CS 473: Algorithms

Chandra Chekuri
chekuri@cs.illinois.edu
3228 Siebel Center

University of Illinois, Urbana-Champaign

Fall 2009

Part I

Reductions Continued

Polynomial Time 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:

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

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 (the one given in the book)

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

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

A More General Reduction

Turing Reduction (the one given in the book)

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

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

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

Above reduction is called a Turing reduction.

Example of Turing Reduction

Recall home work problem:

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

Recall home work problem:

- **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?

Example of Turing Reduction

Recall home work problem:

- **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 $x_1, x_2, \ldots x_n$

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 $\neg x_i$

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.

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

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 conjuction of clauses

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 conjuction 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

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 conjuction 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 3CNF if it is a CNF formula such that every clause has exactly 3 literals

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 conjuction 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 3CNF 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 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

SAT

Given a CNF formula φ, is there a truth assignment to variables 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

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

$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_1 \vee x_2)$ is not satisfiable

Satisfiability

SAT

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

Example

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

$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_1 \vee x_2)$ is not satisfiable

3-SAT

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

Importance of SAT and 3-SAT

- SAT and 3-SAT are basic constraint satisfaction problems
- Many different problems can reduced to them because of the simple yet powerful expressivity 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 $\leq_P$ 3-SAT

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

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

SAT $\leq_P$ 3-SAT

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

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

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

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

SAT $\leq_P$ 3-SAT

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

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

Given φ a SAT formula we create a 3-SAT 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 $\leq_P$ 3-SAT

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.

SAT $\leq_P$ 3-SAT

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.

- Case clause with 1 literal: Let $c = \ell$. 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 $\leq_P$ 3-SAT (contd)

Reduction Ideas: 2 and more literals

SAT $\leq_P$ 3-SAT (contd)

Reduction Ideas: 2 and more literals

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

$$c' = (\ell_1 \vee \ell_2 \vee u) \wedge (\ell_1 \vee \ell_2 \vee \neg u)$$

Again c is satisfiable iff c' is satisfiable

SAT $\leq_P$ 3-SAT (contd)

Reduction Ideas: 2 and more literals

- Case clause with > 3 literals: Let $c = \ell_1 \vee \cdots \vee \ell_k$. Let $u_1, \ldots u_{k-3}$ be new variables. Consider

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

c is satisfiable iff c' is satisfiable

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

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

$$(x_1 \vee x_2 \vee \neg x_3 \vee x_4 \vee \neg x_5)$$

$\Downarrow$

$$(x_1 \vee x_2 \vee \neg x_3 \vee u)(x_4 \vee \neg x_5 \vee \neg u)$$

An Example

Example

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

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

An Example

Example

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

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

An Example

Example

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

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

An Example

Example

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

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

Overall Reduction Algorithm

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

Correctness (informal)

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

What about 2-SAT?

2-SAT can be solved in polynomial time!

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

3-SAT $\leq_P$ Independent Set

Input Given a 3-CNF formula φ

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

3-SAT $\leq_P$ Independent Set

Input Given a 3-CNF formula φ

Goal Construct a graph G_φ and number k such that G_φ has an independent set of size k iff φ is satisfiable. G_φ should be constructible in time polynomial in size of φ

3-SAT $\leq_P$ Independent Set

Input Given a 3-CNF formula φ

Goal Construct a graph G_φ and number k such that G_φ has an independent set of size k iff φ is satisfiable. G_φ should be constructible in time polynomial in size of φ

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

Interpreting 3-SAT

There are two ways to think about 3-SAT

Interpreting 3-SAT

There are two ways to think about 3-SAT

- 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 3-SAT

There are two ways to think about 3-SAT

- 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

Interpreting 3-SAT

There are two ways to think about 3-SAT

- 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 3-SAT to construct the reduction.

The Reduction

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

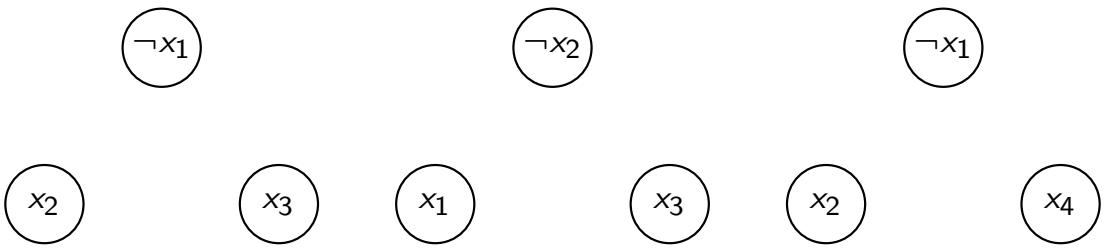

Figure: Graph for $\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

The Reduction

- 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

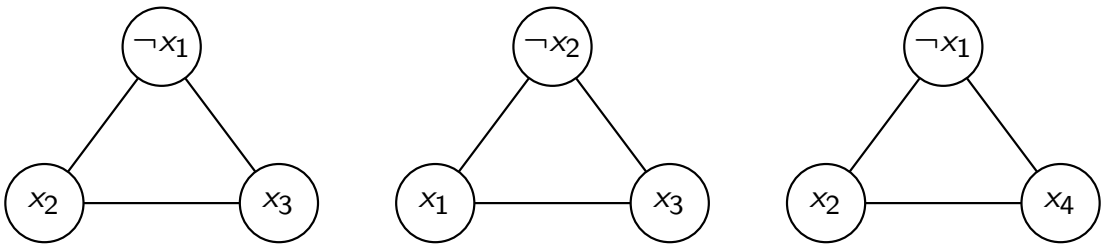

Figure: Graph for $\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

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

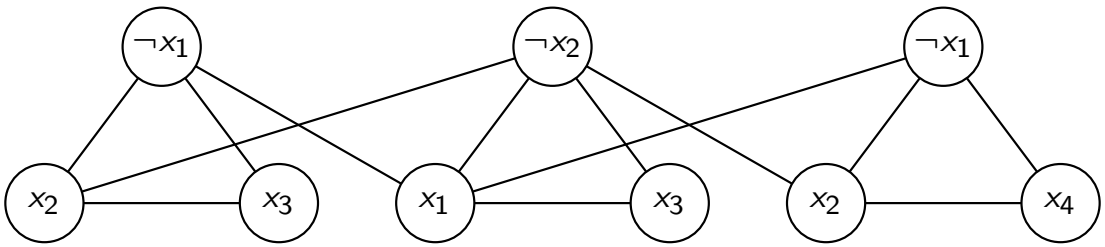

Figure: Graph for $\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

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

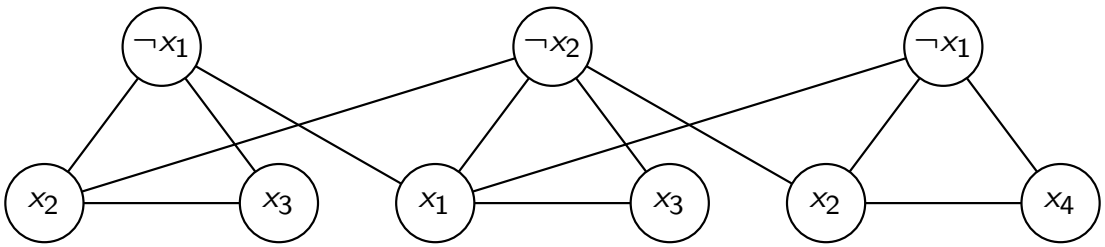

Figure: Graph for $\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

Correctness

Proposition

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

Correctness

Proposition

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

Proof.

Correctness

Proposition

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

Proof.

$\Rightarrow$ Let a be the truth assignment satisfying φ

- Pick one of the vertices, corresponding to true literals under a, from each triangle. This is an independent set of the appropriate size $\square$

Correctness (contd)

Proposition

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

Proof.

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

- S must contain exactly one vertex from each clause
- S cannot contain vertices labelled by conflicting clauses
- Thus, it is possible to obtain a truth assignment that makes in the literals in S true; such an assignment satisfies one literal in every clause $\square$

Transitivity of Reductions

$X \leq_P Y$ 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

Recap . . .

Problems

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

Recap . . .

Problems

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

Relationship

3-SAT $\leq_P$ Independent Set

Recap . . .

Problems

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

Relationship

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

Recap . . .

Problems

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

Relationship

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

Recap . . .

Problems

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

Relationship

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

Problems and Algorithms: Formal Approach

Decision Problems

Problems and Algorithms: Formal Approach

Decision Problems

- Problem Instance: Binary string s, with size $|s|$

Problems and Algorithms: Formal Approach

Decision Problems

- Problem Instance: Binary string s, with size $|s|$
- 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.

Problems and Algorithms: Formal Approach

Decision Problems

- Problem Instance: Binary string s, with size $|s|$
- 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

Problems and Algorithms: Formal Approach

Decision Problems

- Problem Instance: Binary string s, with size $|s|$
- 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

- A is an algorithm for problem X if $A(s) =$ "yes" iff $s \in X$

Problems and Algorithms: Formal Approach

Decision Problems

- Problem Instance: Binary string s, with size $|s|$
- 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

- A is an algorithm for problem X if $A(s) =$ "yes" iff $s \in X$
- 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 P) is the class of all (decision) problems that have an algorithm that solves it in polynomial time

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?

Efficiency Hypothesis

A problem X has an efficient algorithm iff $X \in P$, that is 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
- 3-SAT

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 I_X of X there is a proof/certificate/solution that is of length 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:

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

Examples:

- SAT formulat φ: proof is a satisfying assignment
- Independent Set in graph G and k: a subset S of vertices

Certifiers

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

Certifiers

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

Efficient Certifier

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

Example: Independent Set

- Problem: Does $G = (V, E)$ have an independent set of size $\geq k$?

Example: Independent Set

- Problem: Does $G = (V, E)$ have an independent set of size $\geq k$?
  - Certificate: Set $S \subseteq V$

Example: Independent Set

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

Example: Vertex Cover

- Problem: Does G have a vertex cover of size $\leq k$?

Example: Vertex Cover

- Problem: Does G have a vertex cover of size $\leq k$?
  - Certificate: $S \subseteq V$

Example: Vertex Cover

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

Example: SAT

- Problem: Does formula φ have a satisfying truth assignment?

Example: SAT

- Problem: Does formula φ have a satisfying truth assignment?
  - Certificate: Assignment a of 0/1 values to each variable

Example: SAT

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

Example:Composites

- Problem: Is number s a composite?

Example:Composites

- Problem: Is number s a composite?
  - Certificate: A factor $t \leq s$ such that $t \neq 1$ and $t \neq s$

Example:Composites

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

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, 3-SAT, 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

$P \subseteq NP$

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

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

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

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"

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" □

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

$NP \subseteq EXP$

NP versus *EXP*

Proposition

$NP \subseteq EXP$

Proof.

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

NP versus *EXP*

Proposition

$NP \subseteq EXP$

Proof.

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

- For every t, with $|t| \leq p(|s|)$ run $C(s, t)$; answer "yes" if any one of these calls returns "yes"

NP versus *EXP*

Proposition

$NP \subseteq EXP$

Proof.

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

- For every t, with $|t| \leq p(|s|)$ run $C(s, t)$; answer "yes" if any one of these calls returns "yes"
- The above algorithm correctly solves X (exercise)

NP versus *EXP*

Proposition

$NP \subseteq EXP$

Proof.

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

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

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 $P = NP$ ...

- Many important optimization problems can be solved efficiently

If $P = NP$ ...

- Many important optimization problems can be solved efficiently
- The RSA cryptosystem can be broken

If $P = NP$ ...

- Many important optimization problems can be solved efficiently
- The RSA cryptosystem can be broken
- No security on the web

If $P = NP$ ...

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

If $P = NP$ ...

- 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 Millenium Prize Problem. You can win a million dollars in addition to a Turing award and major fame!