NP-completeness - Part I

K. Subramani[1]

[1] Lane Department of Computer Science and Electrical Engineering
West Virginia University

March 30, 2015

Outline

1. **NP** and **NP-completeness**

Outline

Outline

Outline

Certificate definition of **NP**

Certificate definition of **NP**

Definition

Certificate definition of **NP**

Definition

NP is the class of problems A of the following form:

Certificate definition of **NP**

Definition

NP is the class of problems A of the following form:

x is a yes-instance of A if and only if there exists a w,

Certificate definition of **NP**

Definition

NP is the class of problems A of the following form:

> x is a yes-instance of A if and only if there exists a w, such that (x, w) is is a yes-instance of B,

Certificate definition of **NP**

Definition

NP is the class of problems A of the following form:

> x is a yes-instance of A if and only if there exists a w, such that (x, w) is is a yes-instance of B,

where B is a decision problem in **P** regarding pairs (x, w) and $|w| = poly(|x|)$.

Certificate definition of **NP**

Definition

NP is the class of problems A of the following form:

> x is a yes-instance of A if and only if there exists a w, such that (x, w) is is a yes-instance of B,

where B is a decision problem in **P** regarding pairs (x, w) and $|w| = poly(|x|)$.

w is a witness of the fact that x is a yes-instance.

Certificate definition of **NP**

Definition

NP is the class of problems A of the following form:

> x is a yes-instance of A if and only if there exists a w, such that (x, w) is is a yes-instance of B,

where B is a decision problem in **P** regarding pairs (x, w) and $|w| = poly(|x|)$.

w is a witness of the fact that x is a yes-instance. It is called a *certificate*.

Certificate definition of **NP**

Definition

NP is the class of problems A of the following form:

> x is a yes-instance of A if and only if there exists a w, such that (x, w) is is a yes-instance of B,

where B is a decision problem in **P** regarding pairs (x, w) and $|w| = poly(|x|)$.

w is a witness of the fact that x is a yes-instance. It is called a *certificate*.

w is polynomially balanced.

Nondeterministic computation and **NP**

Nondeterministic computation and **NP**

Definition

Nondeterministic computation and **NP**

Definition

NP is the class of problems for which a nondeterministic program exists that runs in time $poly(n)$, on instances of length n,

Nondeterministic computation and **NP**

Definition

NP is the class of problems for which a nondeterministic program exists that runs in time $poly(n)$, on instances of length n,

such that the input is a yes-instance if and only if there exists a computation path that returns “yes.”

Reductions

Reductions

Definition

Reductions

Definition

A language L_1 is reducible to a language L_2 if there is a function R from strings of L_1 to strings of L_2, such that

$$(\forall x \in \Sigma_1^*) \;\; x \in L_1 \leftrightarrow R(x) \in L_2.$$

Reductions

Definition

A language L_1 is reducible to a language L_2 if there is a function R from strings of L_1 to strings of L_2, such that

$$(\forall x \in \Sigma_1^*) \;\; x \in L_1 \leftrightarrow R(x) \in L_2.$$

Furthermore, the function should be appropriately circumscribed

Reductions

Definition

A language L_1 is reducible to a language L_2 if there is a function R from strings of L_1 to strings of L_2, such that

$$(\forall x \in \Sigma_1^*) \;\; x \in L_1 \leftrightarrow R(x) \in L_2.$$

Furthermore, the function should be appropriately circumscribed (log space,

Reductions

Definition

A language L_1 is reducible to a language L_2 if there is a function R from strings of L_1 to strings of L_2, such that

$$(\forall x \in \Sigma_1^*) \;\; x \in L_1 \leftrightarrow R(x) \in L_2.$$

Furthermore, the function should be appropriately circumscribed (log space, polynomial time, etc.)

Reductions

Definition

A language L_1 is reducible to a language L_2 if there is a function R from strings of L_1 to strings of L_2, such that

$$(\forall x \in \Sigma_1^*) \;\; x \in L_1 \leftrightarrow R(x) \in L_2.$$

Furthermore, the function should be appropriately circumscribed (log space, polynomial time, etc.)

Definition

Reductions

Definition

A language L_1 is reducible to a language L_2 if there is a function R from strings of L_1 to strings of L_2, such that

$$(\forall x \in \Sigma_1^*) \;\; x \in L_1 \leftrightarrow R(x) \in L_2.$$

Furthermore, the function should be appropriately circumscribed (log space, polynomial time, etc.)

Definition

A polynomial-time reduction is a method of solving one problem

Reductions

Definition

A language L_1 is reducible to a language L_2 if there is a function R from strings of L_1 to strings of L_2, such that

$$(\forall x \in \Sigma_1^*) \;\; x \in L_1 \leftrightarrow R(x) \in L_2.$$

Furthermore, the function should be appropriately circumscribed (log space, polynomial time, etc.)

Definition

A polynomial-time reduction is a method of solving one problem by means of a hypothetical subroutine for solving a different problem,

Reductions

Definition

A language L_1 is reducible to a language L_2 if there is a function R from strings of L_1 to strings of L_2, such that

$$(\forall x \in \Sigma_1^*) \;\; x \in L_1 \leftrightarrow R(x) \in L_2.$$

Furthermore, the function should be appropriately circumscribed (log space, polynomial time, etc.)

Definition

A polynomial-time reduction is a method of solving one problem by means of a hypothetical subroutine for solving a different problem, that uses polynomial time excluding the time within the subroutine.

Karp Reductions

Karp Reductions

Definition

Karp Reductions

Definition

A polynomial-time many-one reduction (also called Karp reduction) from a problem A to a problem B (both of which are usually required to be decision problems)

Karp Reductions

Definition

A polynomial-time many-one reduction (also called Karp reduction) from a problem A to a problem B (both of which are usually required to be decision problems)

is a polynomial-time algorithm for transforming inputs to problem A into inputs to problem B,

Karp Reductions

Definition

A polynomial-time many-one reduction (also called Karp reduction) from a problem A to a problem B (both of which are usually required to be decision problems)

is a polynomial-time algorithm for transforming inputs to problem A into inputs to problem B, such that the transformed problem has the same output as the original problem.

Karp Reductions

Definition

A polynomial-time many-one reduction (also called Karp reduction) from a problem A to a problem B (both of which are usually required to be decision problems)

is a polynomial-time algorithm for transforming inputs to problem A into inputs to problem B, such that the transformed problem has the same output as the original problem.

Observation

Karp Reductions

Definition

A polynomial-time many-one reduction (also called Karp reduction) from a problem A to a problem B (both of which are usually required to be decision problems)

is a polynomial-time algorithm for transforming inputs to problem A into inputs to problem B, such that the transformed problem has the same output as the original problem.

Observation

1. *An instance x of problem A can be solved by applying this transformation to produce an instance y of problem B, giving y as the input to an algorithm for problem B, and returning its output.*

Karp Reductions

Definition

A polynomial-time many-one reduction (also called Karp reduction) from a problem A to a problem B (both of which are usually required to be decision problems)

is a polynomial-time algorithm for transforming inputs to problem A into inputs to problem B, such that the transformed problem has the same output as the original problem.

Observation

1. *An instance x of problem A can be solved by applying this transformation to produce an instance y of problem B, giving y as the input to an algorithm for problem B, and returning its output.*
2. *Polynomial-time many-one reductions are also be known as polynomial transformations or Karp reductions, named after Richard Karp. A reduction of this type may be denoted by the expression $A \leq_m^P B$.*

Turing Reductions

Turing Reductions

Definition

Turing Reductions

Definition

A polynomial-time Turing reduction from a problem A to a problem B is an algorithm that solves problem A using a polynomial number of calls to a subroutine for problem B, and polynomial time outside of those subroutine calls

Turing Reductions

Definition

A polynomial-time Turing reduction from a problem A to a problem B is an algorithm that solves problem A using a polynomial number of calls to a subroutine for problem B, and polynomial time outside of those subroutine calls

Observation

Turing Reductions

Definition

A polynomial-time Turing reduction from a problem A to a problem B is an algorithm that solves problem A using a polynomial number of calls to a subroutine for problem B, and polynomial time outside of those subroutine calls

Observation

Polynomial-time Turing reductions are also known as Cook reductions, named after Stephen Cook.

Turing Reductions

Definition

A polynomial-time Turing reduction from a problem A to a problem B is an algorithm that solves problem A using a polynomial number of calls to a subroutine for problem B, and polynomial time outside of those subroutine calls

Observation

Polynomial-time Turing reductions are also known as Cook reductions, named after Stephen Cook.

A reduction of this type may be denoted by the expression $A \leq_T^P B$.

Transitivity of Reductions

Transitivity of Reductions

Proposition

Transitivity of Reductions

Proposition

If $A \leq B$

Transitivity of Reductions

Proposition

If $A \leq B$ and $B \leq C$,

Transitivity of Reductions

Proposition

If $A \leq B$ and $B \leq C$, then $A \leq C$.

Transitivity of Reductions

Proposition

If $A \leq B$ and $B \leq C$, then $A \leq C$. It is understood that both reductions are of the same type.

NP-completeness

NP-completeness

Definition

NP-completeness

Definition

A problem A is said to be **NP-complete**, if

NP-completeness

Definition

A problem A is said to be **NP-complete**, if

NP-completeness

Definition

A problem A is said to be **NP-complete**, if

1. $A \in \mathbf{NP}$.

NP-completeness

Definition

A problem A is said to be **NP-complete**, if

1. $A \in \mathbf{NP}$.
2. $\forall B \in \mathbf{NP}$,

NP-completeness

Definition

A problem A is said to be **NP-complete**, if

1. $A \in \mathbf{NP}$.
2. $\forall B \in \mathbf{NP}$, $B \leq A$.

NP-completeness

Definition

A problem A is said to be **NP-complete**, if

1. $A \in \mathbf{NP}$.
2. $\forall B \in \mathbf{NP}$, $B \leq A$.

Observations

NP-completeness

Definition

A problem A is said to be **NP-complete**, if

1. $A \in \mathbf{NP}$.
2. $\forall B \in \mathbf{NP}$, $B \leq A$.

Observations

1. *If only the second condition is satisfied,*

NP-completeness

Definition

A problem A is said to be **NP-complete**, if

1. $A \in \mathbf{NP}$.
2. $\forall B \in \mathbf{NP}$, $B \leq A$.

Observations

1. *If only the second condition is satisfied, then the problem is said to be* **NP-hard**.

NP-completeness

Definition

A problem A is said to be **NP-complete**, if

1. $A \in \mathbf{NP}$.
2. $\forall B \in \mathbf{NP}$, $B \leq A$.

Observations

1. *If only the second condition is satisfied, then the problem is said to be* **NP-hard**.
2. *The reductions in question can be Karp or Turing, but we will use Karp for the rest of this chapter.*

Boolean Circuits (Syntax)

Boolean Circuits (Syntax)

Syntax

Boolean Circuits (Syntax)

Syntax

1. A boolean circuit C is a DAG $G = \langle V, E \rangle$.

Boolean Circuits (Syntax)

Syntax

1. A boolean circuit C is a DAG $G = \langle V, E \rangle$.
2. The nodes $V = \{1, 2, \ldots n\}$ are called the gates of C.

Boolean Circuits (Syntax)

Syntax

1. A boolean circuit C is a DAG $G = \langle V, E \rangle$.
2. The nodes $V = \{1, 2, \ldots n\}$ are called the gates of C.
3. We can assume without loss of generality that the edges are of the form (i, j), where $i < j$.

Boolean Circuits (Syntax)

Syntax

1. A boolean circuit C is a DAG $G = \langle V, E \rangle$.
2. The nodes $V = \{1, 2, \ldots n\}$ are called the gates of C.
3. We can assume without loss of generality that the edges are of the form (i, j), where $i < j$.
4. Each gate i has a sort $s(i)$ associated with it, where $s(i) \in \{\textbf{true}, \textbf{false}\} \cup \{x_1, x_2, \ldots\} \cup \{\vee, \wedge, \neg\}$.

Boolean Circuits (Syntax)

Syntax

1. A boolean circuit C is a DAG $G = \langle V, E \rangle$.
2. The nodes $V = \{1, 2, \ldots n\}$ are called the gates of C.
3. We can assume without loss of generality that the edges are of the form (i, j), where $i < j$.
4. Each gate i has a sort $s(i)$ associated with it, where $s(i) \in \{\textbf{true}, \ \textbf{false}\} \cup \{x_1, x_2, \ldots\} \cup \{\vee, \wedge, \neg\}$.
5. If $s(i) \in \{\textbf{true}, \ \textbf{false}\} \cup \{x_1, x_2, \ldots\}$, then its in-degree is 0.

Boolean Circuits (Syntax)

Syntax

1. A boolean circuit C is a DAG $G = \langle V, E \rangle$.
2. The nodes $V = \{1, 2, \ldots n\}$ are called the gates of C.
3. We can assume without loss of generality that the edges are of the form (i, j), where $i < j$.
4. Each gate i has a sort $s(i)$ associated with it, where $s(i) \in \{\mathbf{true}, \ \mathbf{false}\} \cup \{x_1, x_2, \ldots\} \cup \{\vee, \wedge, \neg\}$.
5. If $s(i) \in \{\mathbf{true}, \ \mathbf{false}\} \cup \{x_1, x_2, \ldots\}$, then its in-degree is 0.
6. If $s(i) \in \{\neg\}$, its in-degree is 1.

Boolean Circuits (Syntax)

Syntax

1. A boolean circuit C is a DAG $G = \langle V, E \rangle$.
2. The nodes $V = \{1, 2, \ldots n\}$ are called the gates of C.
3. We can assume without loss of generality that the edges are of the form (i, j), where $i < j$.
4. Each gate i has a sort $s(i)$ associated with it, where $s(i) \in \{\textbf{true}, \ \textbf{false}\} \cup \{x_1, x_2, \ldots\} \cup \{\vee, \wedge, \neg\}$.
5. If $s(i) \in \{\textbf{true}, \ \textbf{false}\} \cup \{x_1, x_2, \ldots\}$, then its in-degree is 0.
6. If $s(i) \in \{\neg\}$, its in-degree is 1.
7. All other gates have in-degree 2.

Boolean Circuits (Syntax)

Syntax

1. A boolean circuit C is a DAG $G = \langle V, E \rangle$.
2. The nodes $V = \{1, 2, \ldots n\}$ are called the gates of C.
3. We can assume without loss of generality that the edges are of the form (i, j), where $i < j$.
4. Each gate i has a sort $s(i)$ associated with it, where $s(i) \in \{\textbf{true}, \textbf{false}\} \cup \{x_1, x_2, \ldots\} \cup \{\vee, \wedge, \neg\}$.
5. If $s(i) \in \{\textbf{true}, \textbf{false}\} \cup \{x_1, x_2, \ldots\}$, then its in-degree is 0.
6. If $s(i) \in \{\neg\}$, its in-degree is 1.
7. All other gates have in-degree 2.
8. All gates except gate n have out-degree 1.

Boolean Circuits (Syntax)

Syntax

1. A boolean circuit C is a DAG $G = \langle V, E \rangle$.
2. The nodes $V = \{1, 2, \dots n\}$ are called the gates of C.
3. We can assume without loss of generality that the edges are of the form (i, j), where $i < j$.
4. Each gate i has a sort $s(i)$ associated with it, where $s(i) \in \{\textbf{true}, \textbf{false}\} \cup \{x_1, x_2, \dots\} \cup \{\vee, \wedge, \neg\}$.
5. If $s(i) \in \{\textbf{true}, \textbf{false}\} \cup \{x_1, x_2, \dots\}$, then its in-degree is 0.
6. If $s(i) \in \{\neg\}$, its in-degree is 1.
7. All other gates have in-degree 2.
8. All gates except gate n have out-degree 1.
9. Gate n, is called the output gate and has out-degree 0.

Boolean Circuits (Semantics)

Boolean Circuits (Semantics)

Semantics

Boolean Circuits (Semantics)

Semantics

The semantics of circuits specifies a truth value for the circuit, corresponding to each appropriate assignment.

Boolean Circuits (Semantics)

Semantics

The semantics of circuits specifies a truth value for the circuit, corresponding to each appropriate assignment.

This value can be computed inductively as follows:

Boolean Circuits (Semantics)

Semantics

The semantics of circuits specifies a truth value for the circuit, corresponding to each appropriate assignment.

This value can be computed inductively as follows:

1. If the gate is **true** or **false**, then it retains that value.

Boolean Circuits (Semantics)

Semantics

The semantics of circuits specifies a truth value for the circuit, corresponding to each appropriate assignment.

This value can be computed inductively as follows:

1. If the gate is **true** or **false**, then it retains that value.
2. If the gate is a variable, then its value is equal to its assignment.

Boolean Circuits (Semantics)

Semantics

The semantics of circuits specifies a truth value for the circuit, corresponding to each appropriate assignment.

This value can be computed inductively as follows:

1. If the gate is **true** or **false**, then it retains that value.
2. If the gate is a variable, then its value is equal to its assignment.
3. If the gate has sort $\neg$, then its value is the complement of its input.

Boolean Circuits (Semantics)

Semantics

The semantics of circuits specifies a truth value for the circuit, corresponding to each appropriate assignment.

This value can be computed inductively as follows:

1. If the gate is **true** or **false**, then it retains that value.
2. If the gate is a variable, then its value is equal to its assignment.
3. If the gate has sort $\neg$, then its value is the complement of its input.
4. If the gate has sort $\vee$, then its value is **true** if at least one of its two input gates has value **true** and is **false** otherwise.

Boolean Circuits (Semantics)

Semantics

The semantics of circuits specifies a truth value for the circuit, corresponding to each appropriate assignment.

This value can be computed inductively as follows:

1. If the gate is **true** or **false**, then it retains that value.
2. If the gate is a variable, then its value is equal to its assignment.
3. If the gate has sort $\neg$, then its value is the complement of its input.
4. If the gate has sort $\vee$, then its value is **true** if at least one of its two input gates has value **true** and is **false** otherwise.
5. If the gate has sort $\wedge$, then its value is **true** if both its two input gates have value **true** and is **false** otherwise.

Boolean Circuits (Semantics)

Semantics

The semantics of circuits specifies a truth value for the circuit, corresponding to each appropriate assignment.

This value can be computed inductively as follows:

1. If the gate is **true** or **false**, then it retains that value.
2. If the gate is a variable, then its value is equal to its assignment.
3. If the gate has sort $\neg$, then its value is the complement of its input.
4. If the gate has sort $\vee$, then its value is **true** if at least one of its two input gates has value **true** and is **false** otherwise.
5. If the gate has sort $\wedge$, then its value is **true** if both its two input gates have value **true** and is **false** otherwise.
6. The value of the circuit is the value of the output gate.

Size and Depth

Size and Depth

Definition

Size and Depth

Definition

The size of a boolean circuit is the number of gates in that circuit.

Size and Depth

Definition

The size of a boolean circuit is the number of gates in that circuit.

Definition

Size and Depth

Definition

The size of a boolean circuit is the number of gates in that circuit.

Definition

The depth of a circuit is the maximum distance from an input gate to the output gate.

String acceptance

String acceptance

Definition

String acceptance

Definition

Consider an n-input boolean circuit.

String acceptance

Definition

Consider an n-input boolean circuit. We say that a string x, with $|x| = n$ and $x_i \in \{0, 1\}$ is accepted by a circuit,

String acceptance

Definition

Consider an n-input boolean circuit. We say that a string x, with $|x| = n$ and $x_i \in \{0, 1\}$ is accepted by a circuit, if the output of the circuit is **true**, when presented with this string.

String acceptance

Definition

Consider an n-input boolean circuit. We say that a string x, with $|x| = n$ and $x_i \in \{0, 1\}$ is accepted by a circuit, if the output of the circuit is **true**, when presented with this string.

The i^{th} input is **true** if and only if $x_i = 1$.

Language Acceptance

Language Acceptance

Observations

Language Acceptance

Observations

1. *The above definition holds only for fixed n.*

Language Acceptance

Observations

1. *The above definition holds only for fixed n.*
2. *We can generalize the definition to strings of arbitrary length.*

Circuit Families

Circuit Families

Definition

Circuit Families

Definition

A family of circuits is an infinite sequence $\mathcal{C} = (C_0, C_1, \ldots)$ of Boolean circuits, where C_n has n input variables.

Circuit Families

Definition

A family of circuits is an infinite sequence $\mathcal{C} = (C_0, C_1, \ldots)$ of Boolean circuits, where C_n has n input variables.

Definition

Circuit Families

Definition

A family of circuits is an infinite sequence $\mathcal{C} = (C_0, C_1, \dots)$ of Boolean circuits, where C_n has n input variables.

Definition

A language $L \subseteq \{0,1\}^*$ has polynomial circuits, if there is a family of circuits $\mathcal{C} = (C_0, C_1, \dots)$ such that:

Circuit Families

Definition

A family of circuits is an infinite sequence $\mathcal{C} = (C_0, C_1, \ldots)$ of Boolean circuits, where C_n has n input variables.

Definition

A language $L \subseteq \{0,1\}^*$ has polynomial circuits, if there is a family of circuits $\mathcal{C} = (C_0, C_1, \ldots)$ such that:

1. The size of C_n is at most $p(n)$, for some fixed polynomial $p(n)$.

Circuit Families

Definition

A family of circuits is an infinite sequence $\mathcal{C} = (C_0, C_1, \ldots)$ of Boolean circuits, where C_n has n input variables.

Definition

A language $L \subseteq \{0,1\}^*$ has polynomial circuits, if there is a family of circuits $\mathcal{C} = (C_0, C_1, \ldots)$ such that:

1. The size of C_n is at most $p(n)$, for some fixed polynomial $p(n)$.
2. $\forall x \in \{0,1\}^*$, $x \in L$ if and only if, the output of $C_{|x|}$ is **true**, when the i^{th} input variable is **true** if $x_i = 1$ and **false** otherwise.

Uniform circuit families

Uniform circuit families

Definition

Uniform circuit families

Definition

A family of circuits $\mathcal{C} = (C_0, C_1, \ldots)$ is said to be *uniform* if there is log-space bounded algorithm which on input 1^n outputs C_n.

Uniform circuit families

Definition

A family of circuits $\mathcal{C} = (C_0, C_1, \ldots)$ is said to be *uniform* if there is log-space bounded algorithm which on input 1^n outputs C_n.

Definition

Uniform circuit families

Definition

A family of circuits $\mathcal{C} = (C_0, C_1, \ldots)$ is said to be *uniform* if there is log-space bounded algorithm which on input 1^n outputs C_n.

Definition

A language L has uniformly polynomial circuits if there is a uniform family of circuits that decides it.

P and uniform circuit families

P and uniform circuit families

Theorem

P and uniform circuit families

Theorem

A language L is in **P** *if and only if it has uniformly polynomial circuits.*

The first **NP-complete** problem

The first **NP-complete** problem

Motivation

The first **NP-complete** problem

Motivation

1. How many languages are there in **NP**?

The first **NP-complete** problem

Motivation

1. How many languages are there in **NP**?
2. The task of proving a language to be **NP-complete** is formidable,

The first **NP-complete** problem

Motivation

1. How many languages are there in **NP**?
2. The task of proving a language to be **NP-complete** is formidable, because we have to show that every language in **NP** reduces to the language in question.

The first **NP-complete** problem

Motivation

1. How many languages are there in **NP**?
2. The task of proving a language to be **NP-complete** is formidable, because we have to show that every language in **NP** reduces to the language in question.
3. However, once we have shown a language L to be **NP-complete**, we can show all other languages to be **NP-complete**, by reducing L to these languages!

The first **NP-complete** problem

Motivation

1. How many languages are there in **NP**?
2. The task of proving a language to be **NP-complete** is formidable, because we have to show that every language in **NP** reduces to the language in question.
3. However, once we have shown a language L to be **NP-complete**, we can show all other languages to be **NP-complete**, by reducing L to these languages!
4. So which language (or problem) is the first **NP-complete** language (problem)?

CircuitSAT

CircuitSAT

Theorem

CircuitSAT is **NP-complete***.*

CircuitSAT

Theorem

CircuitSAT is **NP-complete**.

Proof

CircuitSAT

Theorem

CircuitSAT is **NP-complete***.*

Proof

CircuitSAT is clearly in **NP**.

CircuitSAT

Theorem

CircuitSAT is **NP-complete***.*

Proof

CircuitSAT is clearly in **NP**.

CircuitSAT

Theorem

CircuitSAT is **NP-complete***.*

Proof

CircuitSAT is clearly in **NP**.

1. Let A be any language in **NP**.

CircuitSAT

Theorem

CircuitSAT is **NP-complete***.*

Proof

CircuitSAT is clearly in **NP**.

1. Let A be any language in **NP**.
2. A must have a polynomial time verifier V, such that $x \in A$ if and only if V accepts $\langle x, y \rangle$ for some polynomially balanced y.

CircuitSAT

Theorem

CircuitSAT is **NP-complete**.

Proof

CircuitSAT is clearly in **NP**.

1. Let A be any language in **NP**.
2. A must have a polynomial time verifier V, such that $x \in A$ if and only if V accepts $\langle x, y \rangle$ for some polynomially balanced y.
3. Since V runs in polynomial time, we know that there exists a uniform family of polynomial size circuits $\mathcal{C}$ that decides the language decided by V;

CircuitSAT

Theorem

CircuitSAT is **NP-complete***.*

Proof

CircuitSAT is clearly in **NP**.

1. Let A be any language in **NP**.
2. A must have a polynomial time verifier V, such that $x \in A$ if and only if V accepts $\langle x, y \rangle$ for some polynomially balanced y.
3. Since V runs in polynomial time, we know that there exists a uniform family of polynomial size circuits $\mathcal{C}$ that decides the language decided by V; i.e., $\mathcal{C}$ is equivalent to V.

CircuitSAT

Theorem

CircuitSAT is **NP-complete***.*

Proof

CircuitSAT is clearly in **NP**.

1. Let A be any language in **NP**.
2. A must have a polynomial time verifier V, such that $x \in A$ if and only if V accepts $\langle x, y \rangle$ for some polynomially balanced y.
3. Since V runs in polynomial time, we know that there exists a uniform family of polynomial size circuits $\mathcal{C}$ that decides the language decided by V; i.e., $\mathcal{C}$ is equivalent to V.
4. The input of $\mathcal{C}$ is $\langle x, y \rangle$ and a specific $C \in \mathcal{C}$ can be constructed in time polynomial in $|x|$ and $|y|$.

Completing the reduction

Completing the reduction

Proof (contd.)

Completing the reduction

Proof (contd.)

The reduction from A to C is as follows:

Completing the reduction

Proof (contd.)

The reduction from A to C is as follows:

1. Given an input x, output a description of the circuit $C(x, y)$, with the x values set to the given values and the y values left as variables.

Completing the reduction

Proof (contd.)

The reduction from A to C is as follows:

1. Given an input x, output a description of the circuit $C(x, y)$, with the x values set to the given values and the y values left as variables.
2. The resulting circuit is satisfiable if and only if $x \in A$.

Completing the reduction

Proof (contd.)

The reduction from A to C is as follows:

1. Given an input x, output a description of the circuit $C(x, y)$, with the x values set to the given values and the y values left as variables.
2. The resulting circuit is satisfiable if and only if $x \in A$.
3. The reduction is clearly polynomial time, since $\mathcal{C}$ is uniform.

Witness Existence

Witness Existence

Definition

Witness Existence

Definition

Input: A program $P(x, w)$, an input x and an integer t given in unary.

Witness Existence

Definition

Input: A program $P(x, w)$, an input x and an integer t given in unary.

Query: Does there exist a w,

Witness Existence

Definition

Input: A program $P(x, w)$, an input x and an integer t given in unary.

Query: Does there exist a w, with $|w| \leq t$, such that $P(x, w)$ returns "yes" after at most t steps?

Witness Existence

Definition

Input: A program $P(x, w)$, an input x and an integer t given in unary.

Query: Does there exist a w, with $|w| \leq t$, such that $P(x, w)$ returns "yes" after at most t steps?

Observations

Witness Existence

Definition

Input: A program $P(x, w)$, an input x and an integer t given in unary.

Query: Does there exist a w, with $|w| \leq t$, such that $P(x, w)$ returns “yes” after at most t steps?

Observations

1. *Why is the* WITNESS-EXISTENCE *problem* **NP-complete***?*

Witness Existence

Definition

Input: A program $P(x, w)$, an input x and an integer t given in unary.

Query: Does there exist a w, with $|w| \leq t$, such that $P(x, w)$ returns "yes" after at most t steps?

Observations

1. *Why is the* WITNESS-EXISTENCE *problem* **NP-complete***?*
2. *In the textbook, they reduce* WITNESS-EXISTENCE *to CircuitSAT.*

Witness Existence

Definition

Input: A program $P(x, w)$, an input x and an integer t given in unary.

Query: Does there exist a w, with $|w| \leq t$, such that $P(x, w)$ returns "yes" after at most t steps?

Observations

1. *Why is the* WITNESS-EXISTENCE *problem* **NP-complete**?
2. *In the textbook, they reduce* WITNESS-EXISTENCE *to CircuitSAT.*
3. *In his seminal* 1971 *paper, Cook reduced the* WITNESS-EXISTENCE *problem directly to SAT.*

Witness Existence

Definition

Input: A program $P(x, w)$, an input x and an integer t given in unary.

Query: Does there exist a w, with $|w| \leq t$, such that $P(x, w)$ returns “yes” after at most t steps?

Observations

1. *Why is the* WITNESS-EXISTENCE *problem* **NP-complete***?*
2. *In the textbook, they reduce* WITNESS-EXISTENCE *to CircuitSAT.*
3. *In his seminal* 1971 *paper, Cook reduced the* WITNESS-EXISTENCE *problem directly to SAT.* `https://www.cs.toronto.edu/~sacook/homepage/1971.pdf`.

Satisfiability (SAT)

Satisfiability (SAT)

Definition

Satisfiability (SAT)

Definition

Input: A boolean formula ϕ in CNF form over n variables and m clauses,

Satisfiability (SAT)

Definition

Input: A boolean formula ϕ in CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \dots C_m$.

Satisfiability (SAT)

Definition

Input: A boolean formula ϕ in CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$.

Query: Is ϕ satisfiable?

Satisfiability (SAT)

Definition

Input: A boolean formula ϕ in CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$.

Query: Is ϕ satisfiable?

Theorem

Satisfiability (SAT)

Definition

Input: A boolean formula ϕ in CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$.

Query: Is ϕ satisfiable?

Theorem

SAT is **NP-complete***.*

Satisfiability (SAT)

Definition

Input: A boolean formula ϕ in CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$.

Query: Is ϕ satisfiable?

Theorem

SAT is **NP-complete***.*

Proof

Satisfiability (SAT)

Definition

Input: A boolean formula ϕ in CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$.

Query: Is ϕ satisfiable?

Theorem

SAT is **NP-complete***.*

Proof

SAT is clearly in **NP**.

Satisfiability (SAT)

Definition

Input: A boolean formula ϕ in CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$.

Query: Is ϕ satisfiable?

Theorem

SAT is **NP-complete***.*

Proof

SAT is clearly in **NP**. (Why?)

Satisfiability (SAT)

Definition

Input: A boolean formula ϕ in CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$.

Query: Is ϕ satisfiable?

Theorem

SAT is **NP-complete***.*

Proof

SAT is clearly in **NP**. (Why?)

Clearly, CircuitSAT $\leq$ SAT

Satisfiability (SAT)

Definition

Input: A boolean formula ϕ in CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$.

Query: Is ϕ satisfiable?

Theorem

SAT is **NP-complete***.*

Proof

SAT is clearly in **NP**. (Why?)

Clearly, CircuitSAT $\leq$ SAT (Previous chapter).

3SAT

3SAT

Definition

3SAT

Definition

Input: A boolean formula ϕ in 3CNF form over n variables and m clauses,

3SAT

Definition

Input: A boolean formula ϕ in 3CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \dots C_m$, with each clause having *exactly* 3 literals.

3SAT

Definition

Input: A boolean formula ϕ in 3CNF form over *n* variables and *m* clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$, with each clause having *exactly* 3 literals.

Query: Is ϕ satisfiable?

3SAT

Definition

Input: A boolean formula ϕ in 3CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$, with each clause having *exactly* 3 literals.

Query: Is ϕ satisfiable?

Observations

3SAT

Definition

Input: A boolean formula ϕ in 3CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$, with each clause having *exactly* 3 literals.

Query: Is ϕ satisfiable?

Observations

1. *3SAT is clearly in* **NP**.

3SAT

Definition

Input: A boolean formula ϕ in 3CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \dots C_m$, with each clause having *exactly* 3 literals.

Query: Is ϕ satisfiable?

Observations

1. *3SAT is clearly in* **NP**.
2. *Consider a clause in 1CNF form.*

3SAT

Definition

Input: A boolean formula ϕ in 3CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$, with each clause having *exactly* 3 literals.

Query: Is ϕ satisfiable?

Observations

1. *3SAT is clearly in* **NP**.
2. *Consider a clause in 1CNF form. Can you represent it using 3CNF form?*

3SAT

Definition

Input: A boolean formula ϕ in 3CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$, with each clause having *exactly* 3 literals.

Query: Is ϕ satisfiable?

Observations

1. *3SAT is clearly in* **NP**.
2. *Consider a clause in 1CNF form. Can you represent it using 3CNF form?*
3. *Consider a clause in 2CNF form.*

3SAT

Definition

Input: A boolean formula ϕ in 3CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$, with each clause having *exactly* 3 literals.

Query: Is ϕ satisfiable?

Observations

1. *3SAT is clearly in* **NP**.
2. *Consider a clause in 1CNF form. Can you represent it using 3CNF form?*
3. *Consider a clause in 2CNF form. Can you represent it using 3CNF form?*

3SAT

Definition

Input: A boolean formula ϕ in 3CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$, with each clause having *exactly* 3 literals.

Query: Is ϕ satisfiable?

Observations

1. *3SAT is clearly in* **NP***.*
2. *Consider a clause in 1CNF form. Can you represent it using 3CNF form?*
3. *Consider a clause in 2CNF form. Can you represent it using 3CNF form?*
4. *Consider a clause in 4CNF form.*

3SAT

Definition

Input: A boolean formula ϕ in 3CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$, with each clause having *exactly* 3 literals.

Query: Is ϕ satisfiable?

Observations

1. *3SAT is clearly in* **NP**.
2. *Consider a clause in 1CNF form. Can you represent it using 3CNF form?*
3. *Consider a clause in 2CNF form. Can you represent it using 3CNF form?*
4. *Consider a clause in 4CNF form. Can you represent it using 3CNF form?*

3SAT

Definition

Input: A boolean formula ϕ in 3CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$, with each clause having *exactly* 3 literals.

Query: Is ϕ satisfiable?

Observations

1. *3SAT is clearly in* **NP***.*
2. *Consider a clause in 1CNF form. Can you represent it using 3CNF form?*
3. *Consider a clause in 2CNF form. Can you represent it using 3CNF form?*
4. *Consider a clause in 4CNF form. Can you represent it using 3CNF form?*
5. *Generalize . . . !*

3SAT

Definition

Input: A boolean formula ϕ in 3CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$, with each clause having *exactly* 3 literals.

Query: Is ϕ satisfiable?

Observations

1. *3SAT is clearly in* **NP**.
2. *Consider a clause in 1CNF form. Can you represent it using 3CNF form?*
3. *Consider a clause in 2CNF form. Can you represent it using 3CNF form?*
4. *Consider a clause in 4CNF form. Can you represent it using 3CNF form?*
5. *Generalize . . .!*
6. *3SAT is the most versatile of SAT problems.*

NAESAT

NAESAT

Definition

NAESAT

Definition

An assignment to a boolean formula is nae-satisfying, if

NAESAT

Definition

An assignment to a boolean formula is nae-satisfying, if

NAESAT

Definition

An assignment to a boolean formula is nae-satisfying, if

1. It satisfies at least one literal in each clause.

NAESAT

Definition

An assignment to a boolean formula is nae-satisfying, if

1. It satisfies at least one literal in each clause.
2. It falsifies at least one literal in each clause.

NAESAT

Definition

An assignment to a boolean formula is nae-satisfying, if

1. It satisfies at least one literal in each clause.
2. It falsifies at least one literal in each clause.

Definition

NAESAT

Definition

An assignment to a boolean formula is nae-satisfying, if

1. It satisfies at least one literal in each clause.
2. It falsifies at least one literal in each clause.

Definition

Input: A boolean formula ϕ in CNF form over *n* variables and *m* clauses,

NAESAT

Definition

An assignment to a boolean formula is nae-satisfying, if

1. It satisfies at least one literal in each clause.
2. It falsifies at least one literal in each clause.

Definition

Input: A boolean formula ϕ in CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$.

NAESAT

Definition

An assignment to a boolean formula is nae-satisfying, if

1. It satisfies at least one literal in each clause.
2. It falsifies at least one literal in each clause.

Definition

Input: A boolean formula ϕ in CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$.

Query: Is ϕ nae-satisfiable?

NAESAT

Definition

An assignment to a boolean formula is nae-satisfying, if

1. It satisfies at least one literal in each clause.
2. It falsifies at least one literal in each clause.

Definition

Input: A boolean formula ϕ in CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$.

Query: Is ϕ nae-satisfiable?

Reduction

NAESAT

Definition

An assignment to a boolean formula is nae-satisfying, if

1. It satisfies at least one literal in each clause.
2. It falsifies at least one literal in each clause.

Definition

Input: A boolean formula ϕ in CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \dots C_m$.

Query: Is ϕ nae-satisfiable?

Reduction

1. Construct a new formula ϕ' by adding a new variable s to every single clause.

NAESAT

Definition

An assignment to a boolean formula is nae-satisfying, if

1. It satisfies at least one literal in each clause.
2. It falsifies at least one literal in each clause.

Definition

Input: A boolean formula ϕ in CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$.

Query: Is ϕ nae-satisfiable?

Reduction

1. Construct a new formula ϕ' by adding a new variable s to every single clause.
2. If ϕ is satisfiable, then ϕ' is nae-satisfiable.

NAESAT

Definition

An assignment to a boolean formula is nae-satisfying, if

1. It satisfies at least one literal in each clause.
2. It falsifies at least one literal in each clause.

Definition

Input: A boolean formula ϕ in CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$.

Query: Is ϕ nae-satisfiable?

Reduction

1. Construct a new formula ϕ' by adding a new variable s to every single clause.
2. If ϕ is satisfiable, then ϕ' is nae-satisfiable.
3. If ϕ' is nae-satisfiable, then ϕ **must** be satisfiable. (Why?)

NAESAT

Definition

An assignment to a boolean formula is nae-satisfying, if

1. It satisfies at least one literal in each clause.
2. It falsifies at least one literal in each clause.

Definition

Input: A boolean formula ϕ in CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$.

Query: Is ϕ nae-satisfiable?

Reduction

1. Construct a new formula ϕ' by adding a new variable s to every single clause.
2. If ϕ is satisfiable, then ϕ' is nae-satisfiable.
3. If ϕ' is nae-satisfiable, then ϕ **must** be satisfiable. (Why?)
4. Thus, SAT $\leq$ NAESAT.

NAE3SAT

NAE3SAT

Reduction

NAE3SAT

Reduction

1. Using the technique, we can show that NAE4SAT is **NP-complete**. Why?

NAE3SAT

Reduction

1. Using the technique, we can show that NAE4SAT is **NP-complete**. Why?
2. To show that NAE3SAT is **NP-complete**, we simply reduce NAE4SAT to it!

NAE3SAT

Reduction

1. Using the technique, we can show that NAE4SAT is **NP-complete**. Why?
2. To show that NAE3SAT is **NP-complete**, we simply reduce NAE4SAT to it!
3. Consider a 4CNF clause $I = (x, y, z, w)$. Argue that I is nae-satisfiable if and only if the following pair of clauses are:

NAE3SAT

Reduction

1. Using the technique, we can show that NAE4SAT is **NP-complete**. Why?
2. To show that NAE3SAT is **NP-complete**, we simply reduce NAE4SAT to it!
3. Consider a 4CNF clause $I = (x, y, z, w)$. Argue that I is nae-satisfiable if and only if the following pair of clauses are:

$$(x, y, s)$$
$$(z, w, \bar{s})$$

NAE3SAT

Reduction

1. Using the technique, we can show that NAE4SAT is **NP-complete**. Why?
2. To show that NAE3SAT is **NP-complete**, we simply reduce NAE4SAT to it!
3. Consider a 4CNF clause $I = (x, y, z, w)$. Argue that I is nae-satisfiable if and only if the following pair of clauses are:

$$(x, y, s)$$
$$(z, w, \bar{s})$$

4. It follows that NAE3SAT is **NP-complete**, since 3SAT $\leq$ NAE4SAT $\leq$ NAE3SAT.

MaxSAT

MaxSAT

Definition

MaxSAT

Definition

Input: A boolean formula ϕ in CNF form over n variables and m clauses,

MaxSAT

Definition

Input: A boolean formula ϕ in CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$ and a number $K \leq m$.

MaxSAT

Definition

Input: A boolean formula ϕ in CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \dots C_m$ and a number $K \leq m$.

Query: Is there a subset of K or more clauses of ϕ which is satisfiable?

MaxSAT

Definition

Input: A boolean formula ϕ in CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$ and a number $K \leq m$.

Query: Is there a subset of K or more clauses of ϕ which is satisfiable?

Observations

MaxSAT

Definition

Input: A boolean formula ϕ in CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$ and a number $K \leq m$.

Query: Is there a subset of K or more clauses of ϕ which is satisfiable?

Observations

1. *MaxSAT is trivially* **NP-complete**. *(Why?)*

MaxSAT

Definition

Input: A boolean formula ϕ in CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$ and a number $K \leq m$.

Query: Is there a subset of K or more clauses of ϕ which is satisfiable?

Observations

1. *MaxSAT is trivially* **NP-complete**. *(Why?)*
2. *In general, if kSAT is* **NP-complete**, *so is Max$k$SAT.*

MaxSAT

Definition

Input: A boolean formula ϕ in CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$ and a number $K \leq m$.

Query: Is there a subset of K or more clauses of ϕ which is satisfiable?

Observations

1. *MaxSAT is trivially* **NP-complete***. (Why?)*
2. *In general, if kSAT is* **NP-complete***, so is Max$k$SAT.*
3. *How about Max2SAT?*

MaxSAT

Definition

Input: A boolean formula ϕ in CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$ and a number $K \leq m$.

Query: Is there a subset of K or more clauses of ϕ which is satisfiable?

Observations

1. *MaxSAT is trivially* **NP-complete**. *(Why?)*
2. *In general, if* k*SAT is* **NP-complete**, *so is Max*k*SAT.*
3. *How about Max2SAT?*
4. *We will show that NAE3SAT* $\leq$ *Max2SAT.*

Max2SAT

Max2SAT

Definition

Max2SAT

Definition

Input: A boolean formula ϕ in 2CNF form over *n* variables and *m* clauses,

Max2SAT

Definition

Input: A boolean formula ϕ in 2CNF form over *n* variables and *m* clauses, i.e., $\phi = C_1 \wedge C_2 \dots C_m$, with each clause having *exactly* 2 literals and a number $K \leq m$.

Max2SAT

Definition

Input: A boolean formula ϕ in 2CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$, with each clause having *exactly* 2 literals and a number $K \leq m$.

Query: Is there a subset of ϕ with cardinality at least K, which is satisfiable?

Max2SAT

Definition

Input: A boolean formula ϕ in 2CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$, with each clause having *exactly* 2 literals and a number $K \leq m$.

Query: Is there a subset of ϕ with cardinality at least K, which is satisfiable?

Reduction

Max2SAT

Definition

Input: A boolean formula ϕ in 2CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$, with each clause having *exactly* 2 literals and a number $K \leq m$.

Query: Is there a subset of ϕ with cardinality at least K, which is satisfiable?

Reduction

1. Assume that you are given an instance of NAE3SAT over n variables and m clauses.

Max2SAT

Definition

Input: A boolean formula ϕ in 2CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$, with each clause having *exactly* 2 literals and a number $K \leq m$.

Query: Is there a subset of ϕ with cardinality at least K, which is satisfiable?

Reduction

1. Assume that you are given an instance of NAE3SAT over n variables and m clauses.
2. Consider the clause $l = (x, y, z)$ of the NAE3SAT instance. Replace it with the following set:

Max2SAT

Definition

Input: A boolean formula ϕ in 2CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$, with each clause having *exactly* 2 literals and a number $K \leq m$.

Query: Is there a subset of ϕ with cardinality at least K, which is satisfiable?

Reduction

1. Assume that you are given an instance of NAE3SAT over n variables and m clauses.
2. Consider the clause $l = (x, y, z)$ of the NAE3SAT instance. Replace it with the following set:

$$\begin{array}{ccc} (x, y) & (y, z) & (x, z) \\ (\bar{x}, \bar{y}) & (\bar{y}, \bar{z}) & (\bar{x}, \bar{z}) \end{array}$$

Max2SAT

Definition

Input: A boolean formula ϕ in 2CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$, with each clause having *exactly* 2 literals and a number $K \leq m$.

Query: Is there a subset of ϕ with cardinality at least K, which is satisfiable?

Reduction

1. Assume that you are given an instance of NAE3SAT over n variables and m clauses.
2. Consider the clause $l = (x, y, z)$ of the NAE3SAT instance. Replace it with the following set:

$$\begin{array}{ccc} (x, y) & (y, z) & (x, z) \\ (\bar{x}, \bar{y}) & (\bar{y}, \bar{z}) & (\bar{x}, \bar{z}) \end{array}$$

3. Set $K = 5 \cdot m$.

Max2SAT

Definition

Input: A boolean formula ϕ in 2CNF form over n variables and m clauses, i.e., $\phi = C_1 \wedge C_2 \ldots C_m$, with each clause having *exactly* 2 literals and a number $K \leq m$.

Query: Is there a subset of ϕ with cardinality at least K, which is satisfiable?

Reduction

1. Assume that you are given an instance of NAE3SAT over n variables and m clauses.
2. Consider the clause $I = (x, y, z)$ of the NAE3SAT instance. Replace it with the following set:

$$\begin{array}{ccc} (x, y) & (y, z) & (x, z) \\ (\bar{x}, \bar{y}) & (\bar{y}, \bar{z}) & (\bar{x}, \bar{z}) \end{array}$$

3. Set $K = 5 \cdot m$.
4. In argument, note that any assignment satisfies 3 or 5 of the clause set, depending on whether or not it nae-satisfies I.

Integer Programming (IP)

Integer Programming (IP)

Definition

Integer Programming (IP)

Definition

Input: An integer matrix $\mathbf{A}_{m \times n}$ and an integer vector $\mathbf{b}_{m \times 1}$.

Integer Programming (IP)

Definition

Input: An integer matrix $\mathbf{A}_{m \times n}$ and an integer vector $\mathbf{b}_{m \times 1}$.

Query: Is there a lattice point $\mathbf{r} \in \mathcal{Z}_{+}^{n}$, such that $\mathbf{A} \cdot \mathbf{r} \geq \mathbf{b}$?

Integer Programming (IP)

Definition

Input: An integer matrix $\mathbf{A}_{m \times n}$ and an integer vector $\mathbf{b}_{m \times 1}$.

Query: Is there a lattice point $\mathbf{r} \in \mathcal{Z}_{+}^{n}$, such that $\mathbf{A} \cdot \mathbf{r} \geq \mathbf{b}$?

Observation

Integer Programming (IP)

Definition

Input: An integer matrix $\mathbf{A}_{m \times n}$ and an integer vector $\mathbf{b}_{m \times 1}$.

Query: Is there a lattice point $\mathbf{r} \in \mathcal{Z}_{+}^{n}$, such that $\mathbf{A} \cdot \mathbf{r} \geq \mathbf{b}$?

Observation

It is non-trivial to show that IP is in **NP**.

Integer Programming (IP)

Definition

Input: An integer matrix $\mathbf{A}_{m\times n}$ and an integer vector $\mathbf{b}_{m\times 1}$.

Query: Is there a lattice point $\mathbf{r} \in \mathcal{Z}_{+}^{n}$, such that $\mathbf{A} \cdot \mathbf{r} \geq \mathbf{b}$?

Observation

It is non-trivial to show that IP is in **NP***.*

Hence, we will focus on a restriction called $0/1$ *IP, where each component of the vector* $\mathbf{r}$ *is required to be* 0 *or* 1*.*

$0/1$ IP

0/1 IP

Theorem

0/1 IP

Theorem

0/1 IP is **NP-complete***.*

$0/1$ IP

Theorem

$0/1$ *IP is* **NP-complete**.

Proof

$0/1$ IP

Theorem

$0/1$ *IP is* **NP-complete**.

Proof

1. $0/1$ IP is clearly in **NP**.

$0/1$ IP

Theorem

$0/1$ *IP is* **NP-complete**.

Proof

1. $0/1$ IP is clearly in **NP**.
2. We will show that 3SAT $\leq 0/1$ IP.

$0/1$ IP

Theorem

$0/1$ *IP is* **NP-complete**.

Proof

1. $0/1$ IP is clearly in **NP**.
2. We will show that 3SAT $\leq 0/1$ IP.
3. Take the clause $I : (x, \bar{y}, z)$.

$0/1$ IP

Theorem

$0/1$ *IP is* **NP-complete**.

Proof

1. $0/1$ IP is clearly in **NP**.
2. We will show that 3SAT $\leq 0/1$ IP.
3. Take the clause $l : (x, \bar{y}, z)$.
4. Replace it with the constraint: $c : x + (1 - y) + z \geq 1$.

$0/1$ IP

Theorem

$0/1$ *IP is* **NP-complete**.

Proof

1. $0/1$ IP is clearly in **NP**.
2. We will show that 3SAT $\leq 0/1$ IP.
3. Take the clause $I : (x, \bar{y}, z)$.
4. Replace it with the constraint: $c : x + (1 - y) + z \geq 1$.
5. Argue that if I has a satisfying assignment then do does c

$0/1$ IP

Theorem

$0/1$ IP is **NP-complete***.*

Proof

1. $0/1$ IP is clearly in **NP**.
2. We will show that 3SAT $\leq 0/1$ IP.
3. Take the clause $l : (x, \bar{y}, z)$.
4. Replace it with the constraint: $c : x + (1 - y) + z \geq 1$.
5. Argue that if l has a satisfying assignment then do does c and vice versa.

$0/1$ IP

Theorem

$0/1$ *IP is* **NP-complete***.*

Proof

1. $0/1$ IP is clearly in **NP**.
2. We will show that 3SAT $\leq 0/1$ IP.
3. Take the clause $l : (x, \bar{y}, z)$.
4. Replace it with the constraint: $c : x + (1 - y) + z \geq 1$.
5. Argue that if l has a satisfying assignment then do does c and vice versa.
6. The theorem follows.

$0/1$ IP

Theorem

$0/1$ IP is **NP-complete***.*

Proof

1. $0/1$ IP is clearly in **NP**.
2. We will show that 3SAT $\leq 0/1$ IP.
3. Take the clause $I : (x, \bar{y}, z)$.
4. Replace it with the constraint: $c : x + (1 - y) + z \geq 1$.
5. Argue that if I has a satisfying assignment then do does c and vice versa.
6. The theorem follows.

Observations

Integer Programming rivals 3SAT in terms of versatility.