CS514: Design and Analysis of Algorithms

Intractability

Arijit Mondal

Dept of CSE

arijit@iitp.ac.in

https://www.iitp.ac.in/~arijit/

Simple vs Hard problems

- Shortest path vs Longest path in a graph

Simple vs Hard problems

- Shortest path vs Longest path in a graph
- Euler tour vs Hamiltonian cycle
  - An Euler tour of a strongly connected, directed graph $G = (V, E)$ is a cycle that traverses each edge of G exactly once, although it is allowed to visit each vertex more than once
  - A Hamiltonian cycle of a directed graph $G = (V, E)$ is a simple cycle that contains each vertex in V

Simple vs Hard problems

- Shortest path vs Longest path in a graph
- Euler tour vs Hamiltonian cycle
  - An Euler tour of a strongly connected, directed graph $G = (V, E)$ is a cycle that traverses each edge of G exactly once, although it is allowed to visit each vertex more than once
  - A Hamiltonian cycle of a directed graph $G = (V, E)$ is a simple cycle that contains each vertex in V
- 2-SAT vs 3-SAT
  - 2-SAT: $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (x_2 \vee \neg x_3)$
  - 3-SAT: $(x_1 \vee \neg x_2 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4)$

Simple vs Hard problems

- Shortest path vs Longest path in a graph
- Euler tour vs Hamiltonian cycle
  - An Euler tour of a strongly connected, directed graph $G = (V, E)$ is a cycle that traverses each edge of G exactly once, although it is allowed to visit each vertex more than once
  - A Hamiltonian cycle of a directed graph $G = (V, E)$ is a simple cycle that contains each vertex in V
- 2-SAT vs 3-SAT
  - 2-SAT: $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (x_2 \vee \neg x_3)$
  - 3-SAT: $(x_1 \vee \neg x_2 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4)$
- Fractional vs 0-1 knapsack

Time complexity

Time complexity function	Size n					
	10	20	30	40	50	60
n	.00001 second	.00002 second	.00003 second	.00004 second	.00005 second	.00006 second
n^2	.0001 second	.0004 second	.0009 second	.0016 second	.0025 second	.0036 second
n^3	.001 second	.008 second	.027 second	.064 second	.125 second	.216 second
n^5	.1 second	3.2 seconds	24.3 seconds	1.7 minutes	5.2 minutes	13.0 minutes
2^n	.001 second	1.0 second	17.9 minutes	12.7 days	35.7 years	366 centuries
3^n	.059 second	58 minutes	6.5 years	3855 centuries	2×10^8 centuries	1.3×10^{13} centuries

Image source: Computers and Intractability

Problem class

- P – consists of those problems that are solvable in polynomial time

Problem class

- P – consists of those problems that are solvable in polynomial time
- NP – consists of those problems that are '*verifiable*' in polynomial time

Problem class

- P – consists of those problems that are solvable in polynomial time
- NP – consists of those problems that are ‘*verifiable*’ in polynomial time
- NPC – problem belongs to NP and is as **hard** as any problem in NP

Problem class

- P – consists of those problems that are solvable in polynomial time
- NP – consists of those problems that are '*verifiable*' in polynomial time
- NPC – problem belongs to NP and is as **hard** as any problem in NP
- It is obvious that P $\subseteq$ NP. However, the famous open question is whether P is a proper subset of NP

Problem class

- P – consists of those problems that are solvable in polynomial time
- NP – consists of those problems that are '*verifiable*' in polynomial time
- NPC – problem belongs to NP and is as **hard** as any problem in NP
- It is obvious that P $\subseteq$ NP. However, the famous open question is whether P is a proper subset of NP
- If any NP-complete problem can be solved in polynomial time, then every problem in NP has a polynomial-time algorithm

Optimization vs Decision problems

- For **optimization problems**, each feasible solution is associated with a value, and goal is to find a feasible solution with the best value

Optimization vs Decision problems

- For **optimization problems**, each feasible solution is associated with a value, and goal is to find a feasible solution with the best value
  - Shortest path
  - Travelling salesman problem

Optimization vs Decision problems

- For **optimization problems**, each feasible solution is associated with a value, and goal is to find a feasible solution with the best value
  - Shortest path
  - Travelling salesman problem
- In **decision problems**, the final answer is either ‘yes’ or ‘no’

Optimization vs Decision problems

- For **optimization problems**, each feasible solution is associated with a value, and goal is to find a feasible solution with the best value
  - Shortest path
  - Travelling salesman problem
- In **decision problems**, the final answer is either ‘yes’ or ‘no’
  - 2-SAT
  - Hamiltonian cycle

Optimization vs Decision problems

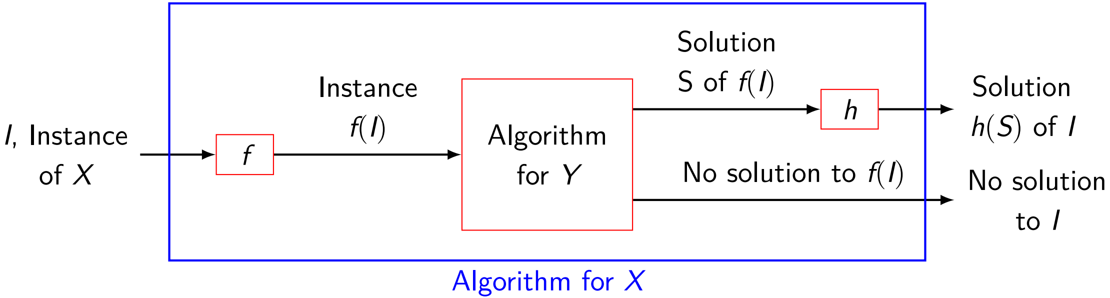
- For **optimization problems**, each feasible solution is associated with a value, and goal is to find a feasible solution with the best value
  - Shortest path
  - Travelling salesman problem
- In **decision problems**, the final answer is either 'yes' or 'no'
  - 2-SAT
  - Hamiltonian cycle
- Which problem is harder?

Optimization vs Decision problems

- For **optimization problems**, each feasible solution is associated with a value, and goal is to find a feasible solution with the best value
  - Shortest path
  - Travelling salesman problem
- In **decision problems**, the final answer is either 'yes' or 'no'
  - 2-SAT
  - Hamiltonian cycle
- Which problem is harder?
- Can an optimization problem be converted as decision problem?

Reduction

- $X \leq_p Y$
- Problem X polynomial-time reduces to problem Y if arbitrary instances of problem X can be solved using:
  - Polynomial number of standard computational steps (f, h), plus
  - Polynomial number of calls to oracle that solves problem Y



Poly-time Reduction

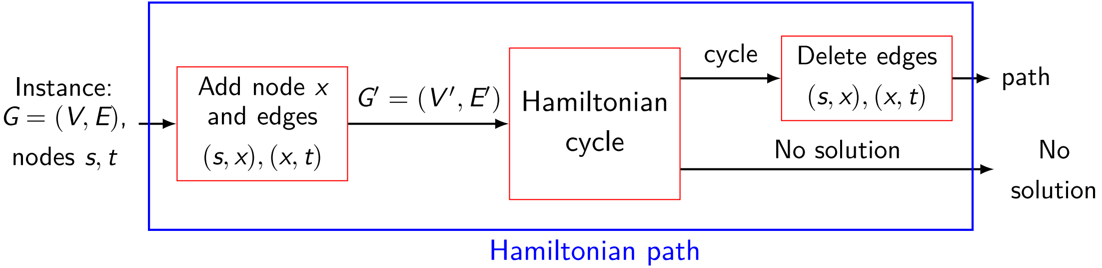
- If $X \leq_p Y$ and Y can be solved in polynomial time, then X can be solved in polynomial time
- If $X \leq_p Y$ and X cannot be solved in polynomial time, then Y cannot be solved in polynomial time

Poly-time Reduction

- If $X \leq_p Y$ and Y can be solved in polynomial time, then X can be solved in polynomial time
- If $X \leq_p Y$ and X cannot be solved in polynomial time, then Y cannot be solved in polynomial time
- If $X \leq_p Y$ and Y can be solved in exponential time, then X – ??

Hamiltonian path $\to$ Hamiltonian cycle

- Hamiltonian cycle: given a graph, is there a cycle that passes through each vertex exactly once?
- Hamiltonian path(s, t): given a graph, is there a path between s and t that passes through each vertex exactly once?



Abstract problem

- An abstract problem Q is defined to be binary relation on a set I of problem instances and a set S of problem solution
  - For shortest-path – problem instance consists of a graph and two vertices, $I = \langle G, u, v \rangle$
  - A solution is sequence of vertices or null if it does not exist
- For NP-Completeness, we are primarily interested in decision problems
- For shortest-path, decision problem can be represented as $I = \langle G, u, v, k \rangle$
  - Given a graph and two vertices, does there exist a path with at most k edges?
- An optimization problem can be converted to decision problem

Encoding

- In order for a computer program to solve an abstract problem, its problem instances must appear in a way that the program understands

Encoding

- In order for a computer program to solve an abstract problem, its problem instances must appear in a way that the program understands
- An encoding of a set S of abstract objects is a mapping e from S to the set of binary strings

Encoding

- In order for a computer program to solve an abstract problem, its problem instances must appear in a way that the program understands
- An encoding of a set S of abstract objects is a mapping e from S to the set of binary strings
- Example: Graph $G = (V, E)$ where $V = [v_1, v_2, v_3, v_4]$, and $E = \{\{v_1, v_2\}, \{v_2, v_3\}\}$

Encoding

- In order for a computer program to solve an abstract problem, its problem instances must appear in a way that the program understands
- An encoding of a set S of abstract objects is a mapping e from S to the set of binary strings
- Example: Graph $G = (V, E)$ where $V = [v_1, v_2, v_3, v_4]$, and $E = \{\{v_1, v_2\}, \{v_2, v_3\}\}$

Encoding scheme	string	length
Vertex and edge list	$v_1 v_2 v_3 v_4 (v_1 v_2)(v_2 v_3)$	36
Neighbor list	$(v_2)(v_1 v_3)(v_2)()$	24
Adjacency matrix rows	0100/1010/0100/0000	19

Encoding

- In order for a computer program to solve an abstract problem, its problem instances must appear in a way that the program understands
- An encoding of a set S of abstract objects is a mapping e from S to the set of binary strings
- Example: Graph $G = (V, E)$ where $V = [v_1, v_2, v_3, v_4]$, and $E = \{\{v_1, v_2\}, \{v_2, v_3\}\}$

Encoding scheme	string	length
Vertex and edge list	$v_1 v_2 v_3 v_4 (v_1 v_2)(v_2 v_3)$	36
Neighbor list	$(v_2)(v_1 v_3)(v_2)()$	24
Adjacency matrix rows	0100/1010/0100/0000	19

- The size of an instance I is just the length of its string, $n = |I|$

Encoding

- In order for a computer program to solve an abstract problem, its problem instances must appear in a way that the program understands
- An encoding of a set S of abstract objects is a mapping e from S to the set of binary strings
- Example: Graph $G = (V, E)$ where $V = [v_1, v_2, v_3, v_4]$, and $E = \{\{v_1, v_2\}, \{v_2, v_3\}\}$

Encoding scheme	string	length
Vertex and edge list	$v_1 v_2 v_3 v_4 (v_1 v_2)(v_2 v_3)$	36
Neighbor list	$(v_2)(v_1 v_3)(v_2)()$	24
Adjacency matrix rows	0100/1010/0100/0000	19

- The size of an instance I is just the length of its string, $n = |I|$
- We call a problem, whose instance set is the set of binary strings, a concrete problem

Encoding

- In order for a computer program to solve an abstract problem, its problem instances must appear in a way that the program understands
- An encoding of a set S of abstract objects is a mapping e from S to the set of binary strings
- Example: Graph $G = (V, E)$ where $V = [v_1, v_2, v_3, v_4]$, and $E = \{\{v_1, v_2\}, \{v_2, v_3\}\}$

Encoding scheme	string	length
Vertex and edge list	$v_1 v_2 v_3 v_4 (v_1 v_2)(v_2 v_3)$	36
Neighbor list	$(v_2)(v_1 v_3)(v_2)()$	24
Adjacency matrix rows	0100/1010/0100/0000	19

- The size of an instance I is just the length of its string, $n = |I|$
- We call a problem, whose instance set is the set of binary strings, a concrete problem
- A concrete problem is polynomial-time solvable if there exist an algorithm to solve it in $O(n^k)$ time for some constant k

Encoding

- We say a function $f\colon \{0,1\}^* \to \{0,1\}^*$ is polynomial-time computable if there exists a polynomial-time algorithm A that given any input $x \in \{0,1\}^*$, produces as output $f(x)$
- We say that two encodings e_1 and e_2 are polynomially related if there exist two polynomial-time computable function f_{12} and f_{21} such that for any $i \in I$, we have $f_{12}(e_1(i)) = e_2(i)$ and $f_{21}(e_2(i)) = e_1(i)$

Encoding

- We say a function $f : \{0,1\}^* \to \{0,1\}^*$ is polynomial-time computable if there exists a polynomial-time algorithm A that given any input $x \in \{0,1\}^*$, produces as output $f(x)$
- We say that two encodings e_1 and e_2 are polynomially related if there exist two polynomial-time computable function f_{12} and f_{21} such that for any $i \in I$, we have $f_{12}(e_1(i)) = e_2(i)$ and $f_{21}(e_2(i)) = e_1(i)$
- Let Q be an abstract decision problem on an instance set I, and let e_1 and e_2 be polynomially related encodings on I. Then, $e_1(Q) \in \mathrm{P}$ if and only if $e_2(Q) \in \mathrm{P}$.

Formal language framework

- An alphabet Σ is a finite set of symbols

Formal language framework

- An alphabet Σ is a finite set of symbols
- A language L over Σ is any set of strings made up of symbols from Σ

Formal language framework

- An alphabet Σ is a finite set of symbols
- A language L over Σ is any set of strings made up of symbols from Σ
- Example: if $\Sigma = \{0, 1\}$, the set $L = \{10, 11, 101, 111, 11001, \ldots\}$

Formal language framework

- An alphabet Σ is a finite set of symbols
- A language L over Σ is any set of strings made up of symbols from Σ
- Example: if $\Sigma = \{0, 1\}$, the set $L = \{10, 11, 101, 111, 11001, \ldots\}$
- Empty string is denoted by ε, the empty language by $\emptyset$, the language of all strings over Σ by Σ^*

Formal language framework

- An alphabet Σ is a finite set of symbols
- A language L over Σ is any set of strings made up of symbols from Σ
- Example: if $\Sigma = \{0, 1\}$, the set $L = \{10, 11, 101, 111, 11001, \ldots\}$
- Empty string is denoted by ε, the empty language by $\emptyset$, the language of all strings over Σ by Σ^*
- Every language L over Σ is a subset of Σ^*

Formal language framework

- An alphabet Σ is a finite set of symbols
- A language L over Σ is any set of strings made up of symbols from Σ
- Example: if $\Sigma = \{0, 1\}$, the set $L = \{10, 11, 101, 111, 11001, \ldots\}$
- Empty string is denoted by ε, the empty language by $\emptyset$, the language of all strings over Σ by Σ^*
- Every language L over Σ is a subset of Σ^*
- Any decision problem Q is simply the set Σ^* where $\Sigma = \{0, 1\}$

Formal language framework

- An alphabet Σ is a finite set of symbols
- A language L over Σ is any set of strings made up of symbols from Σ
- Example: if $\Sigma = \{0, 1\}$, the set $L = \{10, 11, 101, 111, 11001, \ldots\}$
- Empty string is denoted by ε, the empty language by $\emptyset$, the language of all strings over Σ by Σ^*
- Every language L over Σ is a subset of Σ^*
- Any decision problem Q is simply the set Σ^* where $\Sigma = \{0, 1\}$
- As Q is entirely characterized by those problem instances that produce 1 (yes) answer, we can view Q as a language L over $\Sigma = \{0, 1\}$, where $L = \{x \in \Sigma^* : Q(x) = 1\}$

Formal language framework

- An alphabet Σ is a finite set of symbols
- A language L over Σ is any set of strings made up of symbols from Σ
- Example: if $\Sigma = \{0, 1\}$, the set $L = \{10, 11, 101, 111, 11001, \ldots\}$
- Empty string is denoted by ε, the empty language by $\emptyset$, the language of all strings over Σ by Σ^*
- Every language L over Σ is a subset of Σ^*
- Any decision problem Q is simply the set Σ^* where $\Sigma = \{0, 1\}$
- As Q is entirely characterized by those problem instances that produce 1 (yes) answer, we can view Q as a language L over $\Sigma = \{0, 1\}$, where $L = \{x \in \Sigma^* : Q(x) = 1\}$
- A language L is decided in polynomial-time by an algorithm A if there exists a constant k such that for any length-n string $x \in \{0, 1\}^*$, the algorithm correctly decides whether $x \in L$ in $O(n^k)$ time

Complexity class

- P$=\{L \subseteq \{0,1\}^*$: there exists an algorithm A that decides L in polynomial time $\}$

Complexity class

- P=$\{L \subseteq \{0,1\}^*$: there exists an algorithm A that decides L in polynomial time $\}$
- A verification algorithm is a **two-argument algorithm** A, where one argument is an ordinary **input** string x and the other is a binary string y called a **certificate**
- A two-argument algorithm A verifies an input string x if there exists a certificate y such that $A(x, y) = 1$

Complexity class

- P=$\{L \subseteq \{0,1\}^*$: there exists an algorithm A that decides L in polynomial time $\}$
- A verification algorithm is a **two-argument algorithm** A, where one argument is an ordinary **input** string x and the other is a binary string y called a **certificate**
- A two-argument algorithm A verifies an input string x if there exists a certificate y such that $A(x, y) = 1$
- NP is the class of languages that can be verified by a polynomial-time algorithm
  $L = \{x \in \{0,1\}^*$: there exists a certificate y with $|y| = O(|x|^c)$ such that $A(x, y) = 1\}$

Complexity class

- P=$\{L \subseteq \{0,1\}^*$: there exists an algorithm A that decides L in polynomial time $\}$
- A verification algorithm is a **two-argument algorithm** A, where one argument is an ordinary **input** string x and the other is a binary string y called a **certificate**
- A two-argument algorithm A verifies an input string x if there exists a certificate y such that $A(x, y) = 1$
- NP is the class of languages that can be verified by a polynomial-time algorithm
  $L = \{x \in \{0,1\}^*$: there exists a certificate y with $|y| = O(|x|^c)$ such that $A(x, y) = 1\}$
- If $L \in$ P, then $L \in$ NP, thus, P $\subseteq$ NP

Complexity class

- P=$\{L \subseteq \{0,1\}^*$: there exists an algorithm A that decides L in polynomial time $\}$
- A verification algorithm is a **two-argument algorithm** A, where one argument is an ordinary **input** string x and the other is a binary string y called a **certificate**
- A two-argument algorithm A verifies an input string x if there exists a certificate y such that $A(x, y) = 1$
- NP is the class of languages that can be verified by a polynomial-time algorithm $L = \{x \in \{0,1\}^*$: there exists a certificate y with $|y| = O(|x|^c)$ such that $A(x, y) = 1\}$
- If $L \in$ P, then $L \in$ NP, thus, P $\subseteq$ NP
- It leaves the question of whether P $=$ NP

NP-completeness

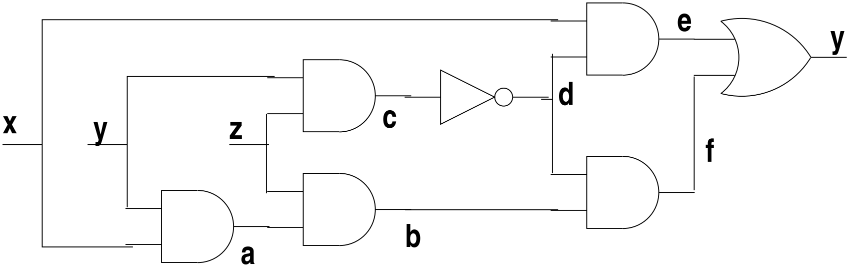
- A language $L \subseteq \{0,1\}^*$ is **NP-complete (NPC)** if
  - $L \in$ NP, and
  - $L' \leq_P L$ for every $L' \in$ NP
- If an language L satisfies the 2nd property but not necessarily the 1st, we say L is **NP-hard**

NP-completeness

- A language $L \subseteq \{0,1\}^*$ is **NP-complete (NPC)** if
  - $L \in \mathrm{NP}$, and
  - $L' \leq_P L$ for every $L' \in \mathrm{NP}$
- If an language L satisfies the 2nd property but not necessarily the 1st, we say L is **NP-hard**
- If any NP-complete problem is polynomial-time solvable, then $\mathrm{P} = \mathrm{NP}$. Equivalently, if any problem in NP is not polynomial-time solvable, then no NP-complete problem is polynomial-time solvable.

Circuit-SAT

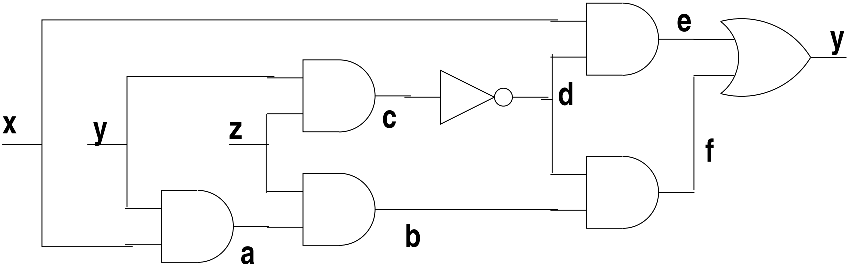
- Circuit-SAT: Given a boolean combinational circuit composed of AND, OR, and NOT gates, is it satisfiable?



Circuit-SAT

- Circuit-SAT: Given a boolean combinational circuit composed of AND, OR, and NOT gates, is it satisfiable?

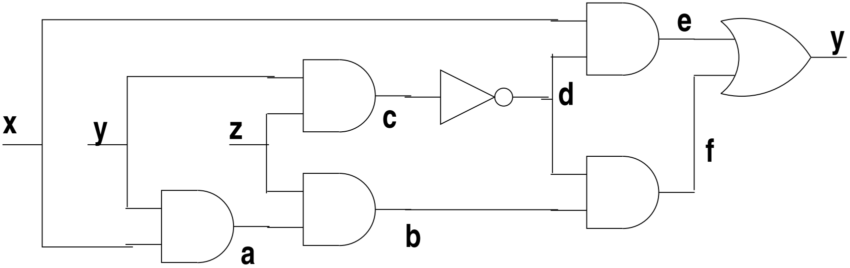
- Circuit-SAT $\in$ NP



Circuit-SAT

- Circuit-SAT: Given a boolean combinational circuit composed of AND, OR, and NOT gates, is it satisfiable?

- Circuit-SAT $\in$ NP
- Circuit-SAT is also NP-Hard (see detailed proof in the book)



NP-completeness proofs

- It is difficult to prove that every language in NP can be reduced to the given language

NP-completeness proofs

- It is difficult to prove that every language in NP can be reduced to the given language

- If L is language such that $L' \leq_P L$ for some $L' \in$ NPC, the L is NP-hard.
  If, in addition we have $L \in$ NP, then $L \in$ NPC

NP-completeness proofs

- It is difficult to prove that every language in NP can be reduced to the given language

- If L is language such that $L' \leq_P L$ for some $L' \in$ NPC, the L is NP-hard.

  If, in addition we have $L \in$ NP, then $L \in$ NPC

  - **Proof:** Since L' is NP-complete, for all $L'' \in$ NP, we have $L'' \leq_P L'$

NP-completeness proofs

- It is difficult to prove that every language in NP can be reduced to the given language

- If L is language such that $L' \leq_P L$ for some $L' \in$ NPC, the L is NP-hard.

  If, in addition we have $L \in$ NP, then $L \in$ NPC

  - **Proof:** Since L' is NP-complete, for all $L'' \in$ NP, we have $L'' \leq_P L'$
  - As we have, $L' \leq_P L$, thus by transitivity, we can say $L'' \leq_P L$

NP-completeness proofs

- It is difficult to prove that every language in NP can be reduced to the given language
- If L is language such that $L' \leq_P L$ for some $L' \in$ NPC, the L is NP-hard.

  If, in addition we have $L \in$ NP, then $L \in$ NPC
  - **Proof:** Since L' is NP-complete, for all $L'' \in$ NP, we have $L'' \leq_P L'$
  - As we have, $L' \leq_P L$, thus by transitivity, we can say $L'' \leq_P L$
  - So, L is NP-hard

NP-completeness proofs

- It is difficult to prove that every language in NP can be reduced to the given language

- If L is language such that $L' \leq_P L$ for some $L' \in$ NPC, the L is NP-hard.

  If, in addition we have $L \in$ NP, then $L \in$ NPC

  - **Proof:** Since L' is NP-complete, for all $L'' \in$ NP, we have $L'' \leq_P L'$
  - As we have, $L' \leq_P L$, thus by transitivity, we can say $L'' \leq_P L$
  - So, L is NP-hard
  - If we have, $L \in$ NP, then we also have $L \in$ NPC

Steps to prove NP-completeness

- Prove $L \in \text{NP}$

Steps to prove NP-completeness

- Prove $L \in$ NP
- Prove that L is NP-hard:

Steps to prove NP-completeness

- Prove $L \in \text{NP}$
- Prove that L is NP-hard:
  - Select a known NP-complete language L'
  - Describe an algorithm that computes a function f mapping every instance $x \in \{0,1\}^*$ of L' to an instance of $f(x)$ of L
  - Prove that the function f satisfies $x \in L'$ if and only if $f(x) \in L$ for all $x \in \{0,1\}^*$
  - Prove that the algorithm computing f runs in polynomial time

SAT $\in$ NPC

- SAT: inputs – n Boolean variables, m connectives $(\wedge, \vee, \neg, \rightarrow, \leftrightarrow)$, and parentheses

SAT $\in$ NPC

- SAT: inputs – n Boolean variables, m connectives $(\wedge, \vee, \neg, \rightarrow, \leftrightarrow)$, and parentheses
- The boolean formula ϕ can be encoded in length that is polynomial in $n + m$
- Example: $(x_1 \rightarrow x_2) \wedge ((x_3 \vee \neg x_4) \wedge (x_3 \leftrightarrow x_4))$

SAT $\in$ NPC

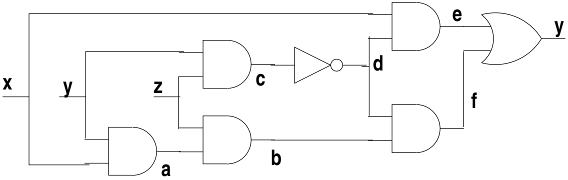
- SAT: inputs – n Boolean variables, m connectives $(\wedge, \vee, \neg, \rightarrow, \leftrightarrow)$, and parentheses
- The boolean formula ϕ can be encoded in length that is polynomial in $n + m$
- Example: $(x_1 \rightarrow x_2) \wedge ((x_3 \vee \neg x_4) \wedge (x_3 \leftrightarrow x_4))$
- Given a SAT instance and an assignment of the variables (certificate) - it can be verified in polynomial time

SAT $\in$ NPC

- SAT: inputs – n Boolean variables, m connectives $(\wedge, \vee, \neg, \rightarrow, \leftrightarrow)$, and parentheses
- The boolean formula ϕ can be encoded in length that is polynomial in $n + m$
- Example: $(x_1 \rightarrow x_2) \wedge ((x_3 \vee \neg x_4) \wedge (x_3 \leftrightarrow x_4))$
- Given a SAT instance and an assignment of the variables (certificate) - it can be verified in polynomial time
- To prove NP-hard, we need to show Circuit-SAT $\leq_P$ SAT

SAT $\in$ NPC

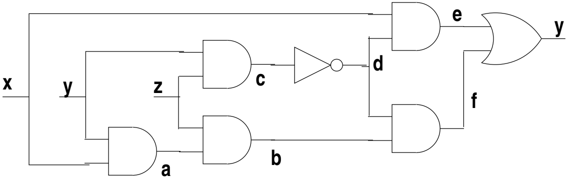
- SAT: inputs – n Boolean variables, m connectives $(\wedge, \vee, \neg, \rightarrow, \leftrightarrow)$, and parentheses
- The boolean formula ϕ can be encoded in length that is polynomial in $n + m$
- Example: $(x_1 \rightarrow x_2) \wedge ((x_3 \vee \neg x_4) \wedge (x_3 \leftrightarrow x_4))$
- Given a SAT instance and an assignment of the variables (certificate) - it can be verified in polynomial time
- To prove NP-hard, we need to show Circuit-SAT $\leq_P$ SAT
- We can express any boolean combinational circuit (C) as a boolean formula (ϕ)



SAT $\in$ NPC

- SAT: inputs – n Boolean variables, m connectives $(\wedge, \vee, \neg, \rightarrow, \leftrightarrow)$, and parentheses
- The boolean formula ϕ can be encoded in length that is polynomial in $n + m$
- Example: $(x_1 \rightarrow x_2) \wedge ((x_3 \vee \neg x_4) \wedge (x_3 \leftrightarrow x_4))$
- Given a SAT instance and an assignment of the variables (certificate) - it can be verified in polynomial time
- To prove NP-hard, we need to show Circuit-SAT $\leq_P$ SAT
- We can express any boolean combinational circuit (C) as a boolean formula (ϕ)

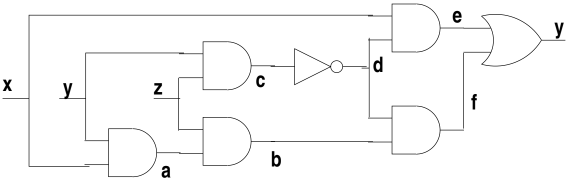
$$\begin{aligned}\phi = \quad & y \wedge (y \leftrightarrow (e \vee f)) \wedge (e \leftrightarrow (x \wedge d)) \wedge \\ & (f \leftrightarrow (d \wedge b)) \wedge (d \leftrightarrow \neg c) \wedge \\ & (b \leftrightarrow (z \wedge a)) \wedge (c \leftrightarrow (z \wedge y)) \wedge \\ & (a \leftrightarrow (x \wedge y))\end{aligned}$$



SAT ∈ NPC

- We can express any boolean combinational circuit (C) as a boolean formula (ϕ)

$$\phi = y \wedge (y \leftrightarrow (e \vee f)) \wedge (e \leftrightarrow (x \wedge d)) \wedge (f \leftrightarrow (d \wedge b)) \wedge (d \leftrightarrow \neg c) \wedge$$
$$(b \leftrightarrow (z \wedge a)) \wedge (c \leftrightarrow (z \wedge y)) \wedge (a \leftrightarrow (x \wedge y))$$

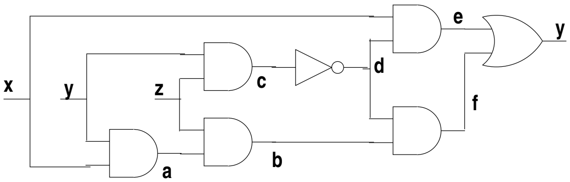


SAT $\in$ NPC

- We can express any boolean combinational circuit (C) as a boolean formula (ϕ)

$$\phi = \; y \wedge (y \leftrightarrow (e \vee f)) \wedge (e \leftrightarrow (x \wedge d)) \wedge (f \leftrightarrow (d \wedge b)) \wedge (d \leftrightarrow \neg c) \wedge (b \leftrightarrow (z \wedge a)) \wedge (c \leftrightarrow (z \wedge y)) \wedge (a \leftrightarrow (x \wedge y))$$

- Now we need to show C is satisfiable exactly when ϕ is satisfiable

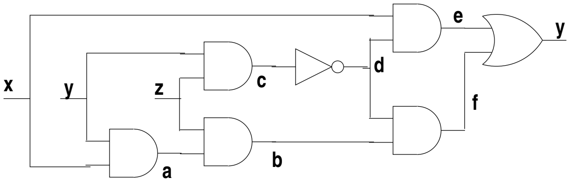


SAT $\in$ NPC

- We can express any boolean combinational circuit (C) as a boolean formula (ϕ)

$$\phi = \; y \wedge (y \leftrightarrow (e \vee f)) \wedge (e \leftrightarrow (x \wedge d)) \wedge (f \leftrightarrow (d \wedge b)) \wedge (d \leftrightarrow \neg c) \wedge$$
$$(b \leftrightarrow (z \wedge a)) \wedge (c \leftrightarrow (z \wedge y)) \wedge (a \leftrightarrow (x \wedge y))$$

- Now we need to show C is satisfiable exactly when ϕ is satisfiable
- If C has a satisfying assignment, then each wire of the circuit has well defined value and output is 1

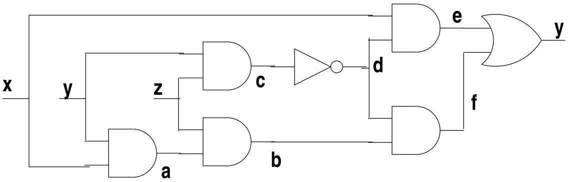


SAT $\in$ NPC

- We can express any boolean combinational circuit (C) as a boolean formula (ϕ)

$$\phi = \quad y \wedge (y \leftrightarrow (e \vee f)) \wedge (e \leftrightarrow (x \wedge d)) \wedge (f \leftrightarrow (d \wedge b)) \wedge (d \leftrightarrow \neg c) \wedge$$
$$(b \leftrightarrow (z \wedge a)) \wedge (c \leftrightarrow (z \wedge y)) \wedge (a \leftrightarrow (x \wedge y))$$

- Now we need to show C is satisfiable exactly when ϕ is satisfiable
- If C has a satisfying assignment, then each wire of the circuit has well defined value and output is 1
- We can assign the wire values to variables in ϕ, each clause will evaluate to 1, hence, $\phi = 1$

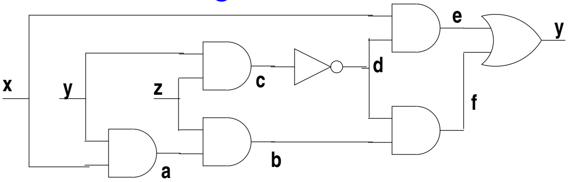


SAT $\in$ NPC

- We can express any boolean combinational circuit (C) as a boolean formula (ϕ)

$$\phi = y \wedge (y \leftrightarrow (e \vee f)) \wedge (e \leftrightarrow (x \wedge d)) \wedge (f \leftrightarrow (d \wedge b)) \wedge (d \leftrightarrow \neg c) \wedge (b \leftrightarrow (z \wedge a)) \wedge (c \leftrightarrow (z \wedge y)) \wedge (a \leftrightarrow (x \wedge y))$$

- Now we need to show C is satisfiable exactly when ϕ is satisfiable
- If C has a satisfying assignment, then each wire of the circuit has well defined value and output is 1
- We can assign the wire values to variables in ϕ, each clause will evaluate to 1, hence, $\phi = 1$
- If some assignment causes ϕ to evaluate to 1, we can assign values to different wires and it will evaluate to 1 for C



Circuit-SAT $\leq_P$ CNF-SAT

- Literal - variable in boolean formula, x_1 or $\neg x_1$
- Clause - OR of any number of literals, $x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4$
- CNF (conjunctive normal form) - AND of clauses

Circuit-SAT $\leq_P$ CNF-SAT

- Literal - variable in boolean formula, x_1 or $\neg x_1$
- Clause - OR of any number of literals, $x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4$
- CNF (conjunctive normal form) - AND of clauses
- CNF-SAT: $(x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_2 \vee \neg x_1)$

Circuit-SAT $\leq_P$ CNF-SAT

- Literal - variable in boolean formula, x_1 or $\neg x_1$
- Clause - OR of any number of literals, $x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4$
- CNF (conjunctive normal form) - AND of clauses
- CNF-SAT: $(x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_2 \vee \neg x_1)$
- Let us assume that the circuit consists of AND, OR and NOT gates. We assume AND and OR gates can have either 2 or 3 inputs

Circuit-SAT $\leq_P$ CNF-SAT

- Literal - variable in boolean formula, x_1 or $\neg x_1$
- Clause - OR of any number of literals, $x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4$
- CNF (conjunctive normal form) - AND of clauses
- CNF-SAT: $(x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_2 \vee \neg x_1)$
- Let us assume that the circuit consists of AND, OR and NOT gates. We assume AND and OR gates can have either 2 or 3 inputs
- For NOT gate:

Circuit-SAT $\leq_P$ CNF-SAT

- Literal - variable in boolean formula, x_1 or $\neg x_1$
- Clause - OR of any number of literals, $x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4$
- CNF (conjunctive normal form) - AND of clauses
- CNF-SAT: $(x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_2 \vee \neg x_1)$
- Let us assume that the circuit consists of AND, OR and NOT gates. We assume AND and OR gates can have either 2 or 3 inputs
- For NOT gate:

$$y \leftrightarrow \neg x = (\neg y \vee \neg x) \wedge (x \vee y)$$

Circuit-SAT $\leq_P$ CNF-SAT

- Literal - variable in boolean formula, x_1 or $\neg x_1$
- Clause - OR of any number of literals, $x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4$
- CNF (conjunctive normal form) - AND of clauses
- CNF-SAT: $(x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_2 \vee \neg x_1)$
- Let us assume that the circuit consists of AND, OR and NOT gates. We assume AND and OR gates can have either 2 or 3 inputs
- For NOT gate:

  $$y \leftrightarrow \neg x = (\neg y \vee \neg x) \wedge (x \vee y)$$

- For 2-input AND gate:

Circuit-SAT $\leq_P$ CNF-SAT

- Literal - variable in boolean formula, x_1 or $\neg x_1$
- Clause - OR of any number of literals, $x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4$
- CNF (conjunctive normal form) - AND of clauses
- CNF-SAT: $(x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_2 \vee \neg x_1)$
- Let us assume that the circuit consists of AND, OR and NOT gates. We assume AND and OR gates can have either 2 or 3 inputs
- For NOT gate:

  $$y \leftrightarrow \neg x = (\neg y \vee \neg x) \wedge (x \vee y)$$

- For 2-input AND gate:

  $$y \leftrightarrow (a \wedge b) = (y \rightarrow (a \wedge b)) \wedge ((a \wedge b) \rightarrow y)$$
  $$= (\neg y \vee (a \wedge b)) \wedge (\neg a \vee \neg b \vee y) = (\neg y \vee a) \wedge (\neg y \vee b) \wedge (\neg a \vee \neg b \vee y)$$

Circuit-SAT $\leq_P$ CNF-SAT

- For 3-input AND gate:

Circuit-SAT $\leq_P$ CNF-SAT

- For 3-input AND gate:

$$y \leftrightarrow (a \wedge b \wedge c) = (\neg y \vee a) \wedge (\neg y \vee b) \wedge (\neg y \vee c) \wedge (\neg a \vee \neg b \vee \neg c \vee y)$$

Circuit-SAT $\leq_P$ CNF-SAT

- For 3-input AND gate:

$$y \leftrightarrow (a \wedge b \wedge c) = (\neg y \vee a) \wedge (\neg y \vee b) \wedge (\neg y \vee c) \wedge (\neg a \vee \neg b \vee \neg c \vee y)$$

- For 2-input OR gate:

Circuit-SAT $\leq_P$ CNF-SAT

- For 3-input AND gate:

  $y \leftrightarrow (a \wedge b \wedge c) = (\neg y \vee a) \wedge (\neg y \vee b) \wedge (\neg y \vee c) \wedge (\neg a \vee \neg b \vee \neg c \vee y)$

- For 2-input OR gate:

  $y \leftrightarrow (a \vee b) = (y \rightarrow (a \vee b)) \wedge ((a \vee b) \rightarrow y)$
  $= (\neg y \vee a \vee b) \wedge ((\neg a \wedge \neg b) \vee y) = (\neg y \vee a \vee b) \wedge (\neg a \vee y) \wedge (\neg b \vee y)$

Circuit-SAT $\leq_P$ CNF-SAT

- For 3-input AND gate:

$$y \leftrightarrow (a \wedge b \wedge c) = (\neg y \vee a) \wedge (\neg y \vee b) \wedge (\neg y \vee c) \wedge (\neg a \vee \neg b \vee \neg c \vee y)$$

- For 2-input OR gate:

$$y \leftrightarrow (a \vee b) = (y \rightarrow (a \vee b)) \wedge ((a \vee b) \rightarrow y)$$
$$= (\neg y \vee a \vee b) \wedge ((\neg a \wedge \neg b) \vee y) = (\neg y \vee a \vee b) \wedge (\neg a \vee y) \wedge (\neg b \vee y)$$

- For 3-input OR gate:

Circuit-SAT $\leq_P$ CNF-SAT

- For 3-input AND gate:

$$y \leftrightarrow (a \wedge b \wedge c) = (\neg y \vee a) \wedge (\neg y \vee b) \wedge (\neg y \vee c) \wedge (\neg a \vee \neg b \vee \neg c \vee y)$$

- For 2-input OR gate:

$$y \leftrightarrow (a \vee b) = (y \rightarrow (a \vee b)) \wedge ((a \vee b) \rightarrow y)$$
$$= (\neg y \vee a \vee b) \wedge ((\neg a \wedge \neg b) \vee y) = (\neg y \vee a \vee b) \wedge (\neg a \vee y) \wedge (\neg b \vee y)$$

- For 3-input OR gate:

$$y \leftrightarrow (a \vee b \vee c) = (\neg y \vee a \vee b \vee c) \wedge (\neg a \vee y) \wedge (\neg b \vee y) \wedge (\neg c \vee y)$$

Circuit-SAT $\leq_P$ CNF-SAT

- For 3-input AND gate:

  $y \leftrightarrow (a \wedge b \wedge c) = (\neg y \vee a) \wedge (\neg y \vee b) \wedge (\neg y \vee c) \wedge (\neg a \vee \neg b \vee \neg c \vee y)$

- For 2-input OR gate:

  $y \leftrightarrow (a \vee b) = (y \rightarrow (a \vee b)) \wedge ((a \vee b) \rightarrow y)$
  $= (\neg y \vee a \vee b) \wedge ((\neg a \wedge \neg b) \vee y) = (\neg y \vee a \vee b) \wedge (\neg a \vee y) \wedge (\neg b \vee y)$

- For 3-input OR gate:

  $y \leftrightarrow (a \vee b \vee c) = (\neg y \vee a \vee b \vee c) \wedge (\neg a \vee y) \wedge (\neg b \vee y) \wedge (\neg c \vee y)$

- Circuit-SAT can be converted to CNF-SAT in polynomial time using above transformations
- It can be shown Circuit-SAT has a solution if and only if CNF-SAT has a solution

3-CNF-SAT $\in$ NPC

- 3-CNF-SAT: each clause has exactly 3 literals

3-CNF-SAT $\in$ NPC

- 3-CNF-SAT: each clause has exactly 3 literals
- Given an assignment of the variables, truth value can be verified in linear time. Hence 3-CNF-SAT $\in$ NP

3-CNF-SAT $\in$ NPC

- 3-CNF-SAT: each clause has exactly 3 literals
- Given an assignment of the variables, truth value can be verified in linear time. Hence 3-CNF-SAT $\in$ NP
- We choose CNF-SAT, where clauses can have 1, 2, 3, or more literals, to reduce to 3-CNF-SAT

3-CNF-SAT $\in$ NPC

- 3-CNF-SAT: each clause has exactly 3 literals
- Given an assignment of the variables, truth value can be verified in linear time. Hence 3-CNF-SAT $\in$ NP
- We choose CNF-SAT, where clauses can have 1, 2, 3, or more literals, to reduce to 3-CNF-SAT
- Clause with 1 literal:

3-CNF-SAT $\in$ NPC

- 3-CNF-SAT: each clause has exactly 3 literals
- Given an assignment of the variables, truth value can be verified in linear time. Hence 3-CNF-SAT $\in$ NP
- We choose CNF-SAT, where clauses can have 1, 2, 3, or more literals, to reduce to 3-CNF-SAT
- Clause with 1 literal:

$$x_1 \equiv (x_1 \vee z_1 \vee z_2) \wedge (x_1 \vee \neg z_1 \vee z_2) \wedge (x_1 \vee z_1 \vee \neg z_2) \wedge (x_1 \vee \neg z_1 \vee \neg z_2)$$

3-CNF-SAT $\in$ NPC

- 3-CNF-SAT: each clause has exactly 3 literals
- Given an assignment of the variables, truth value can be verified in linear time. Hence 3-CNF-SAT $\in$ NP
- We choose CNF-SAT, where clauses can have 1, 2, 3, or more literals, to reduce to 3-CNF-SAT
- Clause with 1 literal:

$$x_1 \equiv (x_1 \vee z_1 \vee z_2) \wedge (x_1 \vee \neg z_1 \vee z_2) \wedge (x_1 \vee z_1 \vee \neg z_2) \wedge (x_1 \vee \neg z_1 \vee \neg z_2)$$

- Clause with 2 literals:

3-CNF-SAT $\in$ NPC

- 3-CNF-SAT: each clause has exactly 3 literals
- Given an assignment of the variables, truth value can be verified in linear time. Hence 3-CNF-SAT $\in$ NP
- We choose CNF-SAT, where clauses can have 1, 2, 3, or more literals, to reduce to 3-CNF-SAT
- Clause with 1 literal:

$$x_1 \equiv (x_1 \vee z_1 \vee z_2) \wedge (x_1 \vee \neg z_1 \vee z_2) \wedge (x_1 \vee z_1 \vee \neg z_2) \wedge (x_1 \vee \neg z_1 \vee \neg z_2)$$

- Clause with 2 literals:

$$x_1 \vee x_2 \equiv (x_1 \vee x_2 \vee z_1) \wedge (x_1 \vee x_2 \vee \neg z_1)$$

3-CNF-SAT $\in$ NPC

- 3-CNF-SAT: each clause has exactly 3 literals
- Given an assignment of the variables, truth value can be verified in linear time. Hence 3-CNF-SAT $\in$ NP
- We choose CNF-SAT, where clauses can have 1, 2, 3, or more literals, to reduce to 3-CNF-SAT
- Clause with 1 literal:

  $$x_1 \equiv (x_1 \vee z_1 \vee z_2) \wedge (x_1 \vee \neg z_1 \vee z_2) \wedge (x_1 \vee z_1 \vee \neg z_2) \wedge (x_1 \vee \neg z_1 \vee \neg z_2)$$

- Clause with 2 literals:

  $$x_1 \vee x_2 \equiv (x_1 \vee x_2 \vee z_1) \wedge (x_1 \vee x_2 \vee \neg z_1)$$

- Clause with 3 literals:

3-CNF-SAT $\in$ NPC

- 3-CNF-SAT: each clause has exactly 3 literals
- Given an assignment of the variables, truth value can be verified in linear time. Hence 3-CNF-SAT $\in$ NP
- We choose CNF-SAT, where clauses can have 1, 2, 3, or more literals, to reduce to 3-CNF-SAT
- Clause with 1 literal:

$$x_1 \equiv (x_1 \vee z_1 \vee z_2) \wedge (x_1 \vee \neg z_1 \vee z_2) \wedge (x_1 \vee z_1 \vee \neg z_2) \wedge (x_1 \vee \neg z_1 \vee \neg z_2)$$

- Clause with 2 literals:

$$x_1 \vee x_2 \equiv (x_1 \vee x_2 \vee z_1) \wedge (x_1 \vee x_2 \vee \neg z_1)$$

- Clause with 3 literals: No need to change

3-CNF-SAT $\in$ NPC

- Clause with >3 literals: $X = (x_1 \vee x_2 \vee \ldots x_k)$

3-CNF-SAT $\in$ NPC

- Clause with >3 literals: $X = (x_1 \vee x_2 \vee \ldots x_k)$

  $Y = (x_1 \vee x_2 \vee z_1)(\neg z_1 \vee x_3 \vee z_2)(\neg z_2 \vee x_4 \vee z_3) \ldots (\neg z_{k-3} \vee x_{k-1} \vee x_k)$

3-CNF-SAT $\in$ NPC

- Clause with >3 literals: $X = (x_1 \vee x_2 \vee \ldots x_k)$

  $Y = (x_1 \vee x_2 \vee z_1)(\neg z_1 \vee x_3 \vee z_2)(\neg z_2 \vee x_4 \vee z_3) \ldots (\neg z_{k-3} \vee x_{k-1} \vee x_k)$

- The above conversion can be done in polynomial time

3-CNF-SAT $\in$ NPC

- Clause with >3 literals: $X = (x_1 \vee x_2 \vee \ldots x_k)$

  $Y = (x_1 \vee x_2 \vee z_1)(\neg z_1 \vee x_3 \vee z_2)(\neg z_2 \vee x_4 \vee z_3)\ldots(\neg z_{k-3} \vee x_{k-1} \vee x_k)$
- The above conversion can be done in polynomial time
- We need to show: { X is satisfied } $\leftrightarrow$ { there is a setting of z_i's s.t. Y is satisfied }

3-CNF-SAT $\in$ NPC

- Clause with >3 literals: $X = (x_1 \vee x_2 \vee \ldots x_k)$

  $Y = (x_1 \vee x_2 \vee z_1)(\neg z_1 \vee x_3 \vee z_2)(\neg z_2 \vee x_4 \vee z_3)\ldots(\neg z_{k-3} \vee x_{k-1} \vee x_k)$
- The above conversion can be done in polynomial time
- We need to show: { X is satisfied } $\leftrightarrow$ { there is a setting of z_i's s.t. Y is satisfied }
- Assume Y is satisfied: we can claim at least one literals $x_1, \ldots, x_k$ must be true.

  How? What is the implication on X?

3-CNF-SAT $\in$ NPC

- Clause with >3 literals: $X = (x_1 \vee x_2 \vee \ldots x_k)$

  $Y = (x_1 \vee x_2 \vee z_1)(\neg z_1 \vee x_3 \vee z_2)(\neg z_2 \vee x_4 \vee z_3)\ldots(\neg z_{k-3} \vee x_{k-1} \vee x_k)$

- The above conversion can be done in polynomial time
- We need to show: { X is satisfied } $\leftrightarrow$ { there is a setting of z_i's s.t. Y is satisfied }
- Assume Y is satisfied: we can claim at least one literals $x_1, \ldots, x_k$ must be true. How? What is the implication on X?
- Conversely, if X is satisfied: some x_i must be true.

3-CNF-SAT $\in$ NPC

- Clause with >3 literals: $X = (x_1 \vee x_2 \vee \ldots x_k)$

  $Y = (x_1 \vee x_2 \vee z_1)(\neg z_1 \vee x_3 \vee z_2)(\neg z_2 \vee x_4 \vee z_3) \ldots (\neg z_{k-3} \vee x_{k-1} \vee x_k)$
- The above conversion can be done in polynomial time
- We need to show: { X is satisfied } $\leftrightarrow$ { there is a setting of z_i's s.t. Y is satisfied }
- Assume Y is satisfied: we can claim at least one literals $x_1, \ldots, x_k$ must be true.

  How? What is the implication on X?
- Conversely, if X is satisfied: some x_i must be true.

  Set $z_1, \ldots, z_{i-2}$ to `true` and rest to `false`

Independent Set (IS) $\in$ NPC

- IS: given a graph G and an integer k, does there exist k vertices that are independent, that is, no two of which have an edge between them

Independent Set (IS) $\in$ NPC

- IS: given a graph G and an integer k, does there exist k vertices that are independent, that is, no two of which have an edge between them
- IS $\in$ NP as a certificate can be verified in polynomial time

Independent Set (IS) $\in$ NPC

- IS: given a graph G and an integer k, does there exist k vertices that are independent, that is, no two of which have an edge between them
- IS $\in$ NP as a certificate can be verified in polynomial time
- We reduce 3-SAT to IS

Independent Set (IS) $\in$ NPC

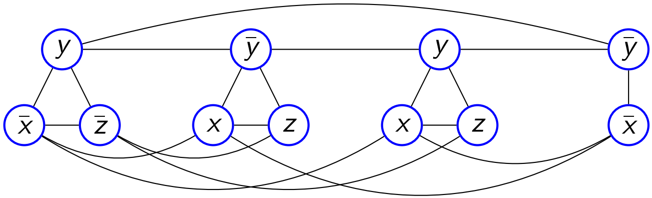
- IS: given a graph G and an integer k, does there exist k vertices that are independent, that is, no two of which have an edge between them
- IS $\in$ NP as a certificate can be verified in polynomial time
- We reduce 3-SAT to IS
- Take an instance of 3-SAT: $X = (\bar{x} \vee y \vee \bar{z})(x \vee \bar{y} \vee z)(x \vee y \vee z)(\bar{x} \vee \bar{y})$

Independent Set (IS) $\in$ NPC

- IS: given a graph G and an integer k, does there exist k vertices that are independent, that is, no two of which have an edge between them
- IS $\in$ NP as a certificate can be verified in polynomial time
- We reduce 3-SAT to IS
- Take an instance of 3-SAT: $X = (\bar{x} \vee y \vee \bar{z})(x \vee \bar{y} \vee z)(x \vee y \vee z)(\bar{x} \vee \bar{y})$
- Construction of G:
  - Each clause is represented as triangle with vertices as the literals
  - Connect an edge between two nodes of different clauses if they represent opposite literals

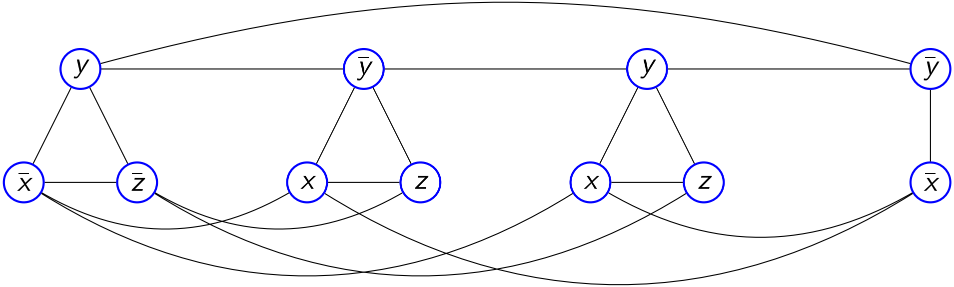
Independent Set (IS) $\in$ NPC

- IS: given a graph G and an integer k, does there exist k vertices that are independent, that is, no two of which have an edge between them
- IS $\in$ NP as a certificate can be verified in polynomial time
- We reduce 3-SAT to IS
- Take an instance of 3-SAT: $X = (\bar{x} \vee y \vee \bar{z})(x \vee \bar{y} \vee z)(x \vee y \vee z)(\bar{x} \vee \bar{y})$
- Construction of G:
  - Each clause is represented as triangle with vertices as the literals
  - Connect an edge between two nodes of different clauses if they represent opposite literals



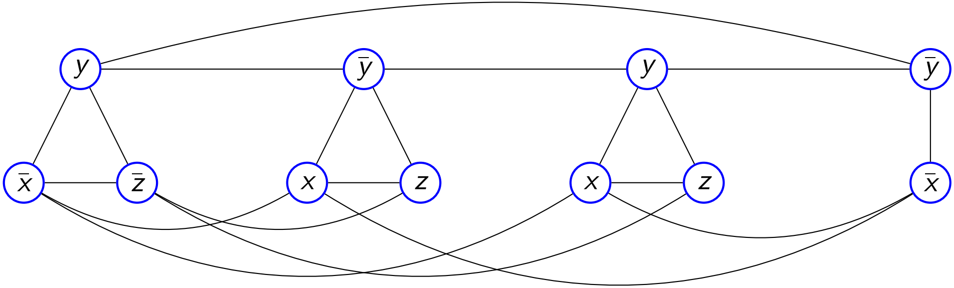
Independent Set (IS) $\in$ NPC

- Given an independent set S of k vertices in G, it is possible to find satisfying truth assignment to I



Independent Set (IS) $\in$ NPC

- Given an independent set S of k vertices in G, it is possible to find satisfying truth assignment to I
- If I has a truth assignment then G has independent set of size k



Vertex Cover (VC) $\in$ NPC

- A vertex cover of an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ such that if $(u, v) \in E$ then $u \in V'$ or $v \in V'$ or both. Does graph G has a vertex cover of size k?

Vertex Cover (VC) $\in$ NPC

- A vertex cover of an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ such that if $(u, v) \in E$ then $u \in V'$ or $v \in V'$ or both. Does graph G has a vertex cover of size k?
- Let a set of nodes S be the vertex cover of G, that is S touches every edge in E

Vertex Cover (VC) $\in$ NPC

- A vertex cover of an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ such that if $(u, v) \in E$ then $u \in V'$ or $v \in V'$ or both. Does graph G has a vertex cover of size k?
- Let a set of nodes S be the vertex cover of G, that is S touches every edge in E
- The remaining nodes $V - S$ must form an independent set!

Vertex Cover (VC) $\in$ NPC

- A vertex cover of an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ such that if $(u, v) \in E$ then $u \in V'$ or $v \in V'$ or both. Does graph G has a vertex cover of size k?
- Let a set of nodes S be the vertex cover of G, that is S touches every edge in E
- The remaining nodes $V - S$ must form an independent set!
- Thus to solve, an instance of (G, k) of independent-set, we simply look for a vertex cover of G with $V - k$ nodes

Vertex Cover (VC) $\in$ NPC

- A vertex cover of an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ such that if $(u, v) \in E$ then $u \in V'$ or $v \in V'$ or both. Does graph G has a vertex cover of size k?
- Let a set of nodes S be the vertex cover of G, that is S touches every edge in E
- The remaining nodes $V - S$ must form an independent set!
- Thus to solve, an instance of (G, k) of independent-set, we simply look for a vertex cover of G with $V - k$ nodes
- If a vertex cover exists, then all nodes not in VC set form IS

Vertex Cover (VC) $\in$ NPC

- A vertex cover of an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ such that if $(u, v) \in E$ then $u \in V'$ or $v \in V'$ or both. Does graph G has a vertex cover of size k?
- Let a set of nodes S be the vertex cover of G, that is S touches every edge in E
- The remaining nodes $V - S$ must form an independent set!
- Thus to solve, an instance of (G, k) of independent-set, we simply look for a vertex cover of G with $V - k$ nodes
- If a vertex cover exists, then all nodes not in VC set form IS
- If no such vertex cover exists, G cannot have an independent set of size k

Clique $\in$ NPC

- A clique in an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices, each pair of which is connected by an edge in E. Given a graph G, does it have a clique of size k?

Clique $\in$ NPC

- A clique in an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices, each pair of which is connected by an edge in E. Given a graph G, does it have a clique of size k?
- A certificate for clique can be verified in polynomial time

Clique $\in$ NPC

- A clique in an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices, each pair of which is connected by an edge in E. Given a graph G, does it have a clique of size k?
- A certificate for clique can be verified in polynomial time
- We reduce 3-CNF-SAT to clique

Clique $\in$ NPC

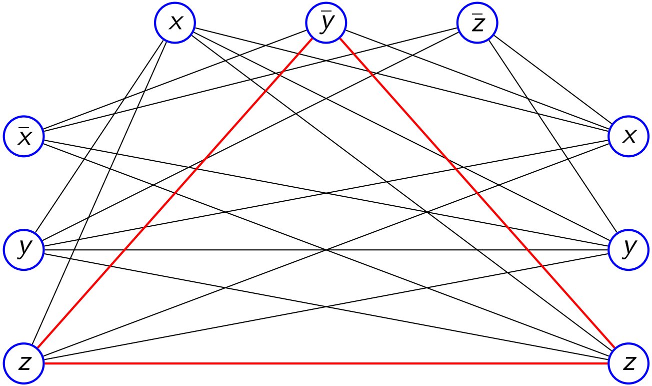
- A clique in an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices, each pair of which is connected by an edge in E. Given a graph G, does it have a clique of size k?
- A certificate for clique can be verified in polynomial time
- We reduce 3-CNF-SAT to clique
- We choose X, CNF-SAT instance, that has k number of clauses $(C_1, \ldots, C_k)$, where each clause has exactly 3 literals

Clique $\in$ NPC

- A clique in an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices, each pair of which is connected by an edge in E. Given a graph G, does it have a clique of size k?
- A certificate for clique can be verified in polynomial time
- We reduce 3-CNF-SAT to clique
- We choose X, CNF-SAT instance, that has k number of clauses $(C_1, \ldots, C_k)$, where each clause has exactly 3 literals
- Graph construction: $G = (V, E)$
  - For each clause $C_r = (l_1^r \vee l_2^r \vee l_3^r)$ in X create three vertices v_1^r, v_2^r, v_3^r into V
  - Add edge (v_i^r, v_j^s) into E if both of the following hold
    - v_i^r and v_j^s are in different triples, that is $r \neq s$, and
    - their corresponding literals are consistent, that is l_i^r is not the negation of l_j^s

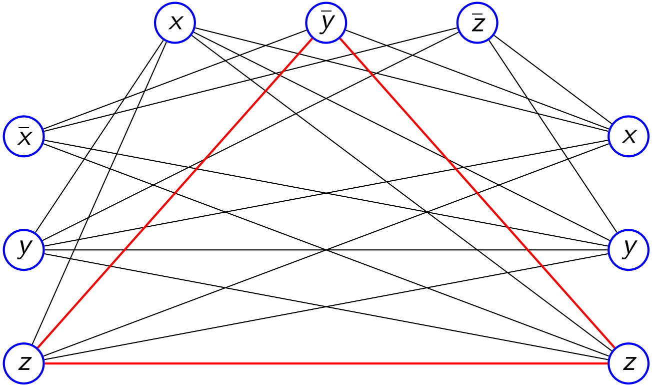
Clique $\in$ NPC

- Consider a 3SAT instance as $X = C_1 \wedge C_2 \wedge C_3 = (\bar{x} \vee y \vee z)(x \vee \bar{y} \vee \bar{z})(x \vee y \vee z)$



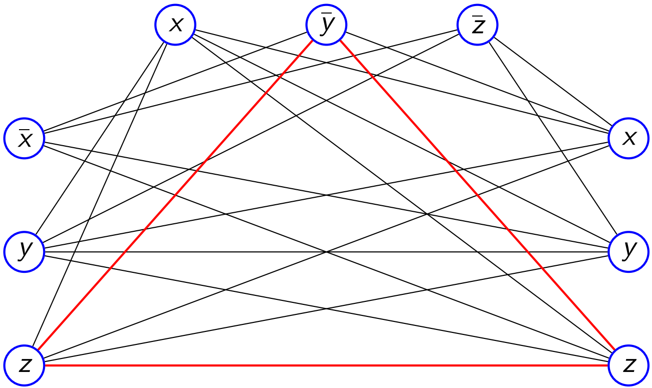
Clique $\in$ NPC

- Consider a 3SAT instance as $X = C_1 \wedge C_2 \wedge C_3 = (\bar{x} \vee y \vee z)(x \vee \bar{y} \vee \bar{z})(x \vee y \vee z)$
- Suppose X has a satisfying assignment: what can we claim?



Clique ∈ NPC

- Consider a 3SAT instance as $X = C_1 \wedge C_2 \wedge C_3 = (\bar{x} \vee y \vee z)(x \vee \bar{y} \vee \bar{z})(x \vee y \vee z)$
- Suppose X has a satisfying assignment: what can we claim?
- Suppose G contains a clique of size k: what can be claimed?



Thank you!