

Reductions and NP

Lecture 21

April 17, 2014

Part I

Reductions Continued

Polynomial Time Reduction

Karp reduction

A **polynomial time reduction** from a *decision* problem X to a *decision* problem Y is an *algorithm* \mathcal{A} that has the following properties:

- 1 given an instance I_X of X , \mathcal{A} produces an instance I_Y of Y
- 2 \mathcal{A} runs in time polynomial in $|I_X|$. This implies that $|I_Y|$ (size of I_Y) is polynomial in $|I_X|$
- 3 Answer to I_X YES *iff* answer to I_Y is YES.

Notation: $X \leq_P Y$ if X reduces to Y

Proposition

If $X \leq_P Y$ then a polynomial time algorithm for Y implies a polynomial time algorithm for X .

Such a reduction is called a **Karp reduction**. Most reductions we will need are Karp reductions.

A More General Reduction

Turing Reduction

Definition (Turing reduction.)

Problem X polynomial time reduces to Y if there is an algorithm \mathcal{A} for X that has the following properties:

- 1 on any given instance I_x of X , \mathcal{A} uses polynomial in $|I_x|$ “steps”
- 2 a step is either a standard computation step, or
- 3 a sub-routine call to an algorithm that solves Y .

This is a **Turing reduction**.

A More General Reduction

Turing Reduction

Definition (Turing reduction.)

Problem X polynomial time reduces to Y if there is an algorithm \mathcal{A} for X that has the following properties:

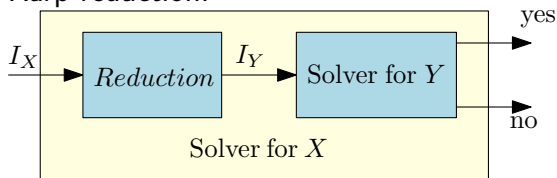
- 1 on any given instance I_x of X , \mathcal{A} uses polynomial in $|I_x|$ “steps”
- 2 a step is either a standard computation step, or
- 3 a sub-routine call to an algorithm that solves Y .

This is a **Turing reduction**.

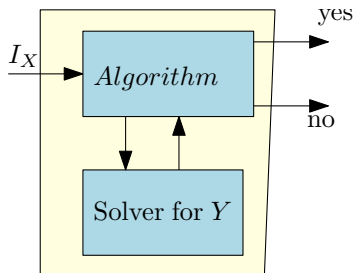
Note: In making sub-routine call to algorithm to solve Y , \mathcal{A} can only ask questions of size polynomial in $|I_x|$. Why?

Comparing reductions

1 Karp reduction:



2 Turing reduction:



Turing reduction

- 1 Algorithm to solve **X** can call solver for **Y** many times.
- 2 Conceptually, every call to the solver of **Y** takes constant time.

Relation between reductions

Consider two problems X and Y . Which of the following statements is correct?

- (A) If there is a Turing reduction from X to Y , then there is a Karp reduction from X to Y .
- (B) If there is a Karp reduction from X to Y , then there is a Turing reduction from X to Y .
- (C) If there is a Karp reduction from X to Y , then there is a Karp reduction from Y to X .
- (D) If there is a Turing reduction from X to Y , then there is a Turing reduction from Y to X .
- (E) All of the above.

Example of Turing Reduction

Problem (Independent set in circular arcs graph.)

Input: *Collection of arcs on a circle.*

Goal: *Compute the maximum number of non-overlapping arcs.*

Reduced to the following problem:?

Example of Turing Reduction

Problem (Independent set in circular arcs graph.)

Input: *Collection of arcs on a circle.*

Goal: *Compute the maximum number of non-overlapping arcs.*

Reduced to the following problem:?

Problem (Independent set of intervals.)

Input: *Collection of intervals on the line.*

Goal: *Compute the maximum number of non-overlapping intervals.*

How?

Example of Turing Reduction

Problem (Independent set in circular arcs graph.)

Input: *Collection of arcs on a circle.*

Goal: *Compute the maximum number of non-overlapping arcs.*

Reduced to the following problem:?

Problem (Independent set of intervals.)

Input: *Collection of intervals on the line.*

Goal: *Compute the maximum number of non-overlapping intervals.*

How? Used algorithm for interval problem multiple times.

Turing vs Karp Reductions

- 1 Turing reductions more general than Karp reductions.
- 2 Turing reduction useful in obtaining algorithms via reductions.
- 3 Karp reduction is simpler and easier to use to prove hardness of problems.
- 4 Perhaps surprisingly, Karp reductions, although limited, suffice for most known **NP-Completeness** proofs.
- 5 Karp reductions allow us to distinguish between **NP** and **co-NP** (more on this later).

Propositional Formulas

Definition

Consider a set of boolean variables x_1, x_2, \dots, x_n .

- 1 A **literal** is either a boolean variable x_i or its negation $\neg x_i$.
- 2 A **clause** is a disjunction of literals.
For example, $x_1 \vee x_2 \vee \neg x_4$ is a clause.
- 3 A **formula in conjunctive normal form (CNF)** is propositional formula which is a conjunction of clauses
 - 1 $(x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee \neg x_3) \wedge x_5$ is a **CNF** formula.

Propositional Formulas

Definition

Consider a set of boolean variables x_1, x_2, \dots, x_n .

- 1 A **literal** is either a boolean variable x_i or its negation $\neg x_i$.
- 2 A **clause** is a disjunction of literals.
For example, $x_1 \vee x_2 \vee \neg x_4$ is a clause.
- 3 A **formula in conjunctive normal form (CNF)** is propositional formula which is a conjunction of clauses
 - 1 $(x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee \neg x_3) \wedge x_5$ is a **CNF** formula.
- 4 A formula φ is a **3CNF**:
A **CNF** formula such that every clause has **exactly** 3 literals.
 - 1 $(x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee x_1)$ is a **3CNF** formula, but $(x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee \neg x_3) \wedge x_5$ is not.

Satisfiability

Problem: SAT

Instance: A CNF formula φ .

Question: Is there a truth assignment to the variables of φ such that φ evaluates to true?

Problem: 3SAT

Instance: A 3CNF formula φ .

Question: Is there a truth assignment to the variables of φ such that φ evaluates to true?

Satisfiability

SAT

Given a **CNF** formula φ , is there a truth assignment to variables such that φ evaluates to true?

Example

- 1 $(x_1 \vee x_2 \vee \neg x_4) \wedge (x_2 \vee \neg x_3) \wedge x_5$ is satisfiable; take x_1, x_2, \dots, x_5 to be all true
- 2 $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_1 \vee x_2)$ is not satisfiable.

3SAT

Given a **3CNF** formula φ , is there a truth assignment to variables such that φ evaluates to true?

(More on **2SAT** in a bit...)

Importance of **SAT** and **3SAT**

- 1 **SAT** and **3SAT** are basic constraint satisfaction problems.
- 2 Many different problems can be reduced to them because of the simple yet powerful expressiveness of logical constraints.
- 3 Arise naturally in many applications involving hardware and software verification and correctness.
- 4 As we will see, it is a fundamental problem in theory of **NP-Completeness**.

$$z = \bar{x}$$

Given two bits x, z which of the following **SAT** formulas is equivalent to the formula $z = \bar{x}$:

(A) $(\bar{z} \vee x) \wedge (z \vee \bar{x})$.

(B) $(z \vee x) \wedge (\bar{z} \vee \bar{x})$.

(C) $(\bar{z} \vee x) \wedge (\bar{z} \vee \bar{x}) \wedge (\bar{z} \vee \bar{x})$.

(D) $z \oplus x$.

(E) $(z \vee x) \wedge (\bar{z} \vee \bar{x}) \wedge (z \vee \bar{x}) \wedge (\bar{z} \vee x)$.

$$z = x \wedge y$$

Given three bits x, y, z which of the following **SAT** formulas is equivalent to the formula $z = x \wedge y$:

(A) $(\bar{z} \vee x \vee y) \wedge (z \vee \bar{x} \vee \bar{y})$.

(B) $(\bar{z} \vee x \vee y) \wedge (\bar{z} \vee \bar{x} \vee y) \wedge (z \vee \bar{x} \vee \bar{y})$.

(C) $(\bar{z} \vee x \vee y) \wedge (\bar{z} \vee \bar{x} \vee y) \wedge (z \vee \bar{x} \vee y) \wedge (z \vee \bar{x} \vee \bar{y})$.

(D) $(z \vee x \vee y) \wedge (\bar{z} \vee \bar{x} \vee y) \wedge (z \vee \bar{x} \vee y) \wedge (z \vee \bar{x} \vee \bar{y})$.

(E) $(z \vee x \vee y) \wedge (z \vee x \vee \bar{y}) \wedge (z \vee \bar{x} \vee y) \wedge (z \vee \bar{x} \vee \bar{y}) \wedge$
 $(\bar{z} \vee x \vee y) \wedge (\bar{z} \vee x \vee \bar{y}) \wedge (\bar{z} \vee \bar{x} \vee y) \wedge (\bar{z} \vee \bar{x} \vee \bar{y})$.

$$z = x \vee y$$

Given three bits x, y, z which of the following **SAT** formulas is equivalent to the formula $z = x \vee y$:

(A) $(\bar{z} \vee x \vee y) \wedge (\bar{z} \vee \bar{x} \vee y) \wedge (z \vee \bar{x} \vee \bar{y})$.

(B) $(\bar{z} \vee x \vee y) \wedge (\bar{z} \vee \bar{x} \vee y) \wedge (z \vee \bar{x} \vee y) \wedge (z \vee \bar{x} \vee \bar{y})$.

(C) $(z \vee x \vee y) \wedge (\bar{z} \vee \bar{x} \vee y) \wedge (z \vee \bar{x} \vee y) \wedge (z \vee \bar{x} \vee \bar{y})$.

(D) $(z \vee x \vee y) \wedge (z \vee x \vee \bar{y}) \wedge (z \vee \bar{x} \vee y) \wedge (z \vee \bar{x} \vee \bar{y}) \wedge$
 $(\bar{z} \vee x \vee y) \wedge (\bar{z} \vee x \vee \bar{y}) \wedge (\bar{z} \vee \bar{x} \vee y) \wedge (\bar{z} \vee \bar{x} \vee \bar{y})$.

(E) $(\bar{z} \vee x \vee y) \wedge (z \vee \bar{x} \vee y) \wedge (z \vee x \vee \bar{y}) \wedge (z \vee \bar{x} \vee \bar{y})$.

SAT \leq_p 3SAT

How **SAT** is different from **3SAT**?

In **SAT** clauses might have arbitrary length: **1, 2, 3, ...** variables:

$$(x \vee y \vee z \vee w \vee u) \wedge (\neg x \vee \neg y \vee \neg z \vee w \vee u) \wedge (\neg x)$$

In **3SAT** every clause must have **exactly 3** different literals.

SAT \leq_p 3SAT

How SAT is different from 3SAT?

In SAT clauses might have arbitrary length: 1, 2, 3, ... variables:

$$(x \vee y \vee z \vee w \vee u) \wedge (\neg x \vee \neg y \vee \neg z \vee w \vee u) \wedge (\neg x)$$

In 3SAT every clause must have **exactly 3** different literals.

To reduce from an instance of SAT to an instance of 3SAT, we must make all clauses to have exactly 3 variables...

Basic idea

- 1 Pad short clauses so they have 3 literals.
- 2 Break long clauses into shorter clauses.
- 3 Repeat the above till we have a 3CNF.

3SAT \leq_P SAT

- 1 3SAT \leq_P SAT.
- 2 Because...
A 3SAT instance is also an instance of SAT.

$SAT \leq_p 3SAT$

Claim

$SAT \leq_p 3SAT$.

SAT \leq_p 3SAT

Claim

SAT \leq_p 3SAT.

Given φ a SAT formula we create a 3SAT formula φ' such that

- 1 φ is satisfiable iff φ' is satisfiable.
- 2 φ' can be constructed from φ in time polynomial in $|\varphi|$.

SAT \leq_P 3SAT

Claim

SAT \leq_P 3SAT.

Given φ a SAT formula we create a 3SAT formula φ' such that

- 1 φ is satisfiable iff φ' is satisfiable.
- 2 φ' can be constructed from φ in time polynomial in $|\varphi|$.

Idea: if a clause of φ is not of length 3, replace it with several clauses of length exactly 3.

SAT \leq_p 3SAT

A clause with a single literal

Reduction Ideas

Challenge: Some of the clauses in φ may have less or more than **3** literals. For each clause with < 3 or > 3 literals, we will construct a set of logically equivalent clauses.

- 1 **Case clause with one literal:** Let c be a clause with a single literal (i.e., $c = \ell$). Let u, v be new variables. Consider

$$c' = (\ell \vee u \vee v) \wedge (\ell \vee u \vee \neg v) \\ \wedge (\ell \vee \neg u \vee v) \wedge (\ell \vee \neg u \vee \neg v).$$

Observe that c' is satisfiable iff c is satisfiable

SAT \leq_p 3SAT

A clause with two literals

Reduction Ideas: 2 and more literals

- 1 **Case clause with 2 literals:** Let $\mathbf{c} = \ell_1 \vee \ell_2$. Let \mathbf{u} be a new variable. Consider

$$\mathbf{c}' = (\ell_1 \vee \ell_2 \vee \mathbf{u}) \wedge (\ell_1 \vee \ell_2 \vee \neg \mathbf{u}).$$

Again \mathbf{c} is satisfiable iff \mathbf{c}' is satisfiable

Breaking a clause

Lemma

For any boolean formulas \mathbf{X} and \mathbf{Y} and \mathbf{z} a new boolean variable.
Then

$\mathbf{X} \vee \mathbf{Y}$ is satisfiable

if and only if, \mathbf{z} can be assigned a value such that

$(\mathbf{X} \vee \mathbf{z}) \wedge (\mathbf{Y} \vee \neg \mathbf{z})$ is satisfiable

(with the same assignment to the variables appearing in \mathbf{X} and \mathbf{Y}).

SAT \leq_p 3SAT (contd)

Clauses with more than 3 literals

Let $\mathbf{c} = \ell_1 \vee \dots \vee \ell_k$. Let $\mathbf{u}_1, \dots, \mathbf{u}_{k-3}$ be new variables. Consider

$$\begin{aligned} \mathbf{c}' = & \left(\ell_1 \vee \ell_2 \vee \mathbf{u}_1 \right) \wedge \left(\ell_3 \vee \neg \mathbf{u}_1 \vee \mathbf{u}_2 \right) \\ & \wedge \left(\ell_4 \vee \neg \mathbf{u}_2 \vee \mathbf{u}_3 \right) \wedge \\ & \dots \wedge \left(\ell_{k-2} \vee \neg \mathbf{u}_{k-4} \vee \mathbf{u}_{k-3} \right) \wedge \left(\ell_{k-1} \vee \ell_k \vee \neg \mathbf{u}_{k-3} \right). \end{aligned}$$

Claim

\mathbf{c} is satisfiable iff \mathbf{c}' is satisfiable.

Another way to see it — reduce size of clause by one:

$$\mathbf{c}' = \left(\ell_1 \vee \ell_2 \dots \vee \ell_{k-2} \vee \mathbf{u}_{k-3} \right) \wedge \left(\ell_{k-1} \vee \ell_k \vee \neg \mathbf{u}_{k-3} \right).$$

An Example

Example

$$\begin{aligned}\varphi = & (\neg x_1 \vee \neg x_4) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \\ & \wedge (\neg x_2 \vee \neg x_3 \vee x_4 \vee x_1) \wedge (x_1).\end{aligned}$$

Equivalent form:

$$\psi = (\neg x_1 \vee \neg x_4 \vee z) \wedge (\neg x_1 \vee \neg x_4 \vee \neg z)$$

An Example

Example

$$\begin{aligned}\varphi = & (\neg x_1 \vee \neg x_4) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \\ & \wedge (\neg x_2 \vee \neg x_3 \vee x_4 \vee x_1) \wedge (x_1).\end{aligned}$$

Equivalent form:

$$\begin{aligned}\psi = & (\neg x_1 \vee \neg x_4 \vee z) \wedge (\neg x_1 \vee \neg x_4 \vee \neg z) \\ & \wedge (x_1 \vee \neg x_2 \vee \neg x_3)\end{aligned}$$

An Example

Example

$$\begin{aligned}\varphi = & \left(\neg x_1 \vee \neg x_4 \right) \wedge \left(x_1 \vee \neg x_2 \vee \neg x_3 \right) \\ & \wedge \left(\neg x_2 \vee \neg x_3 \vee x_4 \vee x_1 \right) \wedge \left(x_1 \right).\end{aligned}$$

Equivalent form:

$$\begin{aligned}\psi = & \left(\neg x_1 \vee \neg x_4 \vee z \right) \wedge \left(\neg x_1 \vee \neg x_4 \vee \neg z \right) \\ & \wedge \left(x_1 \vee \neg x_2 \vee \neg x_3 \right) \\ & \wedge \left(\neg x_2 \vee \neg x_3 \vee y_1 \right) \wedge \left(x_4 \vee x_1 \vee \neg y_1 \right)\end{aligned}$$

An Example

Example

$$\begin{aligned}\varphi = & \left(\neg x_1 \vee \neg x_4 \right) \wedge \left(x_1 \vee \neg x_2 \vee \neg x_3 \right) \\ & \wedge \left(\neg x_2 \vee \neg x_3 \vee x_4 \vee x_1 \right) \wedge \left(x_1 \right).\end{aligned}$$

Equivalent form:

$$\begin{aligned}\psi = & \left(\neg x_1 \vee \neg x_4 \vee z \right) \wedge \left(\neg x_1 \vee \neg x_4 \vee \neg z \right) \\ & \wedge \left(x_1 \vee \neg x_2 \vee \neg x_3 \right) \\ & \wedge \left(\neg x_2 \vee \neg x_3 \vee y_1 \right) \wedge \left(x_4 \vee x_1 \vee \neg y_1 \right) \\ & \wedge \left(x_1 \vee u \vee v \right) \wedge \left(x_1 \vee u \vee \neg v \right) \\ & \wedge \left(x_1 \vee \neg u \vee v \right) \wedge \left(x_1 \vee \neg u \vee \neg v \right).\end{aligned}$$

Overall Reduction Algorithm

Reduction from SAT to 3SAT

```
ReduceSATto3SAT( $\varphi$ ):
```

```
  //  $\varphi$ : CNF formula.
```

```
  for each clause  $c$  of  $\varphi$  do
```

```
    if  $c$  does not have exactly 3 literals then  
      construct  $c'$  as before
```

```
    else
```

```
       $c' = c$ 
```

```
   $\psi$  is conjunction of all  $c'$  constructed in loop
```

```
  return Solver3SAT( $\psi$ )
```

Correctness (informal)

φ is satisfiable iff ψ is satisfiable because for each clause c , the new 3CNF formula c' is logically equivalent to c .

What about **2SAT**?

2SAT can be solved in polynomial time! (specifically, linear time!)

No known polynomial time reduction from **SAT** (or **3SAT**) to **2SAT**. If there was, then **SAT** and **3SAT** would be solvable in polynomial time.

Why the reduction from **3SAT** to **2SAT** fails?

Consider a clause $(x \vee y \vee z)$. We need to reduce it to a collection of **2CNF** clauses. Introduce a fresh variable α , and rewrite this as

$$\begin{array}{ll} (x \vee y \vee \alpha) \wedge (\neg\alpha \vee z) & \text{(bad! clause with 3 vars)} \\ \text{or } (x \vee \alpha) \wedge (\neg\alpha \vee y \vee z) & \text{(bad! clause with 3 vars).} \end{array}$$

(In animal farm language: **2SAT** good, **3SAT** bad.)

What about **2SAT**?

A challenging exercise: Given a **2SAT** formula show to compute its satisfying assignment...

(Hint: Create a graph with two vertices for each variable (for a variable x there would be two vertices with labels $x = 0$ and $x = 1$). For every **2CNF** clause add two directed edges in the graph. The edges are implication edges: They state that if you decide to assign a certain value to a variable, then you must assign a certain value to some other variable.

Now compute the strong connected components in this graph, and continue from there...)

Independent Set

Problem: Independent Set

Instance: A graph G , integer k .

Question: Is there an independent set in G of size k ?

3SAT \leq_P Independent Set

The reduction 3SAT \leq_P Independent Set

Input: Given a 3CNF formula φ

Goal: Construct a graph G_φ and number k such that G_φ has an independent set of size k if and only if φ is satisfiable.

3SAT \leq_p Independent Set

The reduction 3SAT \leq_p Independent Set

Input: Given a 3CNF formula φ

Goal: Construct a graph G_φ and number k such that G_φ has an independent set of size k if and only if φ is satisfiable.

G_φ should be constructable in time polynomial in size of φ

3SAT \leq_P Independent Set

The reduction 3SAT \leq_P Independent Set

Input: Given a 3CNF formula φ

Goal: Construct a graph G_φ and number k such that G_φ has an independent set of size k if and only if φ is satisfiable.

G_φ should be constructable in time polynomial in size of φ

Importance of reduction: Although 3SAT is much more expressive, it can be reduced to a seemingly specialized Independent Set problem.

Notice: We handle only 3CNF formulas – reduction would not work for other kinds of boolean formulas.

Interpreting 3SAT

There are two ways to think about 3SAT

Interpreting 3SAT

There are two ways to think about **3SAT**

- 1 Find a way to assign 0/1 (false/true) to the variables such that the formula evaluates to true, that is each clause evaluates to true.

Interpreting 3SAT

There are two ways to think about **3SAT**

- 1 Find a way to assign 0/1 (false/true) to the variables such that the formula evaluates to true, that is each clause evaluates to true.
- 2 Pick a literal from each clause and find a truth assignment to make all of them true

Interpreting 3SAT

There are two ways to think about **3SAT**

- 1 Find a way to assign 0/1 (false/true) to the variables such that the formula evaluates to true, that is each clause evaluates to true.
- 2 Pick a literal from each clause and find a truth assignment to make all of them true. You will fail if two of the literals you pick are in **conflict**, i.e., you pick x_i and $\neg x_i$

We will take the second view of **3SAT** to construct the reduction.

The Reduction

- 1 G_φ will have one vertex for each literal in a clause

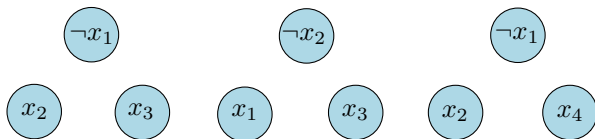


Figure : Graph for

$$\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$$

The Reduction

- 1 G_φ will have one vertex for each literal in a clause
- 2 Connect the 3 literals in a clause to form a triangle; the independent set will pick at most one vertex from each clause, which will correspond to the literal to be set to true

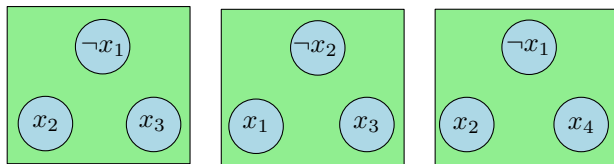


Figure : Graph for

$$\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$$

The Reduction

- 1 G_φ will have one vertex for each literal in a clause
- 2 Connect the 3 literals in a clause to form a triangle; the independent set will pick at most one vertex from each clause, which will correspond to the literal to be set to true

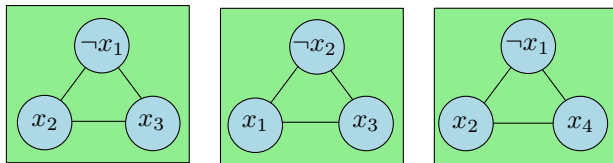


Figure : Graph for

$$\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$$

The Reduction

- 1 G_φ will have one vertex for each literal in a clause
- 2 Connect the 3 literals in a clause to form a triangle; the independent set will pick at most one vertex from each clause, which will correspond to the literal to be set to true
- 3 Connect 2 vertices if they label complementary literals; this ensures that the literals corresponding to the independent set do not have a conflict

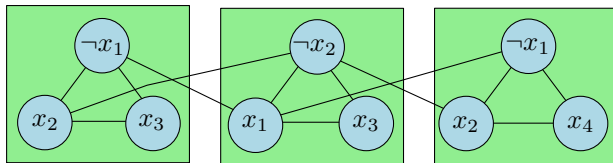


Figure : Graph for

$$\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$$

The Reduction

- 1 G_φ will have one vertex for each literal in a clause
- 2 Connect the 3 literals in a clause to form a triangle; the independent set will pick at most one vertex from each clause, which will correspond to the literal to be set to true
- 3 Connect 2 vertices if they label complementary literals; this ensures that the literals corresponding to the independent set do not have a conflict
- 4 Take k to be the number of clauses

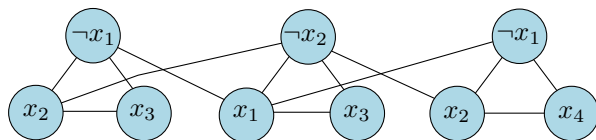


Figure : Graph for

$$\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$$

Correctness

Proposition

φ is satisfiable iff \mathbf{G}_φ has an independent set of size \mathbf{k} (= number of clauses in φ).

Proof.

\Rightarrow Let \mathbf{a} be the truth assignment satisfying φ

Proposition

φ is satisfiable iff G_φ has an independent set of size k (= number of clauses in φ).

Proof.

\Rightarrow Let \mathbf{a} be the truth assignment satisfying φ

- 1 Pick one of the vertices, corresponding to true literals under \mathbf{a} , from each triangle. This is an independent set of the appropriate size □

Correctness (contd)

Proposition

φ is satisfiable iff \mathbf{G}_φ has an independent set of size k (= number of clauses in φ).

Proof.

← Let \mathbf{S} be an independent set of size k

- 1 \mathbf{S} must contain exactly one vertex from each clause
- 2 \mathbf{S} cannot contain vertices labeled by conflicting clauses
- 3 Thus, it is possible to obtain a truth assignment that makes in the literals in \mathbf{S} true; such an assignment satisfies one literal in every clause □

Transitivity of Reductions

Lemma

$X \leq_P Y$ and $Y \leq_P Z$ implies that $X \leq_P Z$.

Note: $X \leq_P Y$ does not imply that $Y \leq_P X$ and hence it is very important to know the FROM and TO in a reduction.

To prove $X \leq_P Y$ you need to show a reduction FROM X TO Y
In other words show that an algorithm for Y implies an algorithm for X .

Part II

Definition of NP

Clique?

Given a graph G with n vertices and m edges, consider a certificate (i.e., proof) that G indeed has a clique of size k . We have that:

- (A) There is such a certificate of length $t = O(n)$, and it can be verified in $O(2^t)$ time.
- (B) There is such a certificate of length $t = O(n2^n)$, and it can be verified in $O(t)$ time.
- (C) There is such a certificate of length $t = O(k)$, and it can be verified in $O(n^2)$ time.
- (D) There is no certificate for this problem.
- (E) If there was such a certificate, then we could solve the problem in polynomial time.

Not Clique?

Given a graph G with n vertices and m edges, consider a certificate (i.e., proof) that G has **NO** clique of size k . We have that:

- (A) There is such a certificate of length $t = O(n)$, and it can be verified in $O(2^t)$ time.
- (B) There is such a certificate of length $t = O(n2^n)$, and it can be verified in $O(t)$ time.
- (C) There is such a certificate of length $t = O(k)$, and it can be verified in $O(n^2)$ time.
- (D) There is no certificate for this problem.
- (E) If there was such a certificate, then we could solve the problem in polynomial time.

Problems

- 1 Independent Set
- 2 Vertex Cover
- 3 Set Cover
- 4 SAT
- 5 3SAT

Problems

- 1 Independent Set
- 2 Vertex Cover
- 3 Set Cover
- 4 SAT
- 5 3SAT

Relationship

3SAT \leq_P **Independent Set**

Problems

- 1 Independent Set
- 2 Vertex Cover
- 3 Set Cover
- 4 SAT
- 5 3SAT

Relationship

$3SAT \leq_P \text{Independent Set} \begin{matrix} \leq_P \\ \geq_P \end{matrix} \text{Vertex Cover}$

Problems

- 1 Independent Set
- 2 Vertex Cover
- 3 Set Cover
- 4 SAT
- 5 3SAT

Relationship

$3SAT \leq_P Independent\ Set \stackrel{\leq_P}{\geq_P} Vertex\ Cover \leq_P Set\ Cover$

Problems

- 1 Independent Set
- 2 Vertex Cover
- 3 Set Cover
- 4 SAT
- 5 3SAT

Relationship

$3SAT \leq_P Independent\ Set \stackrel{\leq_P}{\geq_P} Vertex\ Cover \leq_P Set\ Cover$
 $3SAT \leq_P SAT \leq_P 3SAT$

Problems and Algorithms: Formal Approach

Decision Problems

- 1 **Problem Instance:** Binary string s , with size $|s|$
- 2 **Problem:** A set X of strings on which the answer should be "yes"; we call these YES instances of X . Strings not in X are NO instances of X .

Definition

- 1 A is an **algorithm for problem X** if $A(s) = \text{"yes"}$ iff $s \in X$.
- 2 A is said to have a **polynomial running time** if there is a polynomial $p(\cdot)$ such that for every string s , $A(s)$ terminates in at most $O(p(|s|))$ steps.

Polynomial Time

Definition

Polynomial time (denoted by **P**) is the class of all (decision) problems that have an algorithm that solves it in polynomial time.

Polynomial Time

Definition

Polynomial time (denoted by **P**) is the class of all (decision) problems that have an algorithm that solves it in polynomial time.

Example

Problems in **P** include

- 1 Is there a shortest path from **s** to **t** of length $\leq k$ in **G**?
- 2 Is there a flow of value $\geq k$ in network **G**?
- 3 Is there an assignment to variables to satisfy given linear constraints?

Efficiency Hypothesis

A problem X has an efficient algorithm iff $X \in P$, that is X has a polynomial time algorithm.

Justifications:

- 1 Robustness of definition to variations in machines.
- 2 A sound theoretical definition.
- 3 Most known polynomial time algorithms for “natural” problems have small polynomial running times.

Problems with no known polynomial time algorithms

Problems

- 1 **Independent Set**
- 2 **Vertex Cover**
- 3 **Set Cover**
- 4 **SAT**
- 5 **3SAT**

There are of course undecidable problems (no algorithm at all!) but many problems that we want to solve are of similar flavor to the above.

Question: What is common to above problems?

Efficient Checkability

Above problems share the following feature:

Checkability

For any YES instance I_X of X there is a proof/certificate/solution that is of length $\text{poly}(|I_X|)$ such that given a proof one can efficiently check that I_X is indeed a YES instance.

Efficient Checkability

Above problems share the following feature:

Checkability

For any YES instance I_X of X there is a proof/certificate/solution that is of length $\text{poly}(|I_X|)$ such that given a proof one can efficiently check that I_X is indeed a YES instance.

Examples:

- 1 **SAT** formula φ : proof is a satisfying assignment.
- 2 **Independent Set** in graph G and k : a subset S of vertices.

Definition

An algorithm $C(\cdot, \cdot)$ is a **certifier** for problem X if for every $s \in X$ there is some string t such that $C(s, t) = \text{"yes"}$, and conversely, if for some s and t , $C(s, t) = \text{"yes"}$ then $s \in X$. The string t is called a **certificate** or **proof** for s .

Definition

An algorithm $\mathbf{C}(\cdot, \cdot)$ is a **certifier** for problem \mathbf{X} if for every $\mathbf{s} \in \mathbf{X}$ there is some string \mathbf{t} such that $\mathbf{C}(\mathbf{s}, \mathbf{t}) = \text{"yes"}$, and conversely, if for some \mathbf{s} and \mathbf{t} , $\mathbf{C}(\mathbf{s}, \mathbf{t}) = \text{"yes"}$ then $\mathbf{s} \in \mathbf{X}$.

The string \mathbf{t} is called a **certificate** or **proof** for \mathbf{s} .

Definition (Efficient Certifier.)

A certifier \mathbf{C} is an **efficient certifier** for problem \mathbf{X} if there is a polynomial $\mathbf{p}(\cdot)$ such that for every string \mathbf{s} , we have that

- ★ $\mathbf{s} \in \mathbf{X}$ if and only if
- ★ there is a string \mathbf{t} :
 - ① $|\mathbf{t}| \leq \mathbf{p}(|\mathbf{s}|)$,
 - ② $\mathbf{C}(\mathbf{s}, \mathbf{t}) = \text{"yes"}$,
 - ③ and \mathbf{C} runs in polynomial time.

Example: Independent Set

- 1 **Problem:** Does $G = (V, E)$ have an independent set of size $\geq k$?
 - 1 **Certificate:** Set $S \subseteq V$.
 - 2 **Certifier:** Check $|S| \geq k$ and no pair of vertices in S is connected by an edge.

Example: Vertex Cover

- ① **Problem:** Does \mathbf{G} have a vertex cover of size $\leq \mathbf{k}$?
 - ① **Certificate:** $\mathbf{S} \subseteq \mathbf{V}$.
 - ② **Certifier:** Check $|\mathbf{S}| \leq \mathbf{k}$ and that for every edge at least one endpoint is in \mathbf{S} .

Example: SAT

- 1 **Problem:** Does formula φ have a satisfying truth assignment?
 - 1 **Certificate:** Assignment \mathbf{a} of **0/1** values to each variable.
 - 2 **Certifier:** Check each clause under \mathbf{a} and say “yes” if all clauses are true.

Example: Composites

Problem: Composite

Instance: A number s .

Question: Is the number s a composite?

- 1 **Problem: Composite.**
 - 1 **Certificate:** A factor $t \leq s$ such that $t \neq 1$ and $t \neq s$.
 - 2 **Certifier:** Check that t divides s .

Not composite?

Problem: **Not Composite**

Instance: A number s .

Question: Is the number s not a composite?

The problem **Not Composite** is

- (A) Can be solved in linear time.
- (B) in **P**.
- (C) Can be solved in exponential time.
- (D) Does not have a certificate or an efficient certifier.
- (E) The status of this problem is still open.

Example: NFA Universality

Problem: NFA Universality

Instance: Description of a NFA M .

Question: Is $L(M) = \Sigma^*$, that is, does M accept all strings?

- 1 Problem: NFA Universality.
 - 1 Certificate: A DFA M' equivalent to M
 - 2 Certifier: Check that $L(M') = \Sigma^*$

Example: NFA Universality

Problem: NFA Universality

Instance: Description of a NFA M .

Question: Is $L(M) = \Sigma^*$, that is, does M accept all strings?

- 1 Problem: NFA Universality.
 - 1 Certificate: A DFA M' equivalent to M
 - 2 Certifier: Check that $L(M') = \Sigma^*$

Certifier is efficient but certificate is not necessarily short! We do not know if the problem is in **NP**.

Example: A String Problem

Problem: PCP

Instance: Two sets of binary strings $\alpha_1, \dots, \alpha_n$ and β_1, \dots, β_n

Question: Are there indices i_1, i_2, \dots, i_k such that $\alpha_{i_1} \alpha_{i_2} \dots \alpha_{i_k} = \beta_{i_1} \beta_{i_2} \dots \beta_{i_k}$

① Problem: PCP

- ① **Certificate:** A sequence of indices i_1, i_2, \dots, i_k
- ② **Certifier:** Check that $\alpha_{i_1} \alpha_{i_2} \dots \alpha_{i_k} = \beta_{i_1} \beta_{i_2} \dots \beta_{i_k}$

Example: A String Problem

Problem: PCP

Instance: Two sets of binary strings $\alpha_1, \dots, \alpha_n$ and β_1, \dots, β_n

Question: Are there indices i_1, i_2, \dots, i_k such that $\alpha_{i_1} \alpha_{i_2} \dots \alpha_{i_k} = \beta_{i_1} \beta_{i_2} \dots \beta_{i_k}$

① Problem: PCP

- ① **Certificate:** A sequence of indices i_1, i_2, \dots, i_k
- ② **Certifier:** Check that $\alpha_{i_1} \alpha_{i_2} \dots \alpha_{i_k} = \beta_{i_1} \beta_{i_2} \dots \beta_{i_k}$

PCP = Posts Correspondence Problem and it is undecidable!
Implies no finite bound on length of certificate!

Nondeterministic Polynomial Time

Definition

Nondeterministic Polynomial Time (denoted by **NP**) is the class of all problems that have efficient certifiers.

Nondeterministic Polynomial Time

Definition

Nondeterministic Polynomial Time (denoted by **NP**) is the class of all problems that have efficient certifiers.

Example

Independent Set, **Vertex Cover**, **Set Cover**, **SAT**, **3SAT**, and **Composite** are all examples of problems in **NP**.

Why is it called...

Nondeterministic Polynomial Time

A certifier is an algorithm $C(I, c)$ with two inputs:

- 1 I : instance.
- 2 c : proof/certificate that the instance is indeed a YES instance of the given problem.

One can think about C as an algorithm for the original problem, if:

- 1 Given I , the algorithm guesses (non-deterministically, and who knows how) a certificate c .
- 2 The algorithm now verifies the certificate c for the instance I .

NP can be equivalently described using Turing machines.

Asymmetry in Definition of NP

Note that only YES instances have a short proof/certificate. NO instances need not have a short certificate.

Example

SAT formula φ . No easy way to prove that φ is NOT satisfiable!

More on this and **co-NP** later on.

P versus NP

Proposition

$P \subseteq NP$.

P versus NP

Proposition

$$P \subseteq NP.$$

For a problem in **P** no need for a certificate!

Proof.

Consider problem $X \in P$ with algorithm **A**. Need to demonstrate that X has an efficient certifier:

- 1 Certifier **C** on input s, t , runs $A(s)$ and returns the answer.
- 2 **C** runs in polynomial time.
- 3 If $s \in X$, then for every t , $C(s, t) = \text{"yes"}$.
- 4 If $s \notin X$, then for every t , $C(s, t) = \text{"no"}$.



Exponential Time

Definition

Exponential Time (denoted **EXP**) is the collection of all problems that have an algorithm which on input s runs in exponential time, i.e., $O(2^{\text{poly}(|s|)})$.

Exponential Time

Definition

Exponential Time (denoted **EXP**) is the collection of all problems that have an algorithm which on input s runs in exponential time, i.e., $O(2^{\text{poly}(|s|)})$.

Example: $O(2^n)$, $O(2^{n \log n})$, $O(2^{n^3})$, ...

NP versus EXP

Proposition

$\text{NP} \subseteq \text{EXP}$.

Proof.

Let $X \in \text{NP}$ with certifier C . Need to design an exponential time algorithm for X .

- 1 For every t , with $|t| \leq p(|s|)$ run $C(s, t)$; answer “yes” if any one of these calls returns “yes”.
- 2 The above algorithm correctly solves X (exercise).
- 3 Algorithm runs in $O(q(|s| + |p(s)|)2^{p(|s|)})$, where q is the running time of C . □

Examples

- 1 **SAT**: try all possible truth assignment to variables.
- 2 **Independent Set**: try all possible subsets of vertices.
- 3 **Vertex Cover**: try all possible subsets of vertices.

Is **NP** efficiently solvable?

We know $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXP}$.

Is **NP** efficiently solvable?

We know $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXP}$.

Big Question

Is there are problem in **NP** that **does not** belong to **P**? Is $\mathbf{P} = \mathbf{NP}$?

If $P = NP$...

Or: If pigs could fly then life would be sweet.

- ① Many important optimization problems can be solved efficiently.

If $P = NP$...

Or: If pigs could fly then life would be sweet.

- 1 Many important optimization problems can be solved efficiently.
- 2 The **RSA** cryptosystem can be broken.

If $P = NP$...

Or: If pigs could fly then life would be sweet.

- 1 Many important optimization problems can be solved efficiently.
- 2 The **RSA** cryptosystem can be broken.
- 3 No security on the web.

If $P = NP$. . .

Or: If pigs could fly then life would be sweet.

- 1 Many important optimization problems can be solved efficiently.
- 2 The **RSA** cryptosystem can be broken.
- 3 No security on the web.
- 4 No e-commerce . . .

If $P = NP$. . .

Or: If pigs could fly then life would be sweet.

- 1 Many important optimization problems can be solved efficiently.
- 2 The **RSA** cryptosystem can be broken.
- 3 No security on the web.
- 4 No e-commerce . . .
- 5 Creativity can be automated! Proofs for mathematical statement can be found by computers automatically (if short ones exist).

If $P = NP$ this implies that...

- (A) **Vertex Cover** can be solved in polynomial time.
- (B) $P = EXP$.
- (C) $EXP \subseteq P$.
- (D) All of the above.

P versus NP

Status

Relationship between **P** and **NP** remains one of the most important open problems in mathematics/computer science.

Consensus: Most people feel/believe **P** \neq **NP**.

Resolving **P** versus **NP** is a Clay Millennium Prize Problem. You can win a million dollars in addition to a Turing award and major fame!

Part III

Not for lecture: Converting any
boolean formula into CNF

The dark art of formula conversion into CNF

Consider an arbitrary boolean formula ϕ defined over k variables. To keep the discussion concrete, consider the formula $\phi \equiv x_k = x_i \wedge x_j$. We would like to convert this formula into an equivalent **CNF** formula.

Formula conversion into CNF

Step 1

Build a truth table for the boolean formula.

x_k	x_i	x_j	value of $x_k = x_i \wedge x_j$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Formula conversion into CNF

Step 1.5 - understand what a single CNF clause represents

Given an assignment, say, $x_k = 0$, $x_i = 0$ and $x_j = 1$, consider the CNF clause $x_k \vee x_i \vee \overline{x_j}$ (you negate a variable if it is assigned one). Its truth table is

x_k	x_i	x_j	$x_k \vee x_i \vee \overline{x_j}$
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Observe that a single clause assigns zero to one row, and one everywhere else. A conjunction of several such clauses, as such, would result in a formula that is 0 in all the rows that corresponds to these clauses, and one everywhere else.

Formula conversion into CNF

Step 2

Write down the **CNF** clause for every row in the table that is zero.

x_k	x_i	x_j	$x_k = x_i \wedge x_j$	CNF clause
0	0	0	1	
0	0	1	1	
0	1	0	1	
0	1	1	0	$x_k \vee \bar{x}_i \vee \bar{x}_j$
1	0	0	0	$\bar{x}_k \vee x_i \vee x_j$
1	0	1	0	$\bar{x}_k \vee x_i \vee \bar{x}_j$
1	1	0	0	$\bar{x}_k \vee \bar{x}_i \vee x_j$
1	1	1	1	

The conjunction (i.e., and) of all these clauses is clearly equivalent to the original formula. In this case

$$\psi \equiv (x_k \vee \bar{x}_i \vee \bar{x}_j) \wedge (\bar{x}_k \vee x_i \vee x_j) \wedge (\bar{x}_k \vee x_i \vee \bar{x}_j) \wedge (\bar{x}_k \vee \bar{x}_i \vee x_j)$$

Formula conversion into CNF

Step 3 - simplify if you want to

Using that $(x \vee y) \wedge (x \vee \bar{y}) = x$, we have that:

① $(\bar{x}_k \vee x_i \vee x_j) \wedge (\bar{x}_k \vee x_i \vee \bar{x}_j)$ is equivalent to $(\bar{x}_k \vee x_i)$.

② $(\bar{x}_k \vee x_i \vee x_j) \wedge (\bar{x}_k \vee \bar{x}_i \vee x_j)$ is equivalent to $(\bar{x}_k \vee x_j)$.

Using the above two observations, we have that our formula

$\psi \equiv (\bar{x}_k \vee x_i \vee x_j) \wedge (\bar{x}_k \vee x_i \vee \bar{x}_j) \wedge (\bar{x}_k \vee x_i \vee \bar{x}_j) \wedge (\bar{x}_k \vee \bar{x}_i \vee x_j)$
is equivalent to

$$\psi \equiv (\bar{x}_k \vee x_i \vee \bar{x}_j) \wedge (\bar{x}_k \vee x_i) \wedge (\bar{x}_k \vee x_j).$$

We conclude:

Lemma

The formula $x_k = x_i \wedge x_j$ is equivalent to the CNF formula

$$\psi \equiv (\bar{x}_k \vee \bar{x}_i \vee \bar{x}_j) \wedge (\bar{x}_k \vee x_i) \wedge (\bar{x}_k \vee x_j).$$

Notes

Notes

Notes

Notes