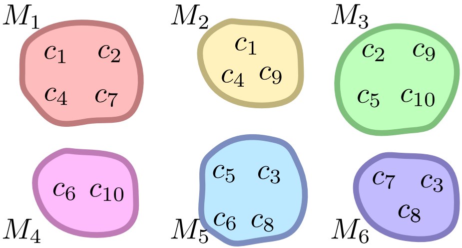
Vertex Cover and Set Cover

- 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.

MICROBE COVER

INSTANCE: A set U of n compounds, a collection $M_1, M_2, \ldots, M_l$ of microbes, where each microbe can make a subset of compounds in U, and an integer k.

QUESTION: Is there a subset of $\leq k$ microbes that can together make all the compounds in U?



$n = 10, l = 6, k = 3$

- Define a "microbe" to be the set of compounds it can make, e.g., $M_1 = \{c_1, c_2, c_4, c_7\}$. Poll

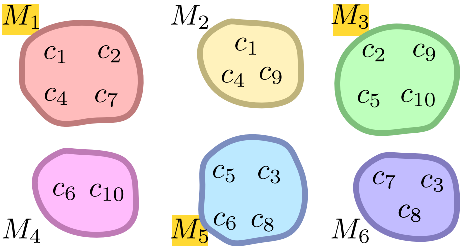
Vertex Cover and Set Cover

- 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.

MICROBE COVER

INSTANCE: A set U of n compounds, a collection $M_1, M_2, \ldots, M_l$ of microbes, where each microbe can make a subset of compounds in U, and an integer k.

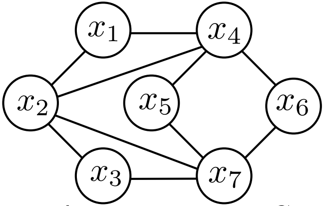
QUESTION: Is there a subset of $\leq k$ microbes that can together make all the compounds in U?



$n = 10, l = 6, k = 3$

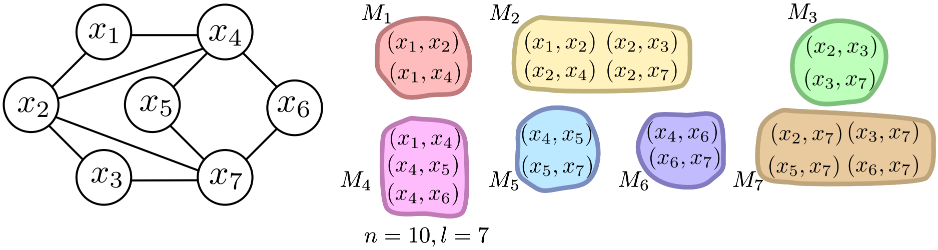
- Define a "microbe" to be the set of compounds it can make, e.g., $M_1 = \{c_1, c_2, c_4, c_7\}$.

Vertex Cover $\leq_P$ Microbe Cover



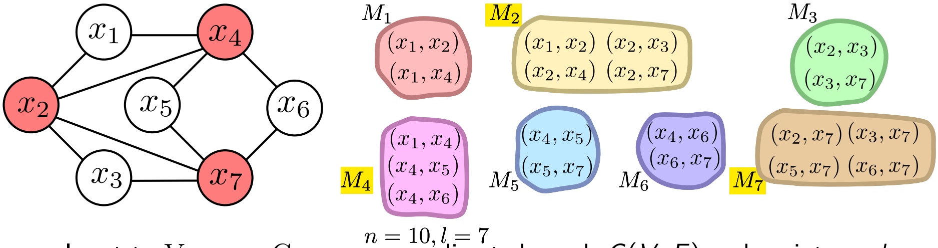
- Input to VERTEX COVER: an undirected graph $G(V, E)$ and an integer k.
- Let $|V| = l$.
- Create an input $\{U, \{M_1, M_2, \ldots M_l\}\}$ to MICROBE COVER where

Vertex Cover $\leq_P$ Microbe Cover



- Input to VERTEX COVER: an undirected graph $G(V, E)$ and an integer k.
- Let $|V| = l$.
- Create an input $\{U, \{M_1, M_2, \ldots M_l\}\}$ to MICROBE COVER where
  - $U = E$, i.e., each element of U is an edge of G, and
  - for each node $i \in V$, create a microbe M_i whose compounds are the set of edges incident on i.

Vertex Cover $\leq_P$ Microbe Cover



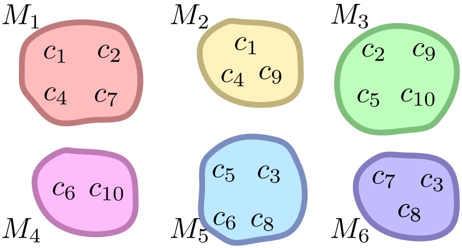
- Input to VERTEX COVER: an undirected graph $G(V, E)$ and an integer k.
- Let $|V| = l$.
- Create an input $\{U, \{M_1, M_2, \ldots M_l\}\}$ to MICROBE COVER where
  - $U = E$, i.e., each element of U is an edge of G, and
  - for each node $i \in V$, create a microbe M_i whose compounds are the set of edges incident on i.
- Claim: U can be covered with $\leq k$ microbes iff G has a vertex cover with at $\leq k$ nodes.
- Proof strategy:
  1. If G has a vertex cover of size $\leq k$, then U can be covered with $\leq k$ microbes.
  2. If U can be covered with $\leq k$ microbes, then G has a vertex cover of size $\leq k$.

Microbe Cover and Set Cover

MICROBE COVER

INSTANCE: A set U of n compounds, a collection $M_1, M_2, \ldots, M_l$ of microbes, where each microbe can make a subset of compounds in U, and an integer k.

QUESTION: Is there a subset of $\leq k$ microbes that can together make all the compounds in U?



$n = 10, l = 6$

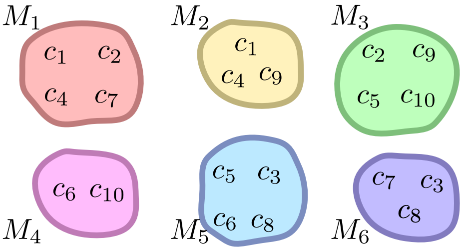
- Purely combinatorial problem: a "microbe" is just a set of "compounds."

Microbe Cover and Set Cover

MICROBE COVER

INSTANCE: A set U of n compounds, a collection $M_1, M_2, \ldots, M_l$ of microbes, where each microbe can make a subset of compounds in U, and an integer k.

QUESTION: Is there a subset of $\leq k$ microbes that can together make all the compounds in U?



$n = 10, l = 6$

- Purely combinatorial problem: a "microbe" is just a set of "compounds."

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?

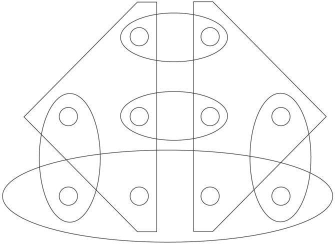


Figure 8.2 An instance of the Set Cover Problem.

Boolean Satisfiability

- Abstract problems formulated in Boolean notation.

Boolean Satisfiability

- Abstract problems formulated in Boolean notation.
- Given a set $X = \{x_1, x_2, \ldots, x_n\}$ of n Boolean variables.
- Each variable can take the value 0 or 1.
- *Term*: a variable x_i or its negation $\overline{x_i}$.
- *Clause* of *length* l: (or) of l distinct terms $t_1 \vee t_2 \vee \cdots t_l$.
- *Truth assignment* for X: is a function $\nu : X \rightarrow \{0, 1\}$.
- An assignment ν *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 $C_1, C_2, \ldots C_k$ if it causes all clauses to evaluate to 1, i.e., $C_1 \wedge C_2 \wedge \cdots C_k = 1$.
  - 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*.

Example

- $X = \{x_1, x_2, x_3, x_4\}$
- Terms: $x_1, \overline{x_1}, x_2, \overline{x_2}, x_3, \overline{x_3}, x_4, \overline{x_4}$

Example

- $X = \{x_1, x_2, x_3, x_4\}$
- Terms: $x_1, \overline{x_1}, x_2, \overline{x_2}, x_3, \overline{x_3}, x_4, \overline{x_4}$
- Clauses: Poll

$$x_1 \vee \overline{x_2} \vee \overline{x_3}$$
$$x_2 \vee \overline{x_3} \vee x_4$$
$$x_3 \vee \overline{x_4}$$

Example

- $X = \{x_1, x_2, x_3, x_4\}$
- Terms: $x_1, \overline{x_1}, x_2, \overline{x_2}, x_3, \overline{x_3}, x_4, \overline{x_4}$
- Clauses:

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

  $x_2 \vee \overline{x_3} \vee x_4$

  $x_3 \vee \overline{x_4}$
- Assignment: $x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 0$

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

  $x_2 \vee \overline{x_3} \vee x_4$

  $x_3 \vee \overline{x_4}$
  - Not a satisfying assignment

Example

- $X = \{x_1, x_2, x_3, x_4\}$
- Terms: $x_1, \overline{x_1}, x_2, \overline{x_2}, x_3, \overline{x_3}, x_4, \overline{x_4}$
- Clauses:

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

  $x_2 \vee \overline{x_3} \vee x_4$

  $x_3 \vee \overline{x_4}$
- Assignment: $x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 0$

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

  $x_2 \vee \overline{x_3} \vee x_4$

  $x_3 \vee \overline{x_4}$
  - Not a satisfying assignment
- Assignment: $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 0$

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

  $x_2 \vee \overline{x_3} \vee x_4$

  $x_3 \vee \overline{x_4}$
  - Is a satisfying assignment

Example

- $X = \{x_1, x_2, x_3, x_4\}$
- Terms: $x_1, \overline{x_1}, x_2, \overline{x_2}, x_3, \overline{x_3}, x_4, \overline{x_4}$
- Clauses:

$$x_1 \vee \overline{x_2} \vee \overline{x_3}$$
$$x_2 \vee \overline{x_3} \vee x_4$$
$$x_3 \vee \overline{x_4}$$

- Assignment: $x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 0$

$$x_1 \vee \overline{x_2} \vee \overline{x_3}$$
$$x_2 \vee \overline{x_3} \vee x_4$$
$$x_3 \vee \overline{x_4}$$

  - Not a satisfying assignment
- Assignment: $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 0$

$$x_1 \vee \overline{x_2} \vee \overline{x_3}$$
$$x_2 \vee \overline{x_3} \vee x_4$$
$$x_3 \vee \overline{x_4}$$

  - Is a satisfying assignment
- Assignment: $x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 1$

$$x_1 \vee \overline{x_2} \vee \overline{x_3}$$
$$x_2 \vee \overline{x_3} \vee x_4$$
$$x_3 \vee \overline{x_4}$$

  - Is not a satisfying assignment

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?

SAT and 3-SAT

3-Satisfiability Problem (SAT)

INSTANCE: A set of clauses $C_1, C_2, \ldots C_k$, each of length three, 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

3-SATISFIABILITY PROBLEM (SAT)

INSTANCE: A set of clauses $C_1, C_2, \ldots C_k$, each of length three, 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.

Examples of 3-SAT

Example:

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

▸ Poll

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 $\text{3-SAT} \leq_P \textsc{Independent Set}$.

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

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

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

$$C_1 = x_1 \vee \overline{x_2} \vee \overline{x_3}$$
$$C_2 = \overline{x_1} \vee x_2 \vee x_4$$
$$C_3 = \overline{x_1} \vee x_3 \vee \overline{x_4}$$

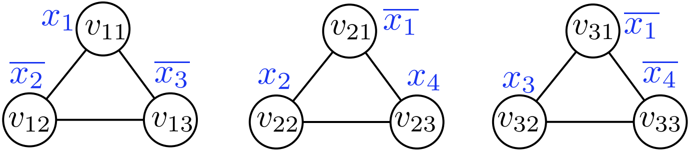
- We are given an input to 3-SAT with k clauses of length three over n variables.
- Construct an input to independent set: graph $G(V, E)$ with $3k$ nodes.

Proving 3-SAT $\leq_P$ Independent Set

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

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

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



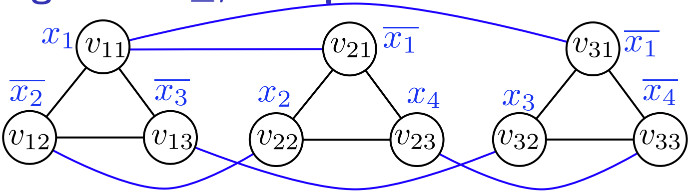
- We are given an input to 3-SAT with k clauses of length three over n variables.
- Construct an input to independent set: graph $G(V, E)$ with $3k$ nodes.
  - For each clause $C_i, 1 \leq i \leq k$, add a triangle of three nodes v_{i1}, v_{i2}, v_{i3} and three edges to G.
  - Label each node $v_{ij}, 1 \leq j \leq 3$ with the jth term in C_i. Poll

Proving 3-SAT $\leq_P$ Independent Set

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

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

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



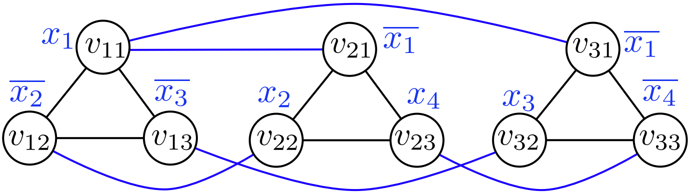
- We are given an input to 3-SAT with k clauses of length three over n variables.
- Construct an input to independent set: graph $G(V, E)$ with $3k$ nodes.
  - For each clause $C_i, 1 \leq i \leq k$, add a triangle of three nodes v_{i1}, v_{i2}, v_{i3} and three edges to G.
  - Label each node $v_{ij},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

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

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

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



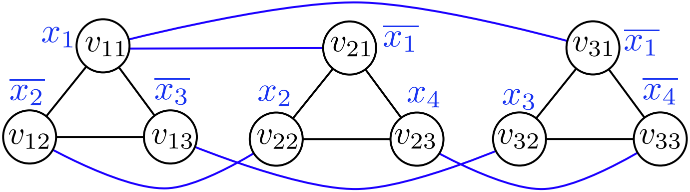
- Claim: Input to 3-SAT is satisfiable iff G has an independent set of size k.

Proving 3-SAT $\leq_P$ Independent Set

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

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

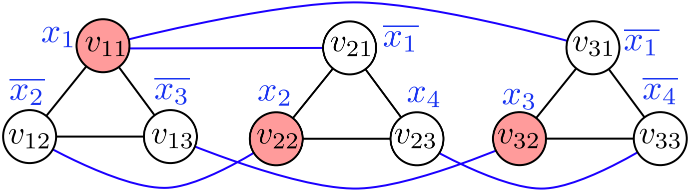
$C_3 = \overline{x_1} \vee x_3 \vee \overline{x_4}$



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

Proving 3-SAT $\leq_P$ Independent Set

$C_1 = x_1 \vee \overline{x_2} \vee \overline{x_3}$
$C_2 = \overline{x_1} \vee x_2 \vee x_4$
$C_3 = \overline{x_1} \vee x_3 \vee \overline{x_4}$



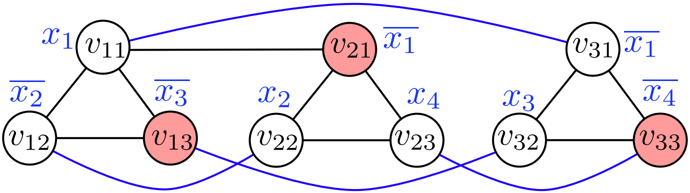
- Claim: Input to 3-SAT is satisfiable iff G has an independent set of size k.
- Satisfiable assignment $\rightarrow$ independent set of size k: Each triangle in G has at least one node whose label evaluates to 1. Set S of nodes consisting of one such node from each triangle forms an independent set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

Proving 3-SAT $\leq_P$ Independent Set

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

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

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



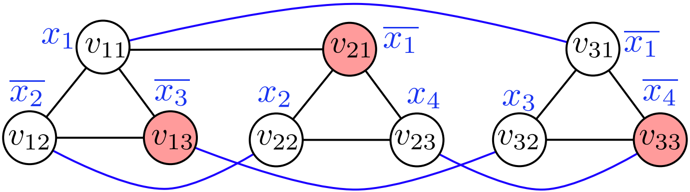
- Claim: Input to 3-SAT is satisfiable iff G has an independent set of size k.
- Satisfiable assignment $\rightarrow$ independent set of size k: Each triangle in G has at least one node whose label evaluates to 1. Set S of nodes consisting of one such node from each triangle forms an independent set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

Proving 3-SAT $\leq_P$ Independent Set

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

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

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



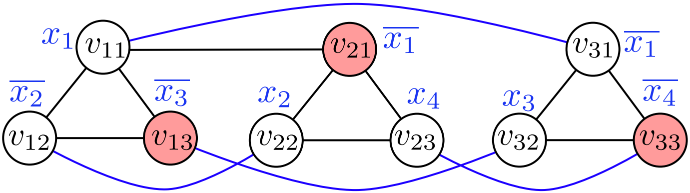
- Claim: Input to 3-SAT is satisfiable iff G has an independent set of size k.
- Satisfiable assignment $\rightarrow$ independent set of size k: Each triangle in G has at least one node whose label evaluates to 1. Set S of nodes consisting of one such node from each triangle forms an independent set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

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

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

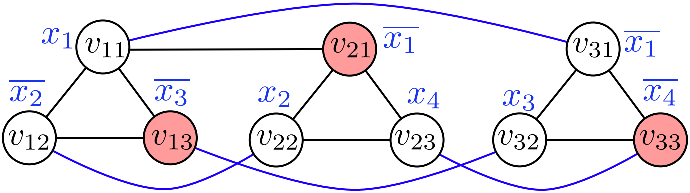
$C_3 = \overline{x_1} \vee x_3 \vee \overline{x_4}$



- Claim: Input to 3-SAT 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 $= k$. 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-SAT $\leq_P$ 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: Input to 3-SAT 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 $= k$. 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 $\mathrm{Z} \leq_P \mathrm{Y}$ and $\mathrm{Y} \leq_P \mathrm{X}$, then $\mathrm{Z} \leq_P \mathrm{X}$.

Transitivity of Reductions

- Claim: If $\text{Z} \leq_P \text{Y}$ and $\text{Y} \leq_P \text{X}$, then $\text{Z} \leq_P \text{X}$.
- We have shown

  3-SAT $\leq_P$ Independent Set $\leq_P$ Vertex Cover $\leq_P$ 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 $\texttt{yes}$ or $\texttt{no}$.
- An algorithm A for a decision problem receives an input s and returns $A(s) \in \{\texttt{yes}, \texttt{no}\}$.
- An algorithm A *solves* the problem X if for every input s,
  - if $X(s) = \texttt{yes}$ then $A(s) = \texttt{yes}$ and
  - if $X(s) = \texttt{no}$ then $A(s) = \texttt{no}$

Problems and Algorithms

PRIMES
INSTANCE: A natural number n
QUESTION: Is n prime?

- Decision problem X: for every input s, answer $X(s)$ is $\texttt{yes}$ or $\texttt{no}$.
- An algorithm A for a decision problem receives an input s and returns $A(s) \in \{\texttt{yes}, \texttt{no}\}$.
- An algorithm A *solves* the problem X if for every input s,
  - if $X(s) = \texttt{yes}$ then $A(s) = \texttt{yes}$ and
  - if $X(s) = \texttt{no}$ then $A(s) = \texttt{no}$
- 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 $p(|s|) = |s|^{12}$ for PRIMES (Agarwal, Kayal, Saxena, 2002, improved to $|s|^6$ by Pomerance and Lenstra, 2005).

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 $A(s) \in \{\texttt{yes}, \texttt{no}\}$.
- An algorithm A *solves* the problem X if for every input s,
  - if $X(s) = \texttt{yes}$ then $A(s) = \texttt{yes}$ and
  - if $X(s) = \texttt{no}$ then $A(s) = \texttt{no}$
- 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 $p(|s|) = |s|^{12}$ for PRIMES (Agarwal, Kayal, Saxena, 2002, improved to $|s|^6$ by Pomerance and Lenstra, 2005).
- $\mathcal{P}$: 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 $\texttt{yes}$ or $\texttt{no}$.
- An algorithm A for a decision problem receives an input s and returns $A(s) \in \{\texttt{yes}, \texttt{no}\}$.
- An algorithm A *solves* the problem X if for every input s,
  - if $X(s) = \texttt{yes}$ then $A(s) = \texttt{yes}$ and
  - if $X(s) = \texttt{no}$ then $A(s) = \texttt{no}$
- 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 $p(|s|) = |s|^{12}$ for PRIMES (Agarwal, Kayal, Saxena, 2002, improved to $|s|^6$ by Pomerance and Lenstra, 2005).
- $\mathcal{P}$: 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 s as well as a separate "certificate" t that contains evidence that $X(s) = \texttt{yes}$.

Efficient Certification

- A "checking" algorithm for a decision problem X has a different structure from an algorithm that solves X.
- Checking algorithm needs input s as well as a separate "certificate" t that contains evidence that $X(s) = \texttt{yes}$.
- 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
     - $X(s) = \texttt{yes}$ iff there is a certificate t such that $B(s, t) = \texttt{yes}$ and
     - the size of t is polynomial in the size of s.

Efficient Certification

- A "checking" algorithm for a decision problem X has a different structure from an algorithm that solves X.
- Checking algorithm needs input s as well as a separate "certificate" t that contains evidence that $X(s) = \texttt{yes}$.
- 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
     - $X(s) = \texttt{yes}$ iff there is a certificate t such that $B(s, t) = \texttt{yes}$ and
     - the size of t is polynomial in the size of s.
- Certifier's job is to take a candidate certificate (t) that $s \in X$ and check in polynomial time whether t is a correct certificate.
- Certificate 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 s as well as a separate "certificate" t that contains evidence that $X(s) = \texttt{yes}$.
- 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
     - ⋆ $X(s) = \texttt{yes}$ iff there is a certificate t such that $B(s, t) = \texttt{yes}$ and
     - ⋆ the size of t is polynomial in the size of s.
- Certifier's job is to take a candidate certificate (t) that $s \in X$ and check in polynomial time whether t is a correct certificate.
- Certificate t must be "short" so that certifier can run in polynomial time.
- Certifier does not care about how to find these certificates.

$\mathcal{NP}$

- $\mathcal{P}$: set of problems X for which there is a polynomial time algorithm.

$\mathcal{NP}$

- $\mathcal{P}$: set of problems X for which there is a polynomial time algorithm.
- $\mathcal{NP}$ is the set of all problems for which there exists an efficient certifier.
- $3\text{-SAT} \in \mathcal{NP}$:

$\mathcal{NP}$

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

$\mathcal{NP}$

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

$\mathcal{NP}$

- $\mathcal{P}$: set of problems X for which there is a polynomial time algorithm.
- $\mathcal{NP}$ is the set of all problems for which there exists an efficient certifier.
- $\text{3-SAT} \in \mathcal{NP}$:
  - Certificate t: a truth assignment to the variables.
  - Certifier B: checks whether assignment causes each clause to evaluate to true.
- $\textsc{Independent Set} \in \mathcal{NP}$:
  - Certificate t: .
  - Certifier B: .

$\mathcal{NP}$

- $\mathcal{P}$: set of problems X for which there is a polynomial time algorithm.
- $\mathcal{NP}$ is the set of all problems for which there exists an efficient certifier.
- $3\text{-SAT} \in \mathcal{NP}$:
  - Certificate t: a truth assignment to the variables.
  - Certifier B: checks whether assignment causes each clause to evaluate to true.
- $\text{Independent Set} \in \mathcal{NP}$:
  - Certificate t: a set of at least k vertices.
  - Certifier B: .

$\mathcal{NP}$

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

$\mathcal{NP}$

- $\mathcal{P}$: set of problems X for which there is a polynomial time algorithm.
- $\mathcal{NP}$ is the set of all problems for which there exists an efficient certifier.
- 3-SAT $\in \mathcal{NP}$:
  - Certificate t: a truth assignment to the variables.
  - Certifier B: checks whether assignment causes each clause to evaluate to true.
- Independent Set $\in \mathcal{NP}$:
  - Certificate t: a set of at least k vertices.
  - Certifier B: checks that no pair of these vertices are connected by an edge.
- Set Cover $\in \mathcal{NP}$:
  - Certificate t: .
  - Certifier B: .

$\mathcal{NP}$

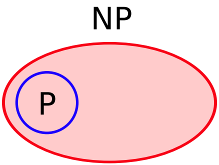
- $\mathcal{P}$: set of problems X for which there is a polynomial time algorithm.
- $\mathcal{NP}$ is the set of all problems for which there exists an efficient certifier.
- 3-SAT $\in \mathcal{NP}$:
  - Certificate t: a truth assignment to the variables.
  - Certifier B: checks whether assignment causes each clause to evaluate to true.
- Independent Set $\in \mathcal{NP}$:
  - Certificate t: a set of at least k vertices.
  - Certifier B: checks that no pair of these vertices are connected by an edge.
- Set Cover $\in \mathcal{NP}$:
  - Certificate t: a list of k sets from the collection.
  - Certifier B: .

$\mathcal{NP}$

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

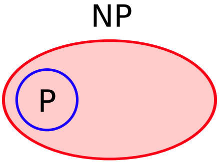
$\mathcal{P}$ vs. $\mathcal{NP}$

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



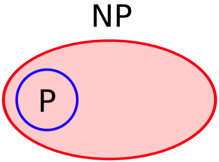
$\mathcal{P}$ vs. $\mathcal{NP}$

- Claim: $\mathcal{P} \subseteq \mathcal{NP}$.
  - Let X be any problem in $\mathcal{P}$.
  - There is a polynomial time algorithm A that solves X.



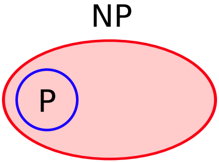
$\mathcal{P}$ vs. $\mathcal{NP}$

- Claim: $\mathcal{P} \subseteq \mathcal{NP}$.
  - Let X be any problem in $\mathcal{P}$.
  - There is a polynomial time algorithm A that solves X.
  - B ignores t and simply returns $A(s)$. Why is B an efficient certifier?



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

- Claim: $\mathcal{P} \subseteq \mathcal{NP}$.
  - Let X be any problem in $\mathcal{P}$.
  - There is a polynomial time algorithm A that solves X.
  - B ignores t and simply 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: $\mathcal{P} \subseteq \mathcal{NP}$.
  - Let X be any problem in $\mathcal{P}$.
  - There is a polynomial time algorithm A that solves X.
  - B ignores t and simply 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: $\mathcal{P} \subseteq \mathcal{NP}$.
  - Let X be any problem in $\mathcal{P}$.
  - There is a polynomial time algorithm A that solves X.
  - B ignores t and simply 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.

CMI ABOUT PROGRAMS PEOPLE MILLENNIUM PROBLEMS PUBLICATIONS EVENTS NEWS

P vs NP Problem

Suppose that you are organizing housing accommodations for a group of four hundred university students. Space is limited and only one hundred of the students will receive places in the dormitory. To complicate matters, the Dean has provided you with a list of pairs of incompatible students, and requested that no pair from this list appear in your final choice. This is an example of what computer scientists call an NP-problem, since it is easy to check if a given choice of one hundred students proposed by a coworker is satisfactory (i.e., no pair taken from your coworker's list also appears on the list from the Dean's office), however the task of generating such a list from scratch seems to be so hard as to be completely impractical. Indeed, the total number of ways of choosing one hundred students from the four hundred applicants is greater than the number of atoms in the known universe! Thus no future civilization could ever hope to build a supercomputer capable of solving the problem by brute force; that is, by checking every possible combination of 100 students. However, this apparent difficulty may only reflect the lack of ingenuity of your programmer. In fact, one of the outstanding problems in computer science is determining whether questions exist whose answer can be quickly checked, but which require an impossibly long time to solve by any direct procedure. Problems like the one listed above certainly seem to be of this kind, but so far no one has managed to prove that any of them really are so hard as they appear, i.e., that there really is no feasible way to generate an answer with the help of a computer. Stephen Cook and Leonid Levin formulated the P (i.e., easy to find) versus NP (i.e., easy to check) problem independently in 1971.

Image credit: on the left, Stephen Cook by Jiří Janíček (cropped). CC BY-SA 3.0

Rules:

Rules for the Millennium Prizes

Related Documents:

Official Problem Description

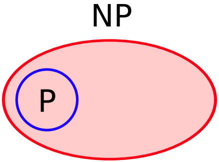
Minesweeper

Related Links:

Lecture by Vijaya Ramachandran

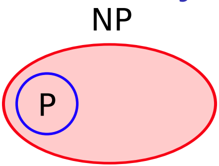
This problem is: Unsolved

Summary



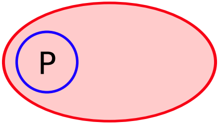
- Dealing with *decision problems*: for every input, the answer is `yes` or `no`.
- A problem is in $\mathcal{P}$ if there is a polynomial time algorithm that solves it.

Summary



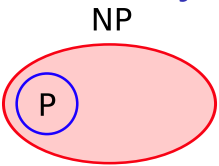
- Dealing with *decision problems*: for every input, the answer is `yes` or `no`.
- A problem is in $\mathcal{P}$ if there is a polynomial time algorithm that solves it.
- A problem is in $\mathcal{NP}$ if there is a polynomial time certifying algorithm for `yes` inputs:
  - Given an input and a "certificate", the certifier can use the certificate to verify in polynomial time if the answer is `yes` for that input.
  - Definition of $\mathcal{NP}$ does not care about inputs for which the answer is `no`.

Summary



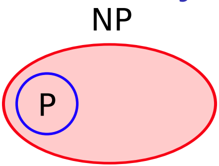
- Dealing with *decision problems*: for every input, the answer is `yes` or `no`.
- A problem is in $\mathcal{P}$ if there is a polynomial time algorithm that solves it.
- A problem is in $\mathcal{NP}$ if there is a polynomial time certifying algorithm for `yes` inputs:
  - Given an input and a "certificate", the certifier can use the certificate to verify in polynomial time if the answer is `yes` for that input.
  - Definition of $\mathcal{NP}$ does not care about inputs for which the answer is `no`.
- $\mathcal{P} \subseteq \mathcal{NP}$
- 3-SAT, VertexCover, SetCover, IndependentSet are in $\mathcal{NP}$.
- 3-SAT $\leq_P$ Independent Set $\leq_P$ Vertex Cover $\leq_P$ Set Cover

Summary



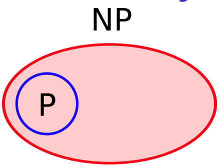
- Dealing with *decision problems*: for every input, the answer is `yes` or `no`.
- A problem is in $\mathcal{P}$ if there is a polynomial time algorithm that solves it.
- A problem is in $\mathcal{NP}$ if there is a polynomial time certifying algorithm for `yes` inputs:
  - Given an input and a "certificate", the certifier can use the certificate to verify in polynomial time if the answer is `yes` for that input.
  - Definition of $\mathcal{NP}$ does not care about inputs for which the answer is `no`.
- $\mathcal{P} \subseteq \mathcal{NP}$
- 3-SAT, VertexCover, SetCover, IndependentSet are in $\mathcal{NP}$.
- 3-SAT $\leq_P$ Independent Set $\leq_P$ Vertex Cover $\leq_P$ Set Cover
- What is the structure of the problems in $\mathcal{NP}$?

Summary



- Dealing with *decision problems*: for every input, the answer is yes or no.
- A problem is in $\mathcal{P}$ if there is a polynomial time algorithm that solves it.
- A problem is in $\mathcal{NP}$ if there is a polynomial time certifying algorithm for yes inputs:
  - Given an input and a "certificate", the certifier can use the certificate to verify in polynomial time if the answer is yes for that input.
  - Definition of $\mathcal{NP}$ does not care about inputs for which the answer is no.
- $\mathcal{P} \subseteq \mathcal{NP}$
- 3-SAT, VertexCover, SetCover, IndependentSet are in $\mathcal{NP}$.
- 3-SAT $\leq_P$ Independent Set $\leq_P$ Vertex Cover $\leq_P$ Set Cover
- What is the structure of the problems in $\mathcal{NP}$?
  1. Is there a sequence of problems $X_1, X_2, X_3, \ldots$ in $\mathcal{NP}$, such that $X_1 \leq_P X_2 \leq_P X_3 \leq_P \ldots$?

Summary



- Dealing with *decision problems*: for every input, the answer is `yes` or `no`.
- A problem is in $\mathcal{P}$ if there is a polynomial time algorithm that solves it.
- A problem is in $\mathcal{NP}$ if there is a polynomial time certifying algorithm for `yes` inputs:
  - Given an input and a "certificate", the certifier can use the certificate to verify in polynomial time if the answer is `yes` for that input.
  - Definition of $\mathcal{NP}$ does not care about inputs for which the answer is `no`.
- $\mathcal{P} \subseteq \mathcal{NP}$
- 3-SAT, VERTEXCOVER, SETCOVER, INDEPENDENTSET are in $\mathcal{NP}$.
- 3-SAT $\leq_P$ INDEPENDENT SET $\leq_P$ VERTEX COVER $\leq_P$ SET COVER
- What is the structure of the problems in $\mathcal{NP}$?
  1. Is there a sequence of problems $X_1, X_2, X_3, \ldots$ in $\mathcal{NP}$, such that $X_1 \leq_P X_2 \leq_P X_3 \leq_P \ldots$?
  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 $\mathcal{NP}$?

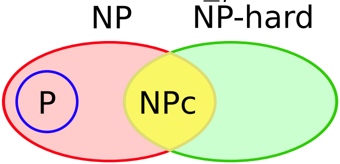
A problem X is *$\mathcal{NP}$-Complete* if

(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 $\mathcal{NP}$?

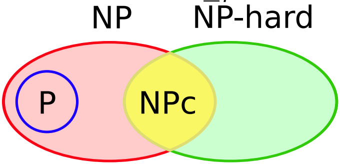
A problem X is *$\mathcal{NP}$-Complete* if

(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 $\mathcal{NP}$?

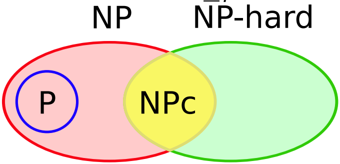
A problem X is *$\mathcal{NP}$-Complete* if

(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.

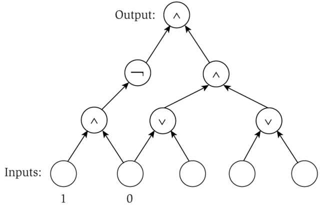


Figure 8.4 A circuit with three inputs, two additional sources that have assigned truth values, and one output.

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.

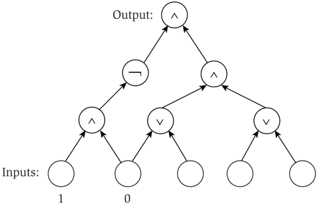


Figure 8.4 A circuit with three inputs, two additional sources that have assigned truth values, and one output.

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 $X \leq_P$ Circuit Satisfiability.

Proving Circuit Satisfiability is $\mathcal{NP}$-Complete

- 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.

Proving Circuit Satisfiability is $\mathcal{NP}$-Complete

- 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.

Proving Circuit Satisfiability is $\mathcal{NP}$-Complete

- 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?

Proving Circuit Satisfiability is $\mathcal{NP}$-Complete

- 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 $B(\cdot, \cdot)$.

Proving Circuit Satisfiability is $\mathcal{NP}$-Complete

- 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 certificate 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 certificate t of length $p(|s|)$ such that $B(s, t) = \texttt{yes}$?"

Proving Circuit Satisfiability is $\mathcal{NP}$-Complete

- To determine whether $s \in X$, we ask "Is there a certificate t of length $p(|s|)$ such that $B(s, t) = \texttt{yes}$?"
- 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 certificate t of length $p(|s|)$ such that $B(s, t) = \texttt{yes}$?"
- 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.

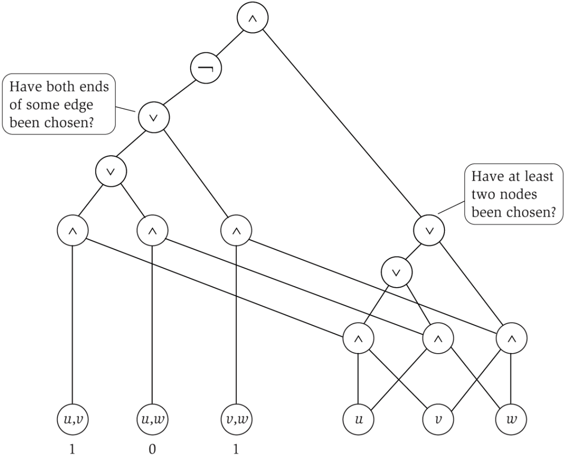


Figure 8.5 A circuit to verify whether a 3-node graph contains a 2-node independent set.

Asymmetry of Certification

- Definition of efficient certification and $\mathcal{NP}$ is fundamentally asymmetric:
  - An input s is a "yes" instance iff there exists a short certificate t such that $B(s,t) = \texttt{yes}$.
  - An input s is a "no" instance iff *for all* short certificates t, $B(s,t) = \texttt{no}$.

Asymmetry of Certification

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

co-$\mathcal{NP}$

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

co-$\mathcal{NP}$

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

co-$\mathcal{NP}$

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

co-$\mathcal{NP}$

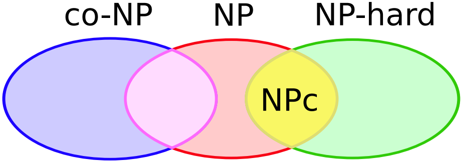
- For a decision problem X, its *complementary problem* $\overline{X}$ is the set of inputs s such that $s \in \overline{X}$ iff $s \notin X$.
- If $X \in \mathcal{P}$, then $\overline{X} \in \mathcal{P}$.
- If $X \in \mathcal{NP}$, then is $\overline{X} \in \mathcal{NP}$?

co-$\mathcal{NP}$

- For a decision problem X, its *complementary problem* $\overline{X}$ is the set of inputs 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 $\mathcal{NP}$.

co-$\mathcal{NP}$

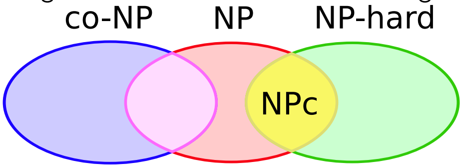
- For a decision problem X, its *complementary problem* $\overline{X}$ is the set of inputs s such that $s \in \overline{X}$ iff $s \notin X$.
- If $X \in \mathcal{P}$, then $\overline{X} \in \mathcal{P}$.
- If $X \in \mathcal{NP}$, then is $\overline{X} \in \mathcal{NP}$? Unclear in general.
- A problem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



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

co-$\mathcal{NP}$

- For a decision problem X, its *complementary problem* $\overline{X}$ is the set of inputs 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 $\mathcal{NP}$.



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

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

- If a problem belongs to both $\mathcal{NP}$ and 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 $\mathcal{NP} \cap$ co-$\mathcal{NP}$

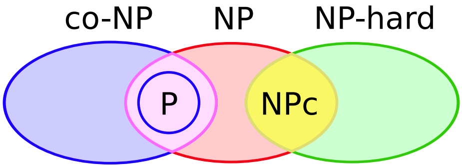
- If a problem belongs to both $\mathcal{NP}$ and co-$\mathcal{NP}$, then
  - When the answer is yes, there is a short proof.
  - When the answer is no, there is a short proof.
- Problems in $\mathcal{NP} \cap$ co-$\mathcal{NP}$ have a *good characterisation*.

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

- If a problem belongs to both $\mathcal{NP}$ and 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 $\mathcal{NP} \cap$ 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 $\mathcal{NP} \cap$ co-$\mathcal{NP}$

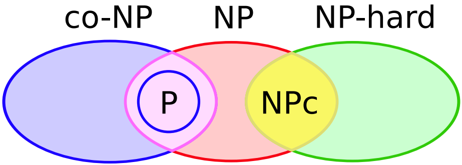
- If a problem belongs to both $\mathcal{NP}$ and co-$\mathcal{NP}$, then
  - When the answer is yes, there is a short proof.
  - When the answer is no, there is a short proof.
- Problems in $\mathcal{NP} \cap$ co-$\mathcal{NP}$ have a *good characterisation*.
- Example is the problem of determining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

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

- If a problem belongs to both $\mathcal{NP}$ and 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 $\mathcal{NP} \cap$ 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: $\mathcal{P} \subseteq \mathcal{NP} \cap$ co-$\mathcal{NP}$.
- Open problem: Is $\mathcal{P} = \mathcal{NP} \cap$ co-$\mathcal{NP}$?