Reductions and NP

Lecture 21

April 14, 2011

Part I

Reductions Continued

Polynomial Time Reduction

Karp reduction

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

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

Notation: $\mathbf{X} \leq_P \mathbf{Y}$ if $\mathbf{X}$ reduces to $\mathbf{Y}$

Proposition

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

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

A More General Reduction

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

- on any given instance $\mathbf{I_X}$ of $\mathbf{X}$, $\mathcal{A}$ uses polynomial in $|\mathbf{I_X}|$ "steps"
- a step is either a standard computation step or
- a sub-routine call to an algorithm that solves $\mathbf{Y}$

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

Above reduction is a **Turing reduction**.

A More General Reduction

Turing Reduction

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

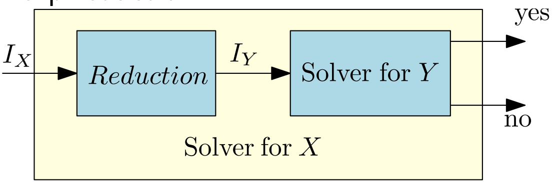
- on any given instance $\mathbf{I_X}$ of $\mathbf{X}$, $\mathcal{A}$ uses polynomial in $|\mathbf{I_X}|$ "steps"
- a step is either a standard computation step or
- a sub-routine call to an algorithm that solves $\mathbf{Y}$

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

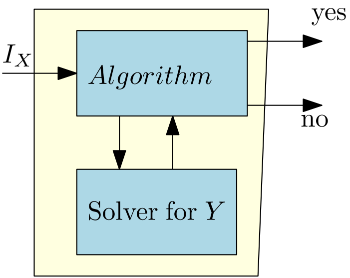
Above reduction is a **Turing reduction**.

Comparing reductions

- Karp reduction:



- Turing reduction:



Turing reduction

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

Example of Turing Reduction

Input Collection of arcs on a circle.

Goal Compute the maximum number of non-overlapping arcs.

Reduced to the following problem:?

Input Collection of intervals on the line.

Goal Compute the maximum number of non-overlapping intervals.

How? Used algorithm for interval problem multiple times.

Example of Turing Reduction

Input Collection of arcs on a circle.

Goal Compute the maximum number of non-overlapping arcs.

Reduced to the following problem:?

Input Collection of intervals on the line.

Goal Compute the maximum number of non-overlapping intervals.

How? Used algorithm for interval problem multiple times.

Example of Turing Reduction

Input Collection of arcs on a circle.

Goal Compute the maximum number of non-overlapping arcs.

Reduced to the following problem:?

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

- Turing reductions more general than Karp reductions
- Turing reduction useful in obtaining algorithms via reductions
- Karp reduction is simpler and easier to use to prove hardness of problems
- Perhaps surprisingly, Karp reductions, although limited, suffice for most known NP-Completeness proofs

Propositional Formulas

Definition

Consider a set of boolean variables $x_1, x_2, \ldots x_n$

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

Satisfiability

SAT

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

Example

$(x_1 \lor x_2 \lor \neg x_4) \land (x_2 \lor \neg x_3) \land x_5$ is satisfiable; take $x_1, x_2, \ldots x_5$ to be all true

$(x_1 \lor \neg x_2) \land (\neg x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2) \land (x_1 \lor x_2)$ 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

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

SAT $\leq_P$ 3SAT

How SAT is different from 3SAT?

In **SAT** clauses might have arbitrary length - $\mathbf{1, 2, 3, \ldots}$ variables:

$$\left(\mathbf{x} \vee \mathbf{y} \vee \mathbf{z} \vee \mathbf{w} \vee \mathbf{u}\right) \wedge \left(\neg\mathbf{x} \vee \neg\mathbf{y} \vee \neg\mathbf{z} \vee \mathbf{w} \vee \mathbf{u}\right) \wedge \left(\neg\mathbf{x}\right)$$

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

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

SAT $\leq_P$ 3SAT

Easy to see that **3SAT** $\leq_P$ **SAT**. A **3SAT** instance is also an instance of **SAT**.

We can show that **SAT** $\leq_P$ **3SAT**.

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

- φ is satisfiable iff φ' is satisfiable
- 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

Easy to see that **3SAT** $\leq_P$ **SAT**. A **3SAT** instance is also an instance of **SAT**.

We can show that **SAT** $\leq_P$ **3SAT**.

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

- φ is satisfiable iff φ' is satisfiable
- 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

Easy to see that **3SAT** $\leq_P$ **SAT**. A **3SAT** instance is also an instance of **SAT**.

We can show that **SAT** $\leq_P$ **3SAT**.

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

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

Idea: if a clause of φ is not of length $\mathbf{3}$, replace it with several clauses of length exactly $\mathbf{3}$

Reduction Ideas

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

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

$$\begin{aligned} \mathbf{c}' = \quad & (\ell \vee \mathbf{u} \vee \mathbf{v}) \wedge (\ell \vee \mathbf{u} \vee \neg\mathbf{v}) \\ & \wedge (\ell \vee \neg\mathbf{u} \vee \mathbf{v}) \wedge (\ell \vee \neg\mathbf{u} \vee \neg\mathbf{v}) \end{aligned}$$

  Observe that $\mathbf{c}'$ is satisfiable iff $\mathbf{c}$ is satisfiable

SAT $\leq_P$ 3SAT (contd)

Reduction Ideas: 2 and more literals

- 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

SAT $\leq_P$ 3SAT (contd)

Clauses with more than 3 literals

Let $c = \ell_1 \vee \cdots \vee \ell_k$. Let $u_1, \ldots u_{k-3}$ be new variables. Consider

$$
\begin{aligned}
c' = & (\ell_1 \vee \ell_2 \vee u_1) \;\wedge\; (\ell_3 \vee \neg u_1 \vee u_2) \\
& \wedge\; (\ell_4 \vee \neg u_2 \vee u_3) \wedge \\
& \cdots \wedge\; (\ell_{k-2} \vee \neg u_{k-4} \vee u_{k-3}) \;\wedge\; (\ell_{k-1} \vee \ell_k \vee \neg u_{k-3})
\end{aligned}
$$

c is satisfiable iff c' is satisfiable

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

$$c' = (\ell_1 \vee \ell_2 \ldots \vee \ell_{k-2} \vee u_{k-3}) \wedge (\ell_{k-1} \vee \ell_k \vee \neg u_{k-3})$$

An Example

Example

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

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

An Example

Example

$$\varphi = (\neg x_1 \lor \neg x_4) \land (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_2 \lor \neg x_3 \lor x_4 \lor x_1) \land (x_1).$$

$$\begin{aligned}\psi \;=\; & (\neg x_1 \lor \neg x_4 \lor z) \land (\neg x_1 \lor \neg x_4 \lor \neg z) \\ & \land (x_1 \lor \neg x_2 \lor \neg x_3) \\ & \land (\neg x_2 \lor \neg x_3 \lor y_1) \land (x_4 \lor x_1 \lor \neg y_1) \\ & \land (x_1 \lor u \lor v) \land (x_1 \lor u \lor \neg v) \land (x_1 \lor \neg u \lor v) \land (x_1 \lor\end{aligned}$$

An Example

Example

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

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

An Example

Example

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

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

Overall Reduction Algorithm

```
Input:  CNF formula φ
for each clause c of φ
    if c does not have exactly 3 literals
        construct c' as before
    else
        c' = c
ψ is conjunction of all c' constructed in loop
is ψ satisfiable?
```

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! (In fact, 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 $(\mathbf{x} \vee \mathbf{y} \vee \mathbf{z})$. We need to reduce it to a collection of $\mathbf{2}\mathrm{CNF}$ clauses. Introduce a face variable α, and rewrite this as

$$(\mathbf{x} \vee \mathbf{y} \vee \alpha) \wedge (\neg\alpha \vee \mathbf{z}) \qquad \text{(bad! clause with 3 vars)}$$

or $$(\mathbf{x} \vee \alpha) \wedge (\neg\alpha \vee \mathbf{y} \vee \mathbf{z}) \qquad \text{(bad! clause with 3 vars).}$$

(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 $\mathbf{x}$ there would be two vertices with labels $\mathbf{x = 0}$ and $\mathbf{x = 1}$). For ever $\mathbf{2}\mathrm{CNF}$ 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...)

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.

$3SAT \leq_P$ 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**

- 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
- 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.

Interpreting 3SAT

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

- 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
- 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.

Interpreting 3SAT

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

- 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
- 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.

Interpreting 3SAT

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

- 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
- 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

- G_φ will have one vertex for each literal in a clause
- 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
- Connect 2 vertices if they label complementary literals; this ensures that the literals corresponding to the independent set do not have a conflict
- Take k to be the number of clauses

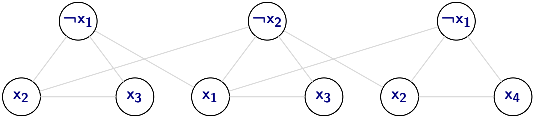


Figure: Graph for
$\varphi = (\neg x_1 \lor x_2 \lor x_3) \land (x_1 \lor \neg x_2 \lor x_3) \land (\neg x_1 \lor x_2 \lor x_4)$

The Reduction

- G_φ will have one vertex for each literal in a clause
- 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
- Connect 2 vertices if they label complementary literals; this ensures that the literals corresponding to the independent set do not have a conflict
- Take k to be the number of clauses

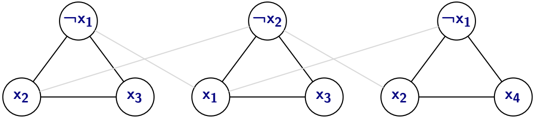


Figure: Graph for
$\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

The Reduction

- G_φ will have one vertex for each literal in a clause
- 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
- Connect 2 vertices if they label complementary literals; this ensures that the literals corresponding to the independent set do not have a conflict
- Take k to be the number of clauses

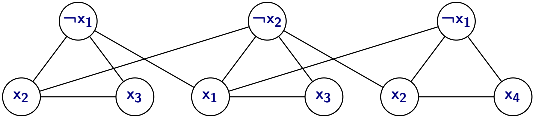


Figure: Graph for
$\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

The Reduction

- G_φ will have one vertex for each literal in a clause
- 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
- Connect 2 vertices if they label complementary literals; this ensures that the literals corresponding to the independent set do not have a conflict
- Take k to be the number of clauses

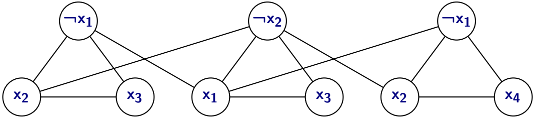


Figure: Graph for
$\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

Correctness

Proposition

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

Proof.

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

- 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

Proposition

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

Proof.

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

- 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 $\mathbf{G}_\varphi$ has an independent set of size $\mathbf{k}$ ($=$ number of clauses in φ).

Proof.

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

- $\mathbf{S}$ must contain exactly one vertex from each clause
- $\mathbf{S}$ cannot contain vertices labeled by conflicting clauses
- 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

$\mathbf{X} \leq_{\mathbf{P}} \mathbf{Y}$ and $\mathbf{Y} \leq_{\mathbf{P}} \mathbf{Z}$ implies that $\mathbf{X} \leq_{\mathbf{P}} \mathbf{Z}$.

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

To prove $\mathbf{X} \leq_{\mathbf{P}} \mathbf{Y}$ you need to show a reduction FROM $\mathbf{X}$ TO $\mathbf{Y}$
In other words show that an algorithm for $\mathbf{Y}$ implies an algorithm for $\mathbf{X}$.

Part II

Definition of NP

Recap . . .

Problems

- **Independent Set**
- **Vertex Cover**
- **Set Cover**
- **SAT**
- **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**

Recap . . .

Problems

- **Independent Set**
- **Vertex Cover**
- **Set Cover**
- **SAT**
- **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**

Recap ...

Problems

- **Independent Set**
- **Vertex Cover**
- **Set Cover**
- **SAT**
- **3SAT**

Relationship

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

Recap . . .

Problems

- **Independent Set**
- **Vertex Cover**
- **Set Cover**
- **SAT**
- **3SAT**

Relationship

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

3SAT $\leq_P$ SAT $\leq_P$ 3SAT

Recap ...

Problems

- **Independent Set**
- **Vertex Cover**
- **Set Cover**
- **SAT**
- **3SAT**

Relationship

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

Problems and Algorithms: Formal Approach

Decision Problems

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

Definition

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

Polynomial Time

Definition

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

Example

Problems in **P** include

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

Polynomial Time

Definition

Polynomial time (denoted $\mathbf{P}$) is the class of all (decision) problems that have an algorithm that solves it in polynomial time

Example

Problems in $\mathbf{P}$ include

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

Efficiency Hypothesis

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

Justifications:

- robustness of definition to variations in machines
- a sound theoretical definition
- most known polynomial time algorithms for "natural" problems have small polynomial running times

Problems with no known polynomial time algorithms

Problems

- Independent Set
- Vertex Cover
- Set Cover
- **SAT**
- **3SAT**

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

Question: What is common to above problems?

Efficient Checkability

Above problems share the following feature:

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

Examples:

- **SAT** formula φ: proof is a satisfying assignment
- Independent Set in graph $\mathbf{G}$ and $\mathbf{k}$: a subset $\mathbf{S}$ of vertices

Efficient Checkability

Above problems share the following feature:

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

Examples:

- **SAT** formula φ: proof is a satisfying assignment
- Independent Set in graph $\mathbf{G}$ and $\mathbf{k}$: a subset $\mathbf{S}$ of vertices

Certifiers

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 $\mathbf{C}(\mathbf{s}, \mathbf{t}) =$ "yes", and conversely, if for some $\mathbf{s}$ and $\mathbf{t}$, $\mathbf{C}(\mathbf{s}, \mathbf{t}) =$ "yes" then $\mathbf{s} \in \mathbf{X}$.
The string $\mathbf{t}$ is called a certificate or proof for $\mathbf{s}$

Efficient Certifier

$\mathbf{C}$ is an efficient certifier for problem $\mathbf{X}$ if there is a polynomial $\mathbf{p}(\cdot)$ such that for every string $\mathbf{s}$, $\mathbf{s} \in \mathbf{X}$ iff there is a string $\mathbf{t}$ with $|\mathbf{t}| \leq \mathbf{p}(|\mathbf{s}|)$, $\mathbf{C}(\mathbf{s}, \mathbf{t}) =$ "yes" and $\mathbf{C}$ runs in polynomial time

Certifiers

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 $\mathbf{C}(\mathbf{s},\mathbf{t}) =$ "yes", and conversely, if for some $\mathbf{s}$ and $\mathbf{t}$, $\mathbf{C}(\mathbf{s},\mathbf{t}) =$ "yes" then $\mathbf{s} \in \mathbf{X}$.
The string $\mathbf{t}$ is called a certificate or proof for $\mathbf{s}$

Efficient Certifier

$\mathbf{C}$ is an efficient certifier for problem $\mathbf{X}$ if there is a polynomial $\mathbf{p}(\cdot)$ such that for every string $\mathbf{s}$, $\mathbf{s} \in \mathbf{X}$ iff there is a string $\mathbf{t}$ with $|\mathbf{t}| \leq \mathbf{p}(|\mathbf{s}|)$, $\mathbf{C}(\mathbf{s},\mathbf{t}) =$ "yes" and $\mathbf{C}$ runs in polynomial time

Example: Independent Set

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

Example: Vertex Cover

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

Example: SAT

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

Example: Composites

- Problem: Is number $\mathbf{s}$ a composite?
  - Certificate: A factor $\mathbf{t} \leq \mathbf{s}$ such that $\mathbf{t} \neq \mathbf{1}$ and $\mathbf{t} \neq \mathbf{s}$
  - Certifier: Check that $\mathbf{t}$ divides $\mathbf{s}$ (Euclid's algorithm)

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**, Composites are all examples of problems in **NP**

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**, Composites are all examples of problems in **NP**

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**

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

- Certifier C on input s, t, runs $A(s)$ and returns the answer
- C runs in polynomial time
- If $s \in X$ then for every t, $C(s, t) =$ "yes"
- If $s \notin X$ then for every t, $C(s, t) =$ "no" □

P versus NP

Proposition

$\mathbf{P} \subseteq \mathbf{NP}$

For a problem in $\mathbf{P}$ no need for a certificate!

Proof.

Consider problem $\mathbf{X} \in \mathbf{P}$ with algorithm $\mathbf{A}$. Need to demonstrate that $\mathbf{X}$ has an efficient certifier

- Certifier $\mathbf{C}$ on input $\mathbf{s}, \mathbf{t}$, runs $\mathbf{A}(\mathbf{s})$ and returns the answer
- $\mathbf{C}$ runs in polynomial time
- If $\mathbf{s} \in \mathbf{X}$ then for every $\mathbf{t}$, $\mathbf{C}(\mathbf{s}, \mathbf{t}) =$ "yes"
- If $\mathbf{s} \notin \mathbf{X}$ then for every $\mathbf{t}$, $\mathbf{C}(\mathbf{s}, \mathbf{t}) =$ "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., $O(2^{\mathrm{poly}(|s|)})$

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

Exponential Time

Definition

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

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

NP versus EXP

Proposition

$\mathbf{NP} \subseteq \mathbf{EXP}$

Proof.

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

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

Examples

- **SAT**: try all possible truth assignment to variables
- Independent set: try all possible subsets of vertices
- Vertex cover: try all possible subsets of vertices

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 $\mathbf{P} = \mathbf{NP}$?

Is **NP** efficiently solvable?

We know **P** $\subseteq$ **NP** $\subseteq$ **EXP**

Big Question

Is there are problem in **NP** that does not belong to **P**? Is **P** = **NP**?

If P = NP . . .

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

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

If $\mathbf{P} = \mathbf{NP}\dots$

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

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

If P = NP . . .

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

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

If P = NP . . .

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

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

If P = NP . . .

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

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

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 **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!

Notes

Notes

Notes

Notes