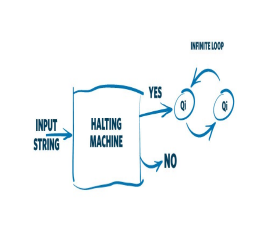
Introduction to Algorithms
Lecture 16 Computational Complexity

Xue Chen
xuechen1989@ustc.edu.cn
2024 spring in

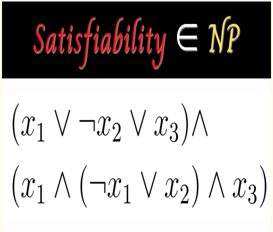
Outline

Overview

1. We have discussed various methods to design efficient algorithms
2. Many problems do not have efficient algorithms, e.g.



(a) HALTING PROBLEM

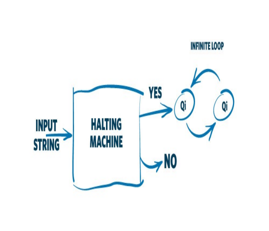


(b) SATISFIABILITY

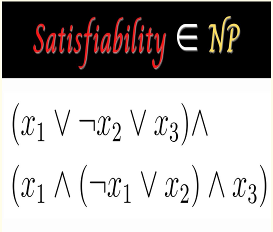
(c) GO GAME PROBLEM

Overview

1. We have discussed various methods to design efficient algorithms
2. Many problems do not have efficient algorithms, e.g.



(d) HALTING PROBLEM



(e) SATISFIABILITY

(f) GO GAME PROBLEM

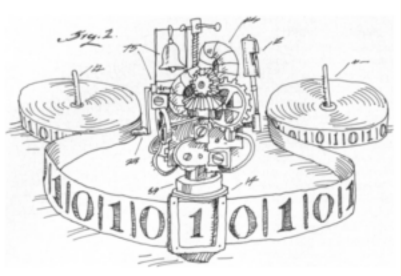
Computational Complexity

Study computational efficiency of problems:

1. Can computers solve it?
2. How long? Does it have efficient (poly-time) algorithms?

History

1. Turing defined the math model of modern computer around 1936

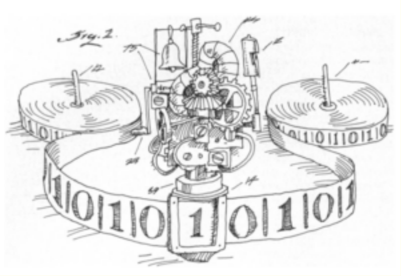
   

   — called Turing Machine

2. Basically, Turing Machine is an automaton with a memory tape

History

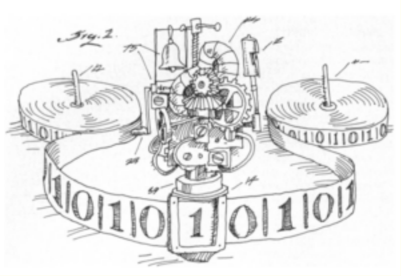
1. Turing defined the math model of modern computer around 1936

   — called Turing Machine
2. Basically, Turing Machine is an automaton with a memory tape
3. Turing-Church Theorem/Thesis: Any physical computation can be simulated by a Turing machine (with poly-time overhead)

History

1. Turing defined the math model of modern computer around 1936

   — called Turing Machine
2. Basically, Turing Machine is an automaton with a memory tape
3. Turing-Church Theorem/Thesis: Any physical computation can be simulated by a Turing machine (with poly-time overhead)
4. ☺Turing machine is simple enough to argue the limitation of modern computers

Turing Machines

An informal introduction:

1. Consider it as a low-level programming language — even lower than assembly language
2. A fixed number (say $\leqslant 100$) of instructions
3. Its description has a fixed number of lines/instructions
4. Its input is all $\{0, 1\}$-strings, i.e., $\{0, 1\}^*$
5. Its output is $\{0, 1\}$
6. Unbound memory space — different from automata

An Assembly Language Program

```
;
; Program to multiply a number by the constant 6
;
        .ORIG   x3050
        LD      R1, SIX
        LD      R2, NUMBER
        AND     R3, R3, #0      ; Clear R3.  It will
                                ; contain the product.
; The inner loop
;
AGAIN   ADD     R3, R3, R2
        ADD     R1, R1, #-1     ; R1 keeps track of
        BRp     AGAIN           ; the iteration.
;
        HALT
;
NUMBER  .BLKW   1
SIX     .FILL   x0006
;
        .END
```

Math Models

Binary Encoding of Problems — Languages

1. Only consider decision problems for now: For an instance/input I, the output is either 1 (means YES) or 0 (means NO)
2. Fix a decision problem Q, encode the instance I as a binary string $enc(\mathrm{I})$ in $\{0,1\}^*$
3. Define the language of Q as $L_Q = \{enc(\mathrm{I}) : \mathrm{Q}(\mathrm{I}) = 1\}$

- Example 1 PRIME: For a binary number $t \in \{0,1\}^*$, output 1 iff it is a prime. $L_{\mathrm{PRIME}} = \left\{ t_0 \cdots t_n \in \{0,1\}^* \,\middle|\, \sum_{i=0}^{n} t_i \cdot 2^i \text{ is a prime} \right\} = \{10, 11, 101, 111, 1011, 1101, \dots\}$

Math Models

Binary Encoding of Problems — Languages

1. Only consider decision problems for now: For an instance/input I, the output is either 1 (means YES) or 0 (means NO)
2. Fix a decision problem Q, encode the instance I as a binary string $enc(\mathrm{I})$ in $\{0, 1\}^*$
3. Define the language of Q as $L_Q = \{enc(\mathrm{I}) : \mathrm{Q}(\mathrm{I}) = 1\}$

- Example 1 PRIME: For a binary number $t \in \{0, 1\}^*$, output 1 iff it is a prime. $L_{\mathrm{PRIME}} = \left\{ t_0 \cdots t_n \in \{0, 1\}^* \middle| \sum_{i=0}^{n} t_i \cdot 2^i \text{ is a prime} \right\} = \left\{10, 11, 101, 111, 1011, 1101, \dots \right\}$
- Example 2 CONNECTIVITY: For a graph G, output 1 iff G is a connected. $L_{\mathrm{CONN}} = \left\{ T \in \{0, 1\}^* \middle| T \text{ presents a connected graph} \right\}$

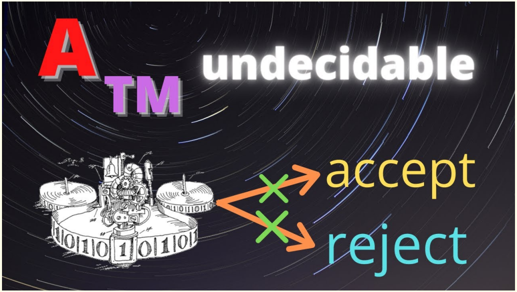
Overview

Why do we need Turing Machines and binary encodings?

Overview

Why do we need Turing Machines and binary encodings?

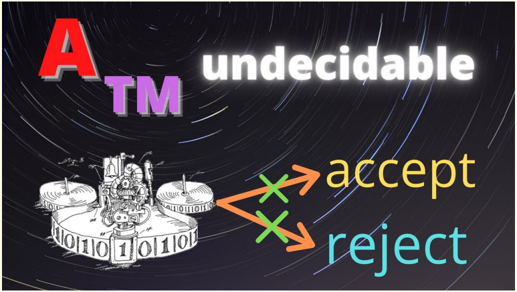
1. Use simple descriptions to prove that computers can not solve some problems (called undecidable) despite running time



Overview

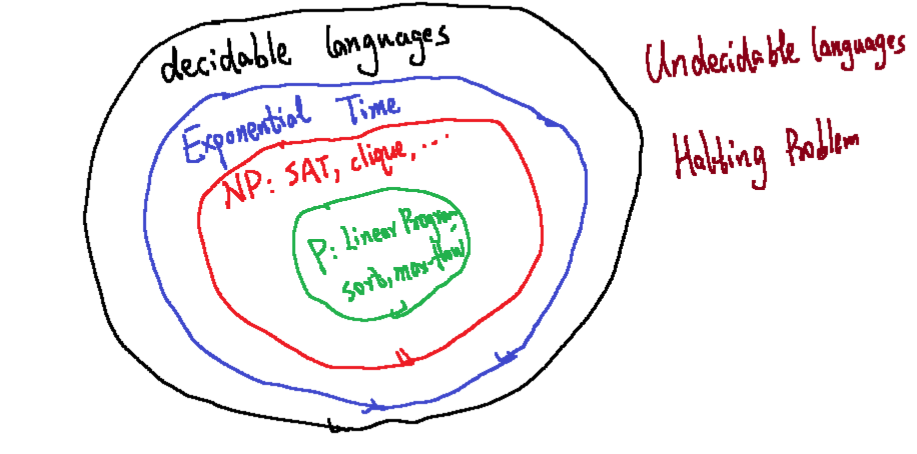
Why do we need Turing Machines and binary encodings?

1. Use simple descriptions to prove that computers can not solve some problems (called undecidable) despite running time



2. Use their binary encodings and executions on simple models to prove that some problem is the hardest in a class (called complete) like SAT problems in NP and LP in P

Computational Classes



Outline

Introduction

Main Question

Where is the limit of computation? — Are there problems/languages that can not be solved by computers (called undecidable)?

Yes — in fact, many problems

Introduction

Main Question

Where is the limit of computation? — Are there problems/languages that can not be solved by computers (called undecidable)?

Yes — in fact, many problems

1. Halting Problem: Given a program M and input x, will M halt on input x within a finite number of steps?

Introduction

Main Question

Where is the limit of computation? — Are there problems/languages that can not be solved by computers (called undecidable)?

Yes — in fact, many problems

1. Halting Problem: Given a program M and input x, will M halt on input x within a finite number of steps?
2. EMPTY: Given a program M, will it ever output YES?

Introduction

Main Question

Where is the limit of computation? — Are there problems/languages that can not be solved by computers (called undecidable)?

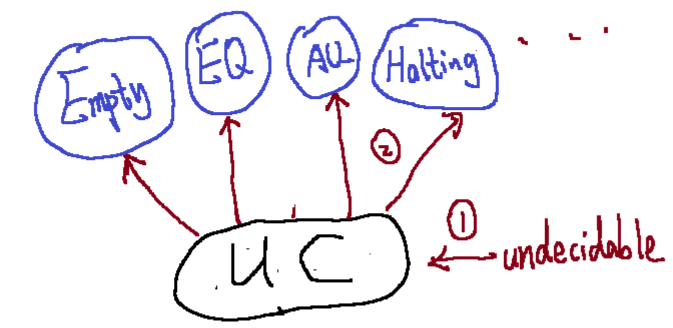
Yes — in fact, many problems

1. Halting Problem: Given a program M and input x, will M halt on input x within a finite number of steps?
2. EMPTY: Given a program M, will it ever output YES?
3. ALL: Given a program M, will it accept any input?
4. EQ: Given two programs M_1 and M_2, are they equivalent?

Road Map

To show they are undecidable,

1. Show Language UC is undecidable — the hardness of UC is the cornerstone of undecidable theory
2. Reduce UC to Halting and other problems



— if a computer solves any other prob, then it solves UC, which is impossible from (1)

1st Undecidable Language

Definition

1. Sort all programs from 1 to ∞: $M_1, M_2, \ldots$
2. Sort all inputs in $\{0,1\}^*$ from 1 to ∞: $\mathrm{I}_1, \mathrm{I}_2, \ldots$
3. For each $\alpha \in \{0,1\}^*$, consider the natural number corresponding to it

1st Undecidable Language

Definition

1. Sort all programs from 1 to ∞: $M_1, M_2, \ldots$
2. Sort all inputs in $\{0,1\}^*$ from 1 to ∞: $\mathrm{I}_1, \mathrm{I}_2, \ldots$
3. For each $\alpha \in \{0,1\}^*$, consider the natural number corresponding to it
4. Define $UC(\alpha) = 0$ only if $M_\alpha(\mathrm{I}_\alpha) = 1$; o.w. $UC(\alpha) = 1$ when $M_\alpha(\mathrm{I}_\alpha) = 0$ or never halts

Theorem 1

UC is undecidable

Halting Problem

Definition

Given a program M and input x, output 1 to indicate that M halts on input x within a finite number of steps

Halting Problem

Definition

Given a program M and input x, output 1 to indicate that M halts on input x within a finite number of steps

1. Reduce UC to Halting — direction is very important!
2. Halting is decidable $\Rightarrow$ UC is decidable

Halting Problem

Definition

Given a program M and input x, output 1 to indicate that M halts on input x within a finite number of steps

1. Reduce UC to Halting — direction is very important!
2. Halting is decidable $\Rightarrow$ UC is decidable
3. Since THM1, this is impossible. So Halting is undecidable.

EMPTY Problem

Definition

Given a program M, output 1 if $M(x) = 1$ for any $x \in \{0, 1\}^*$.

EMPTY Problem

Definition

Given a program M, output 1 if $M(x) = 1$ for any $x \in \{0, 1\}^*$.

1. We leave the rest problems in homework
2. One more question: Is this language decidable or not?

   Given a program M, an input x, and t, output 1 if M accepts x in t steps.

EMPTY Problem

Definition

Given a program M, output 1 if $M(x) = 1$ for any $x \in \{0, 1\}^*$.

1. We leave the rest problems in homework
2. One more question: Is this language decidable or not?

> Given a program M, an input x, and t, output 1 if M accepts x in t steps.

3. Next: What is the limit of **efficient computation**?

Outline

Intro

Definition

P is the class of all problems with a (deterministic) polynomial time algorithm — same as P := class of poly-time algorithms

Consider P captures efficient computation

Intro

Definition

P is the class of all problems with a (deterministic) polynomial time algorithm — same as $\mathrm{P} :=$ class of poly-time algorithms

Consider P captures efficient computation

1. Good News: Definition P is very robust — poly-time reductions keeps the total running in P

Intro

Definition

P is the class of all problems with a (deterministic) polynomial time algorithm — same as $\mathrm{P} :=$ class of poly-time algorithms

Consider P captures efficient computation

1. Good News: Definition P is very robust — poly-time reductions keeps the total running in P
2. Issue 1: How about randomized poly-time algorithms? Most researchers believe Class(randomized poly-time algorithms)=P ☺
3. Issue 2: Definition P is too strict — the algorithm runs in poly-time for all inputs. Average-case complexity ☺

Intro

Definition

P is the class of all problems with a (deterministic) polynomial time algorithm — same as $\mathrm{P} :=$ class of poly-time algorithms

Consider P captures efficient computation

1. Good News: Definition P is very robust — poly-time reductions keeps the total running in P
2. Issue 1: How about randomized poly-time algorithms? Most researchers believe Class(randomized poly-time algorithms)=P ☺
3. Issue 2: Definition P is too strict — the algorithm runs in poly-time for all inputs. Average-case complexity ☺
4. Issue 3: Lack of precision, is n^{10} efficient? ☹
5. Issue 4: How about quantum algorithms? Not sure whether general quantum computers are realizable ☹

Relation between Languages

Given two problems A and B, how to compare their hardness?

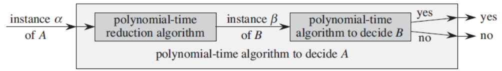
1. Prime is equivalent to composite
2. How to compare connectivity and set cover?

Relation between Languages

Given two problems A and B, how to compare their hardness?

1. Prime is equivalent to composite
2. How to compare connectivity and set cover?

Poly Time Reduction (a.k.a. Cook/Karp Reduction)



$A \leq_p B$: poly-time reduction from A to B

Requirements:

1. Reduction time is polynomial
2. Map Yes instance to Yes instance and No instance to No — the most technical part

— for now, $A \leqslant_p B$ means that B is harder than A; essentially, $B \in \mathrm{P} \Rightarrow A \in \mathrm{P}$

Complete problems

Complete Problems of a class

A problem P is complete in the class $\mathcal{C}$ only if (1) $P \in \mathcal{C}$; (2) any problem $Q \in \mathcal{C}$ has a reduction $Q \leqslant_p P$

— informally, P is the “hardest" problem in $\mathcal{C}$

1. If there is a poly-time algorithm of P, then every problem in $\mathcal{C}$ has poly-time algorithms

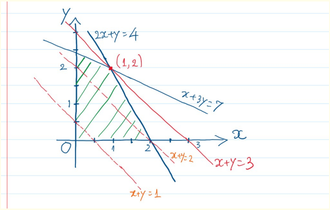
Complete problems

Complete Problems of a class

A problem P is complete in the class $\mathcal{C}$ only if (1) $P \in \mathcal{C}$; (2) any problem $Q \in \mathcal{C}$ has a reduction $Q \leqslant_p P$

— informally, P is the “hardest" problem in $\mathcal{C}$

1. If there is a poly-time algorithm of P, then every problem in $\mathcal{C}$ has poly-time algorithms
2. Example 1: Linear programming is complete in P



3. Another P-complete problem: Given a program M, input x, and string S, determine $M(x) = 1$ in $|S|$ steps or not

Outline

Background

There are many problems not in P:

Example

Given a program M, input x, and number N, determine $M(x) = 1$ in N steps or not

1. This problem is exponential-time complete.
2. Question: What's difference between the P-complete problem — Given M, x, and string S, determine $M(x) = 1$ in $|S|$ steps or not?

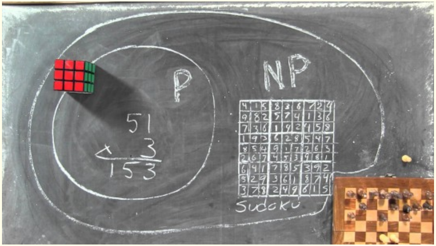
Background

There are many problems not in P:

Example

Given a program M, input x, and number N, determine $M(x) = 1$ in N steps or not

1. This problem is exponential-time complete.
2. Question: What's difference between the P-complete problem — Given M, x, and string S, determine $M(x) = 1$ in $|S|$ steps or not?
3. However, this problem is not very interesting
4. The most interesting class probably not in P is NP



What is NP?

The original definition is by non-deterministic Turing machine. We will consider the modern interpretation:

Definition of NP

A language/problem L is in NP only if $\exists$ an algorithm, called verifier, V such that

1. For any $x \in L$, $\exists$ a proof y s.t. $V(x, y) = 1$ in poly$(|x|)$ time — completeness
2. For any $x \notin L$, for any proof y, $V(x, y) = 0$ in poly$(|x|)$ time — soundness

1. $V(x, \cdot)$ is a non-deterministic algorithm but $V(x, y)$ is deterministic

What is NP?

The original definition is by non-deterministic Turing machine. We will consider the modern interpretation:

Definition of NP

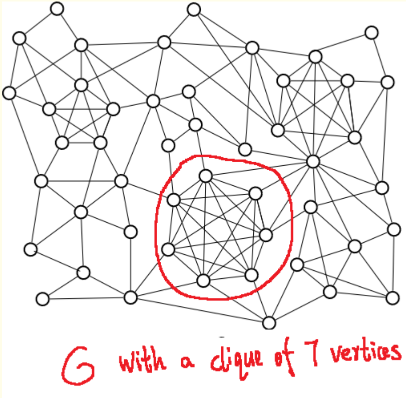
A language/problem L is in NP only if $\exists$ an algorithm, called verifier, V such that

1. For any $x \in L$, $\exists$ a proof y s.t. $V(x, y) = 1$ in poly$(|x|)$ time — completeness
2. For any $x \notin L$, for any proof y, $V(x, y) = 0$ in poly$(|x|)$ time — soundness

1. $V(x, \cdot)$ is a non-deterministic algorithm but $V(x, y)$ is deterministic
2. In Case 1, y depends on x
3. $|y| = \text{poly}(n)$ because V runs in poly-time
4. Example 1: $L_{\text{CLIQUE}} = \left\{(G, k) : \exists \text{ a clique of size } k \text{ in } G\right\}$

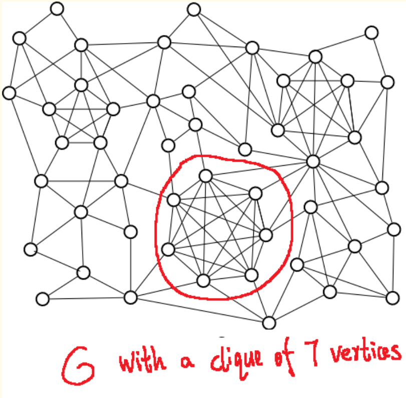
Example 1: Clique

1. y is a subset of k vertices in G
2. $V\Big((G,k),y\Big)$ checks all pairs in y are connected in G



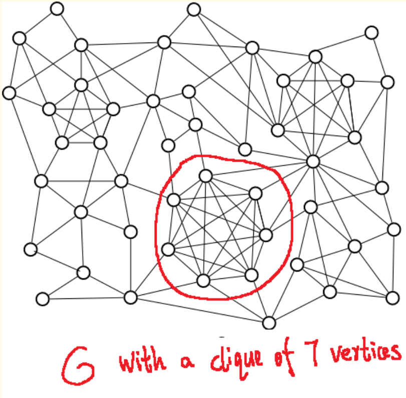
Example 1: Clique

1. y is a subset of k vertices in G
2. $V\Big((G,k),y\Big)$ checks all pairs in y are connected in G
3. Completeness: If $(G,k) \in L_{\text{CLIQUE}}$ — $\exists$ a k-clique in G, y encodes that subset



Example 1: Clique

1. y is a subset of k vertices in G
2. $V\Big((G,k),y\Big)$ checks all pairs in y are connected in G
3. Completeness: If $(G,k) \in L_{\text{CLIQUE}}$ — $\exists$ a k-clique in G, y encodes that subset
4. Soundness: If $(G,k) \notin L_{\text{CLIQUE}}$ — $\forall$ k-subset in G, V rejects it since it is not a clique



Example 2: SAT problem

1. Description: each instance Φ has n Boolean variables $x_1, \ldots, x_n$ and m clauses $C_1, \ldots, C_m$
2. $L_{\mathrm{SAT}} = \left\{ \Phi \text{ is satisfiable} : \exists \sigma \in \{T, F\}^n \text{ s.t. } C_1(\sigma) = C_2(\sigma) = \cdots = C_m(\sigma) = T \right\}$

$C_1: x_1 \vee x_2 \vee \overline{x_4} \vee \overline{x_n}$

$C_2: \overline{x_3} \vee \overline{x_{n-2}} \vee \overline{x_{n-1}} \vee x_n$

$C_3: x_5$

$C_4: x_6 \vee x_7$

$\vdots$

$C_m: x_1 \vee x_2 \vee x_3 \cdots \vee x_n$

Example 2: SAT problem

1. Description: each instance Φ has n Boolean variables $x_1, \ldots, x_n$ and m clauses $C_1, \ldots, C_m$
2. $L_{\mathrm{SAT}} = \Big\{ \Phi \text{ is satisfiable} : \exists \sigma \in \{T, F\}^n \text{ s.t. } C_1(\sigma) = C_2(\sigma) = \cdots = C_m(\sigma) = T \Big\}$
3. y is a string $\in \{T, F\}^n$ as the assignment on x
4. V verifies $C_1(y) = C_2(y) = \cdots = C_m(y) = T$

C_1: $x_1 \vee x_2 \vee \overline{x_4} \vee \overline{x_n}$

C_2: $\overline{x_3} \vee \overline{x_{n-2}} \vee \overline{x_{n-1}} \vee x_n$

C_3: x_5

C_4: $x_6 \vee x_7$

$\vdots$

C_m: $x_1 \vee x_2 \vee x_3 \cdots \vee x_n$

Example 2: SAT problem

1. Description: each instance Φ has n Boolean variables $x_1, \ldots, x_n$ and m clauses $C_1, \ldots, C_m$
2. $L_{\mathrm{SAT}} = \Big\{ \Phi \text{ is satisfiable} : \exists \sigma \in \{T, F\}^n \text{ s.t. } C_1(\sigma) = C_2(\sigma) = \cdots = C_m(\sigma) = T \Big\}$
3. y is a string $\in \{T, F\}^n$ as the assignment on x
4. V verifies $C_1(y) = C_2(y) = \cdots = C_m(y) = T$
5. Equivalent to define Φ as a CNF $C_1 \wedge C_2 \wedge C_3 \wedge \cdots \wedge C_m$

$C_1: x_1 \vee x_2 \vee \overline{x_4} \vee \overline{x_n}$

$C_2: \overline{x_3} \vee \overline{x_{n-2}} \vee \overline{x_{n-1}} \vee x_n$

$C_3: x_5$

$C_4: x_6 \vee x_7$

$\vdots$

$C_m: x_1 \vee x_2 \vee x_3 \cdots \vee x_n$

P vs NP

NP captures many interesting problems and is probably $\not\subset$P

1. P$\subseteq$ NP: Why?

P vs NP

NP captures many interesting problems and is probably $\not\subset$P

1. P$\subseteq$ NP: Why?
2. We believe P$\neq$NP: Intuitively, one can verify a proof $\not\Rightarrow$ One can generate that proof — one million dollar problem

P vs NP

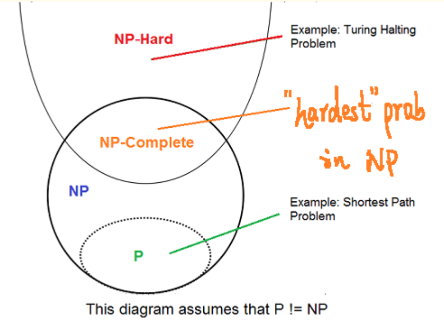
NP captures many interesting problems and is probably $\not\subset$P

1. P$\subseteq$ NP: Why?
2. We believe P$\neq$NP: Intuitively, one can verify a proof $\not\Rightarrow$ One can generate that proof — one million dollar problem
3. Next: How to prove P$\neq$NP formally?
   Find a problem and prove it does not have poly-time algorithms

P vs NP

NP captures many interesting problems and is probably $\not\subset$P

1. P$\subseteq$ NP: Why?
2. We believe P$\neq$NP: Intuitively, one can verify a proof $\not\Rightarrow$ One can generate that proof — one million dollar problem
3. Next: How to prove P$\neq$NP formally?
   Find a problem and prove it does not have poly-time algorithms



4. Which problem? — NP-complete

Outline

NP-complete

NP-complete: “Hardest" problems in NP under poly-time reduction

Definition

A problem L is NP-complete iff (1) $L \in \mathrm{NP}$; (2) for any $Q \in \mathrm{NP}$, $Q \leqslant_p L$.

NP-complete

NP-complete: "Hardest" problems in NP under poly-time reduction

Definition

A problem L is NP-complete iff (1) $L \in \mathrm{NP}$; (2) for any $Q \in \mathrm{NP}$, $Q \leqslant_p L$.

1. THM 34.4 in CLRS: $\mathrm{P} \neq \mathrm{NP} \Leftrightarrow \forall$ NP-complete problem $L \notin \mathrm{P}$
2. A concrete plan to show $\mathrm{P} \neq \mathrm{NP}$: (1) Find an NP-complete problem L; (2) Prove that L does not admit poly-time algorithms

NP-complete

NP-complete: "Hardest" problems in NP under poly-time reduction

Definition

A problem L is NP-complete iff (1) $L \in \mathrm{NP}$; (2) for any $Q \in \mathrm{NP}$, $Q \leqslant_p L$.

1. THM 34.4 in CLRS: $\mathrm{P} \neq \mathrm{NP} \Leftrightarrow \forall$ NP-complete problem $L \notin \mathrm{P}$
2. A concrete plan to show $\mathrm{P} \neq \mathrm{NP}$: (1) Find an NP-complete problem L; (2) Prove that L does not admit poly-time algorithms
3. Good news: A long list of natural NP-complete problems: Clique, SAT, independent set, . . .
4. Bad news: Extremely difficult to show: a problem *does not* have poly-time algorithms

NP-complete

NP-complete: “Hardest" problems in NP under poly-time reduction

Definition

A problem L is NP-complete iff (1) $L \in \mathrm{NP}$; (2) for any $Q \in \mathrm{NP}$, $Q \leqslant_p L$.

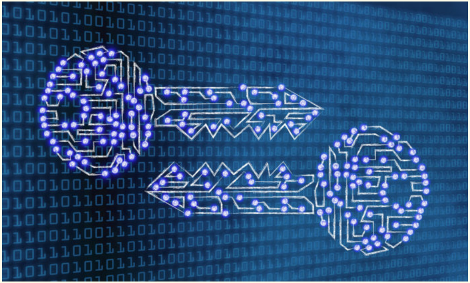
1. THM 34.4 in CLRS: $\mathrm{P} \neq \mathrm{NP} \Leftrightarrow \forall$ NP-complete problem $L \notin \mathrm{P}$
2. A concrete plan to show $\mathrm{P} \neq \mathrm{NP}$: (1) Find an NP-complete problem L; (2) Prove that L does not admit poly-time algorithms
3. Good news: A long list of natural NP-complete problems: Clique, SAT, independent set, ...
4. Bad news: Extremely difficult to show: a problem *does not* have poly-time algorithms
5. Since $\mathrm{P} \neq \mathrm{NP}$ is very plausible, another way: problem Q is NP-complete means $Q \notin \mathrm{P}$s.t. Q does not have a poly-time algorithm

More about NP

1. If we can not design poly-time algorithms for Q, showing Q is NP-complete gives an excuse

More about NP

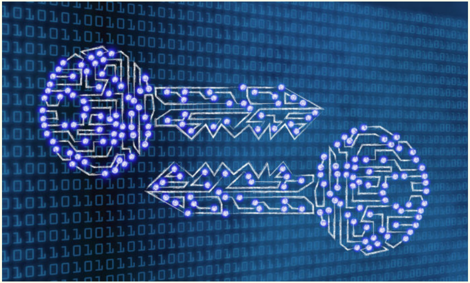
1. If we can not design poly-time algorithms for Q, showing Q is NP-complete gives an excuse
2. Do not feel frustrated for P$\neq$NP— that's a good news for cryptography



In fact, cryptography assumptions are much stronger than P$\neq$NP

More about NP

1. If we can not design poly-time algorithms for Q, showing Q is NP-complete gives an excuse
2. Do not feel frustrated for P$\neq$NP— that's a good news for cryptography



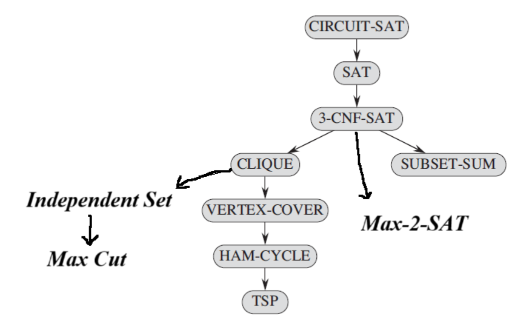
   In fact, cryptography assumptions are much stronger than P$\neq$NP
3. Surprisingly, very few hard problems are presumably not NP-complete: graph isomorphism, factoring, . . .;

NP-compelete

Next question: How to show a problem is NP-complete?

Recall definition

A problem L is NP-complete iff (1) $L \in \mathrm{NP}$; (2) for any $Q \in \mathrm{NP}$, $Q \leqslant_p L$.



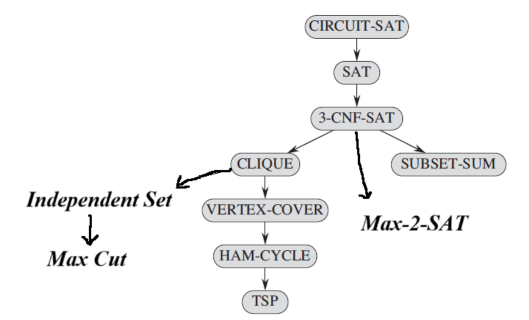
NP-compelete

Next question: How to show a problem is NP-complete?

Recall definition

A problem L is NP-complete iff (1) $L \in \mathrm{NP}$; (2) for any $Q \in \mathrm{NP}$, $Q \leqslant_p L$.

(1) is easy. But (2) is challenging — Enough to show $R \leqslant_R Q$ for some $R \in$NP-complete.



NP-compelete

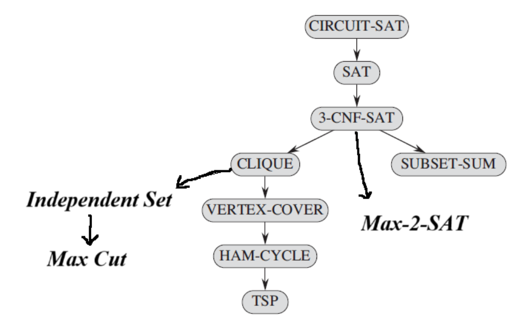
Next question: How to show a problem is NP-complete?

Recall definition

A problem L is NP-complete iff (1) $L \in \mathrm{NP}$; (2) for any $Q \in \mathrm{NP}$, $Q \leqslant_p L$.

(1) is easy. But (2) is challenging — Enough to show $R \leqslant_R Q$ for some $R \in$NP-complete.

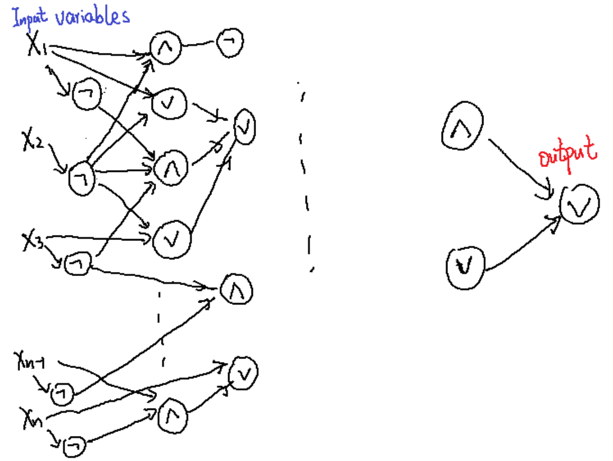
1. But which problem shall we begin with?
2. Roadmap of NPC problems



CIRCUIT-SAT

Description

$L_{\text{CIRCUIT-SAT}} = \{C : C \text{ is satisfiable}\}$ where circuit C is a DAG with AND $\wedge$, OR $\vee$, NOT $\neg$ gates and n input variables $x_1, \ldots, x_n$



CIRCUIT-SAT (II)

Theorem 34.7

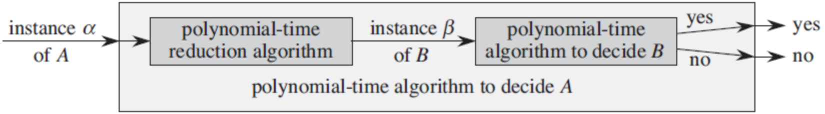
$L_{\mathrm{CIRCUIT-SAT}}$ is NP-complete.

CIRCUIT-SAT (II)

Theorem 34.7

$L_{\mathrm{CIRCUIT-SAT}}$ is NP-complete.

1. Lemma 34.5 in CLRS: $L_{\mathrm{CIRCUIT-SAT}}$ is in NP



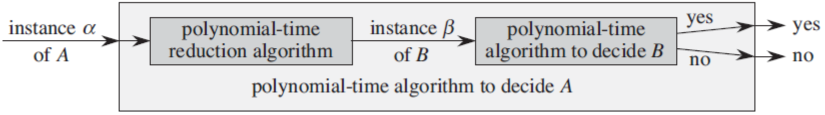
$A \leq_p B$: poly-time reduction from A to B

CIRCUIT-SAT (II)

Theorem 34.7

$L_{\text{CIRCUIT-SAT}}$ is NP-complete.

1. Lemma 34.5 in CLRS: $L_{\text{CIRCUIT-SAT}}$ is in NP

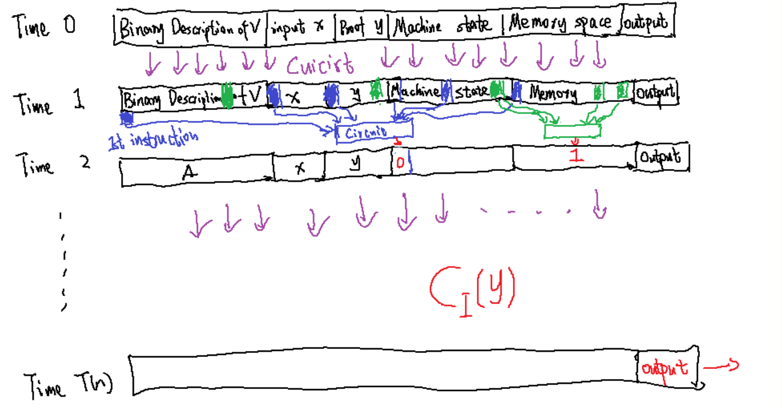


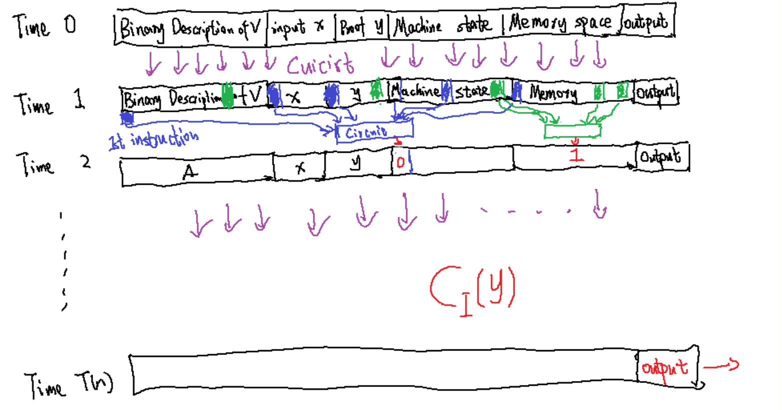
$A \leq_p B$: poly-time reduction from A to B

2. Lemma 34.6 in CLRS: $\forall Q \in \mathrm{NP}$, $Q \leqslant_p L_{\text{CIRCUIT-SAT}}$ — roughly, fix V and x then treat proof y as Boolean variables such that execution on the hardware level is a circuit

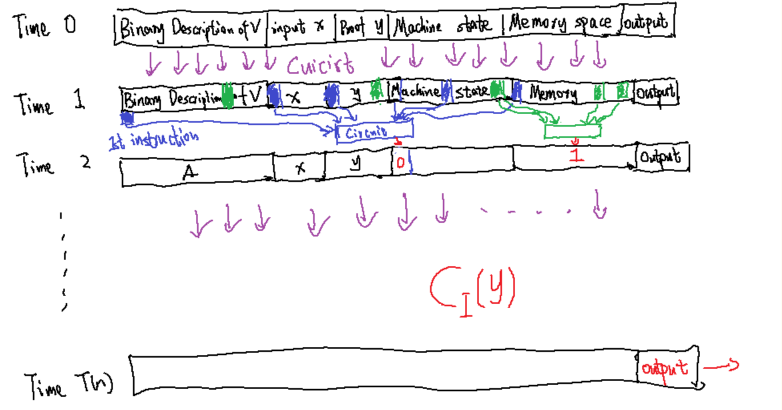
1. To show $Q \leqslant_p L_{\mathrm{CIRCUIT-SAT}}$, consider Q's verifier V such that $Q(x) = 1$ iff $\exists y$ s.t. $V(x, y) = 1$.
2. Our goal is to show a reduction that given V and x, outputs a circuit $C_{V,x}(y)$ s.t. $C_{V,x}(y) = V(x, y)$

1. To show $Q \leqslant_p L_{\mathrm{CIRCUIT-SAT}}$, consider Q's verifier V such that $Q(x) = 1$ iff $\exists y$ s.t. $V(x, y) = 1$.
2. Our goal is to show a reduction that given V and x, outputs a circuit $C_{V,x}(y)$ s.t. $C_{V,x}(y) = V(x, y)$
3. In one sentence, $C_{V,x}(y)$ is the circuit computing $V(I, y)$ in hardware level

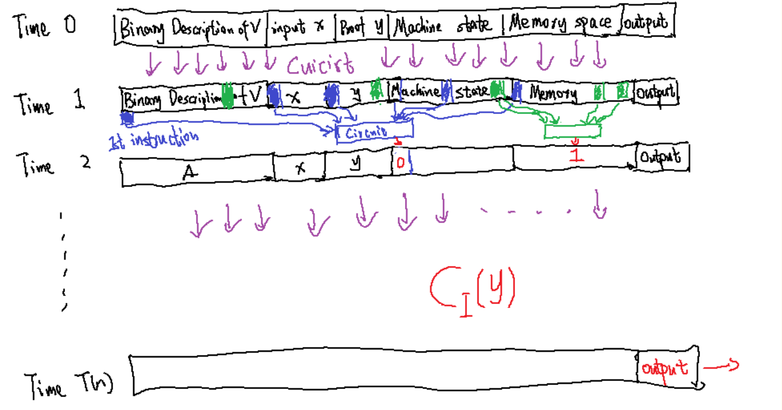




1. Because the time and memory space of V is $\text{poly}(n)$, its size is $\text{poly}(n)$

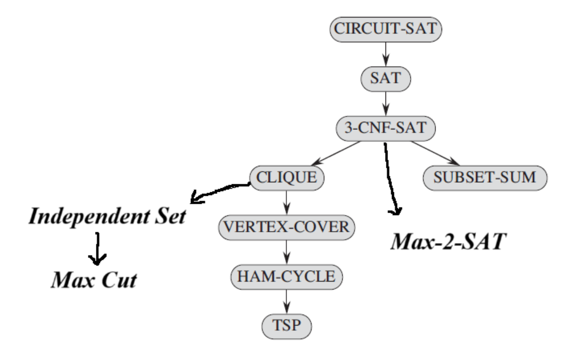


1. Because the time and memory space of V is $\mathrm{poly}(n)$, its size is $\mathrm{poly}(n)$
2. We can compute $C_{V,x}(y)$ in $\mathrm{poly}(n)$ time



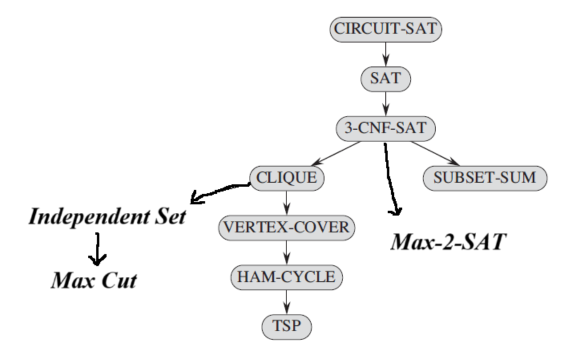
1. Because the time and memory space of V is $\text{poly}(n)$, its size is $\text{poly}(n)$
2. We can compute $C_{V,x}(y)$ in $\text{poly}(n)$ time
3. Because $C_{V,x}(y) = V(x,y)$ for any y, $x \in L_Q$ iff $C_{V,x}$ is satisfiable ☺

Road Map



1. CIRCUIT-SAT is the 1st non-trivial NP-complete problem

Road Map



1. CIRCUIT-SAT is the 1st non-trivial NP-complete problem
2. Next show SAT is NP-complete:

> Each instance Φ has n Boolean variables $x_1, \ldots, x_n$ and m clauses $C_1, \ldots, C_m$ such that $L_{\mathrm{SAT}} = \{\Phi : \exists \sigma \in \{T, F\}^n \text{ s.t. } C_1(\sigma) = C_2(\sigma) = \cdots = C_m(\sigma) = T\}$

SAT Problem

Theorem 34.9 in CLRS

SAT is NP-complete

SAT Problem

Theorem 34.9 in CLRS

SAT is NP-complete

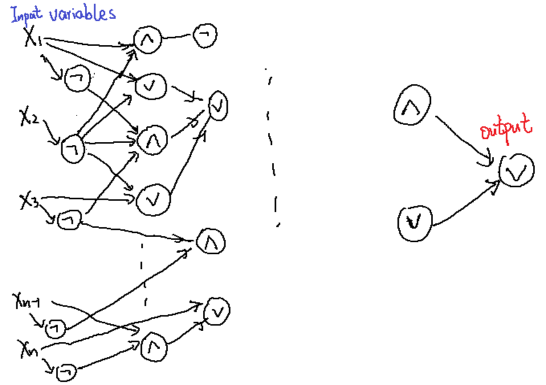
1. SAT is in NP

SAT Problem

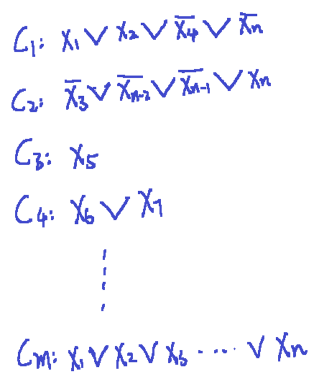
Theorem 34.9 in CLRS

SAT is NP-complete

1. SAT is in NP
2. Reduce CIRCUIT-SAT $\rightarrow$ SAT

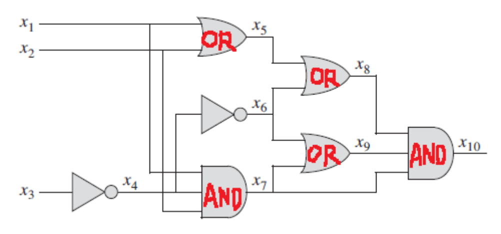


$\Rightarrow$



Basic Idea

1. Assign a Boolean variable to each gate to denote its evaluation:



2. Question: How to make sure $x_4 = \overline{x_3}$ and $x_5 = x_1 \vee x_2$?
3. Completeness & Soundness: $\exists x$ s.t. $C(x) = \mathit{True} \iff \exists \sigma$ s.t. all clauses in $\Phi(\sigma)$ are true

3-SAT

Definition: Each instance Φ has n Boolean variables $x_1, \ldots, x_n$ and m clauses $C_1, \ldots, C_m$ of width 3 such that
$L_{\mathrm{3SAT}} = \{\Phi : \exists \sigma \in \{T, F\}^n$ s.t. $C_1(\sigma) = C_2(\sigma) = \cdots = C_m(\sigma) = T\}$

Theorem 34.10 in CLRS

3SAT (CNF) is NP-complete.

3-SAT

Definition: Each instance Φ has n Boolean variables $x_1, \ldots, x_n$ and m clauses $C_1, \ldots, C_m$ of width 3 such that
$L_{\mathrm{3SAT}} = \{\Phi : \exists \sigma \in \{T, F\}^n \text{ s.t. } C_1(\sigma) = C_2(\sigma) = \cdots = C_m(\sigma) = T\}$

Theorem 34.10 in CLRS

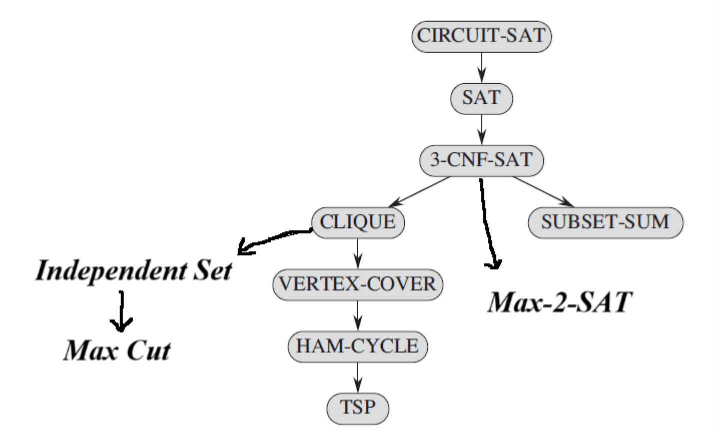
3SAT (CNF) is NP-complete.

Proof Sketch:

1. 3SAT is in NP
2. Reduce SAT to 3SAT

Outline

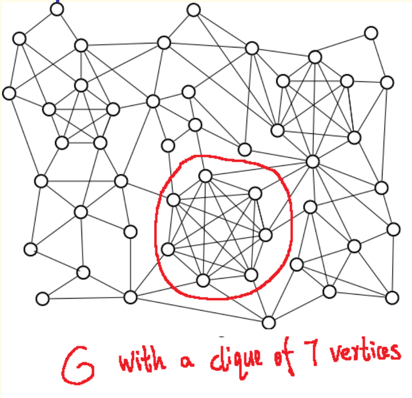
Overview



To show a problem Q is NP-complete

1. Easy part: $Q \in \mathrm{NP}$
2. Tricky part: Reduce a NP-complete problem (known ones like CIRCUIT-SAT, SAT, 3SAT) to Q

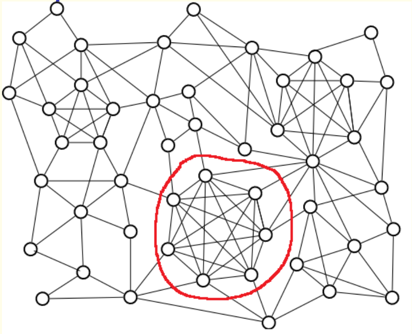
Clique



Theorem 34.11 in CLRS

$L_{\mathrm{CLIQUE}} := \{(G, k) : \exists$ a k-clique in $G\}$ is NP-complete.

Clique



Theorem 34.11 in CLRS

$L_{\text{CLIQUE}} := \{(G, k) : \exists$ a k-clique in $G\}$ is NP-complete.

Proof Sketch:

1. It is in NP.
2. Reduce 3SAT to CLIQUE.

Discussion

1. Gadgets are the key in reductions — more gadgets next!
2. We only discussed decision problem so far: Given (G, k), output YES to indicate $\exists$ a k-clique in G.

Discussion

1. Gadgets are the key in reductions — more gadgets next!
2. We only discussed decision problem so far: Given (G, k), output YES to indicate $\exists$ a k-clique in G.
3. How about optimization problem: Given G, find the largest clique in G?
4. Question: Is the optimization problem NP-complete?

Discussion

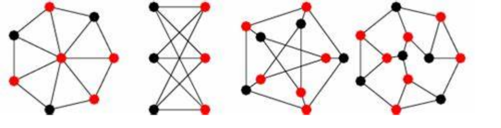
1. Gadgets are the key in reductions — more gadgets next!
2. We only discussed decision problem so far: Given (G, k), output YES to indicate $\exists$ a k-clique in G.
3. How about optimization problem: Given G, find the largest clique in G?
4. Question: Is the optimization problem NP-complete?
   — While it is harder than the decision problem, can not prove it is in NP ☹

Discussion

1. Gadgets are the key in reductions — more gadgets next!
2. We only discussed decision problem so far: Given (G, k), output YES to indicate $\exists$ a k-clique in G.
3. How about optimization problem: Given G, find the largest clique in G?
4. Question: Is the optimization problem NP-complete?
   — While it is harder than the decision problem, can not prove it is in NP ☹
5. Definition: For a problem P, if $Q \leqslant_p P$ for any $Q \in \mathrm{NP}$, call p NP-hard
6. Examples of NP-hard: Largest clique in G; find an assignment to maximize # clauses for 3SAT

Set Cover

A vertex cover of $G = (V, E)$ is a subset $S \subseteq V$ such that every edge has at least one endpoint in S

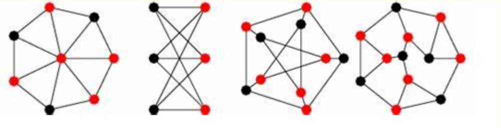


Theorem 34.12 in CLRS

$L_{\text{Vertex}-\text{Cover}} = \{(G, k) : G \text{ has a vertex cover of size } k\}$ is NP-complete

Set Cover

A vertex cover of $G = (V, E)$ is a subset $S \subseteq V$ such that every edge has at least one endpoint in S



Theorem 34.12 in CLRS

$L_{\mathrm{Vertex-Cover}} = \{(G, k) : G$ has a vertex cover of size $k\}$ is NP-complete

1. $L_{\mathrm{Vertex-Cover}}$ is in NP
2. Reduce CLIQUE to Vertex-Cover

Subset Sum

Theorem 34.15 in CLRS

$L_{\mathrm{Subset-Sum}} = \left\{ \langle S = (s_1, \ldots, s_n), t \rangle : \exists S' \subseteq S \text{ with summation } = t \right\}$ is NP-complete

1. $L_{\mathrm{Subset-Sum}}$ is in NP
2. Reduce 3SAT to Subset-Sum

Reduction

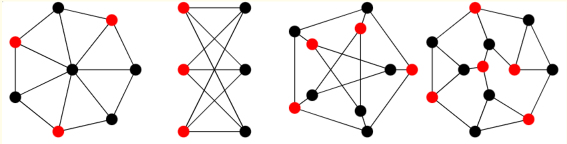
$C_1 = x_1 \vee \overline{x_2} \vee \overline{x_3}$, $C_2 = \overline{x_1} \vee \overline{x_2} \vee \overline{x_3}$, $C_3 = \overline{x_1} \vee \overline{x_2} \vee x_3$, $C_4 = x_1 \vee x_2 \vee x_3$

		x_1	x_2	x_3	C_1	C_2	C_3	C_4
v_1	=	1	0	0	1	0	0	1
v_1'	=	1	0	0	0	1	1	0
v_2	=	0	1	0	0	0	0	1
v_2'	=	0	1	0	1	1	1	0
v_3	=	0	0	1	0	0	1	1
v_3'	=	0	0	1	1	1	0	0
s_1	=	0	0	0	1	0	0	0
s_1'	=	0	0	0	2	0	0	0
s_2	=	0	0	0	0	1	0	0
s_2'	=	0	0	0	0	2	0	0
s_3	=	0	0	0	0	0	1	0
s_3'	=	0	0	0	0	0	2	0
s_4	=	0	0	0	0	0	0	1
s_4'	=	0	0	0	0	0	0	2
t	=	1	1	1	4	4	4	4

More NP-hard Problems

Max Independent Set

Given a graph G, it is NP-hard to find the maximal independent set.



Which problem shall we reduce from?

Max Cut

Theorem

Given a graph G, it is NP-hard to find a max cut.

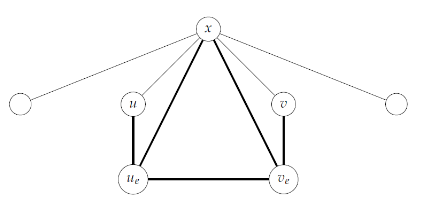
Reduction from Max Independent Set:

Max Cut

Theorem

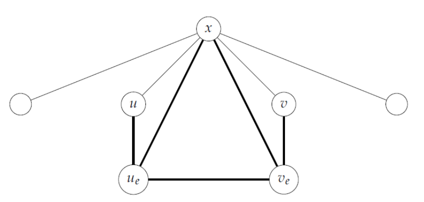
Given a graph G, it is NP-hard to find a max cut.

Reduction from Max Independent Set: Given $G = (V, E)$ from MIS, construct $G' = (V', E')$ for Max Cut:



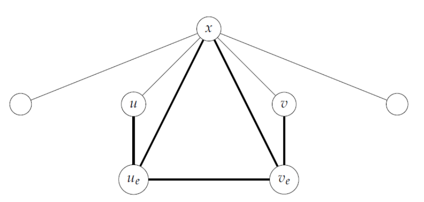
$V' = x \cup V \cup \{u_e, v_e : \forall e \in E\}$
$E' = \{(x, v) : \forall v \in V\} \cup \{e - \text{gadget} : \forall e \in E\}$
where (u, v)-gadget $:= (x, u_e), (x, v_e), (u, u_e), (v, v_e), (u_e, v_e)$

Analysis



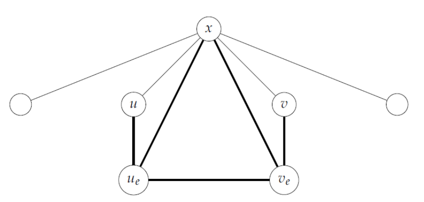
1. $\Rightarrow$ If $I \subset V$ is independent of size k, define a cut $S := I$ then for each (u, v)-gadget, add u_e to S if $v \in S$ or add v_e to S if $u \in S$
2. OBS: cut S is $|I| + 4 \cdot |E|$

Analysis



1. $\Rightarrow$ If $I \subset V$ is independent of size k, define a cut $S := I$ then for each (u, v)-gadget, add u_e to S if $v \in S$ or add v_e to S if $u \in S$
2. OBS: cut S is $|I| + 4 \cdot |E|$
3. $\Leftarrow$ Given cut $S \subset V'$ of $k + 4 \cdot |E|$ edges, assume $x \notin S$
4. Consider $I := S \cap V$ — suppose $\exists m(I)$ edges inside I s.t. it is not independent

Analysis



1. $\Rightarrow$ If $I \subset V$ is independent of size k, define a cut $S := I$ then for each (u, v)-gadget, add u_e to S if $v \in S$ or add v_e to S if $u \in S$
2. OBS: cut S is $|I| + 4 \cdot |E|$
3. $\Leftarrow$ Given cut $S \subset V'$ of $k + 4 \cdot |E|$ edges, assume $x \notin S$
4. Consider $I := S \cap V$ — suppose $\exists m(I)$ edges inside I s.t. it is not independent
5. For a gadget of $e = (u, v)$, if both u and $v \in I$, S cuts at most 3 edges in this gadget. Otherwise, S cuts 4 edges.
6. $E(S, S') \leqslant |I| + 4 \cdot |E| - |m(I)| \Rightarrow$ deleting one point in each $m(I)$ provides an independent set of size $|I| - |m(I)| \geqslant k$

2-SAT

2-SAT

1. n Boolean variables $x_1, \ldots, x_n \in \{T, F\}$ and m clauses of width 2 like $x_1 \vee \overline{x_2}, \overline{x_3} \vee \overline{x_4}, x_2 \vee x_3, \ldots$
2. Find an assignment to satisfy all clauses.

Different from 3-SAT, 2-SAT is in P; however, MAX-2-SAT is NP-hard

1. Why 2-SAT $\in \mathrm{P}$?

2-SAT

2-SAT

1. n Boolean variables $x_1, \ldots, x_n \in \{T, F\}$ and m clauses of width 2 like $x_1 \vee \overline{x_2}, \overline{x_3} \vee \overline{x_4}, x_2 \vee x_3, \ldots$
2. Find an assignment to satisfy all clauses.

Different from 3-SAT, 2-SAT is in P; however, MAX-2-SAT is NP-hard

1. Why 2-SAT $\in \mathrm{P}$?
2. $x_1 \vee \overline{x_2}$ implies $x_1 = F \Rightarrow x_2 = F$ and $x_2 = T \Rightarrow x_1 = T$
3. Construct a directed graph on $2n$ vertices and draw the above $2m$ relations

2-SAT

2-SAT

1. n Boolean variables $x_1, \ldots, x_n \in \{T, F\}$ and m clauses of width 2 like $x_1 \vee \overline{x_2}, \overline{x_3} \vee \overline{x_4}, x_2 \vee x_3, \ldots$
2. Find an assignment to satisfy all clauses.

Different from 3-SAT, 2-SAT is in P; however, MAX-2-SAT is NP-hard

1. Why 2-SAT $\in \mathrm{P}$?
2. $x_1 \vee \overline{x_2}$ implies $x_1 = F \Rightarrow x_2 = F$ and $x_2 = T \Rightarrow x_1 = T$
3. Construct a directed graph on $2n$ vertices and draw the above $2m$ relations
4. If $\exists$ a cycle contains $x_i = T$ and $x_i = F$, no solution
5. Otherwise, assign a variable to T and repeat it

Max-2-SAT

Max-2-SAT

1. n Boolean variables $x_1, \ldots, x_n \in \{T, F\}$ and m clauses of width 2 like $x_1 \vee \overline{x_2}, \overline{x_3} \vee \overline{x_4}, x_2 \vee x_3, \ldots$
2. Find an assignment to maximize # good clauses

Reduction from 3-SAT:

Max-2-SAT

Max-2-SAT

1. n Boolean variables $x_1, \ldots, x_n \in \{T, F\}$ and m clauses of width 2 like $x_1 \vee \overline{x_2}, \overline{x_3} \vee \overline{x_4}, x_2 \vee x_3, \ldots$
2. Find an assignment to maximize # good clauses

Reduction from 3-SAT:

1. For each clause $C = x \vee y \vee z$ in 3-SAT, consider 10 clauses in 2-SAT with an extra variable w_C
2. $x, y, z, w_C, \overline{x} \vee \overline{y}, \overline{x} \vee \overline{z}, \overline{y} \vee \overline{z}, x \vee \overline{w_C}, y \vee \overline{w_C}, z \vee \overline{w_C}$

Max-2-SAT

Max-2-SAT

1. n Boolean variables $x_1, \ldots, x_n \in \{T, F\}$ and m clauses of width 2 like $x_1 \vee \overline{x_2}$, $\overline{x_3} \vee \overline{x_4}$, $x_2 \vee x_3, \ldots$
2. Find an assignment to maximize # good clauses

Reduction from 3-SAT:

1. For each clause $C = x \vee y \vee z$ in 3-SAT, consider 10 clauses in 2-SAT with an extra variable w_C
2. $x, y, z, w_C, \overline{x} \vee \overline{y}, \overline{x} \vee \overline{z}, \overline{y} \vee \overline{z}, x \vee \overline{w_C}, y \vee \overline{w_C}, z \vee \overline{w_C}$
3. $C(\sigma) = \mathit{True} \Rightarrow$ 7 of them are satisfied with some w_C; otherwise at most 6
4. Claim: 3-SAT is satisfiable $\Leftrightarrow$ value(2-SAT) $= 7m$

Discussion

1. There are poly-time algorithms for special cases of Vertex-Cover and Subset-Sum
2. Example 1: Vertex-Cover of bipartite graphs is in P
3. Example 2: Subset-Sum of small integers is in P

Discussion

1. There are poly-time algorithms for special cases of Vertex-Cover and Subset-Sum
2. Example 1: Vertex-Cover of bipartite graphs is in P
3. Example 2: Subset-Sum of small integers is in P
4. There are various ways to design the reductions!

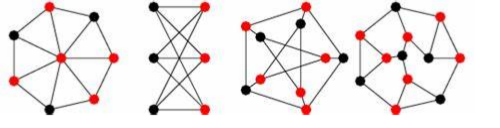
Discussion

1. There are poly-time algorithms for special cases of Vertex-Cover and Subset-Sum
2. Example 1: Vertex-Cover of bipartite graphs is in P
3. Example 2: Subset-Sum of small integers is in P
4. There are various ways to design the reductions!
5. Most important thing: The direction is from a known NP-complete problem to this problem!

Outline

Introduction

Many interesting problems, like finding the smallest Vertex-Cover in G, are NP-hard

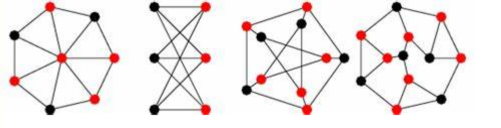


Question

In both theory and practice, what can we do for NP-hard problems?

Introduction

Many interesting problems, like finding the smallest Vertex-Cover in G, are NP-hard



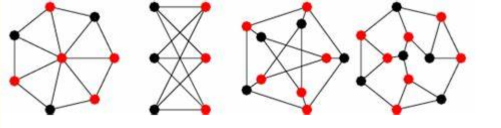
Question

In both theory and practice, what can we do for NP-hard problems?

1. Exhaustive Search (for SAT problems)
2. Local search heuristics (like evolution)

Introduction

Many interesting problems, like finding the smallest Vertex-Cover in G, are NP-hard



Question

In both theory and practice, what can we do for NP-hard problems?

1. Exhaustive Search (for SAT problems)
2. Local search heuristics (like evolution)
3. Approximation Algorithms

Approximation Algorithms

For an optimization problem like MAX-2-SAT, it may take exponential time to find the optimal solution whose value is OPT

Key Insight

Design a trade-off between time and accuracy

Approximation Algorithms

For an optimization problem like MAX-2-SAT, it may take exponential time to find the optimal solution whose value is OPT

Key Insight

Design a trade-off between time and accuracy

1. This is an important idea in CS: in big data algorithms, design trade-offs between space and accuracy

Approximation Algorithms

For an optimization problem like MAX-2-SAT, it may take exponential time to find the optimal solution whose value is OPT

Key Insight

Design a trade-off between time and accuracy

1. This is an important idea in CS: in big data algorithms, design trade-offs between space and accuracy
2. What do we mean accuracy?

Approximation Algorithms

For an optimization problem like MAX-2-SAT, it may take exponential time to find the optimal solution whose value is OPT

Key Insight

Design a trade-off between time and accuracy

1. This is an important idea in CS: in big data algorithms, design trade-offs between space and accuracy
2. What do we mean accuracy?
3. Suppose our algorithm finds a solution with value ANS
4. Define ANS/OPT as the approximation ratio — >1 for minimizations and <1 for maximizations

Vertex-Cover

Given G, find a vertex-cover with a minimum size

Theorem 35.1 in CLRS

There are poly-time algorithms that guarantee a 2-approximation

Vertex-Cover

Given G, find a vertex-cover with a minimum size

Theorem 35.1 in CLRS

There are poly-time algorithms that guarantee a 2-approximation

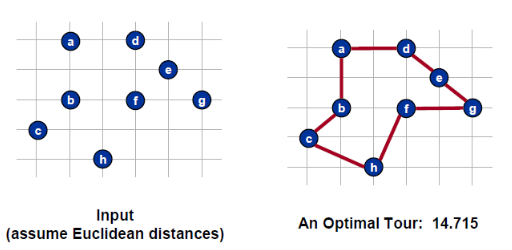
Recall that for Set-Cover, the approximation algorithm is $\ln n$.

Traveling Salesman Problem — Hamilton Cycle

Description: Given a weighted graph $G = (V, E)$, find a sequence $\vec{u} = (u_0, \ldots, u_n)$ of V s.t. (1) $u_0 = u_n$ and each vertex appears once; (2) the total weight $\sum_{i=1}^{k} c(u_{i-1}, u_i)$ is minimum.

Traveling Salesman Problem — Hamilton Cycle

Description: Given a weighted graph $G = (V, E)$, find a sequence $\vec{u} = (u_0, \ldots, u_n)$ of V s.t. (1) $u_0 = u_n$ and each vertex appears once; (2) the total weight $\sum_{i=1}^{k} c(u_{i-1}, u_i)$ is minimum.

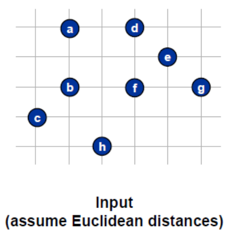


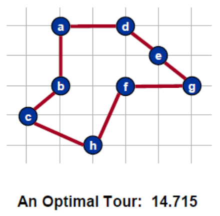
THM 35.2 in CLRS

If $c(\cdot, \cdot)$ satisfies the triangle inequality, there are poly-time algorithms with $\leqslant$ 2-approximation ratios

Traveling Salesman Problem — Hamilton Cycle

Description: Given a weighted graph $G = (V, E)$, find a sequence $\vec{u} = (u_0, \ldots, u_n)$ of V s.t. (1) $u_0 = u_n$ and each vertex appears once; (2) the total weight $\sum_{i=1}^{k} c(u_{i-1}, u_i)$ is minimum.



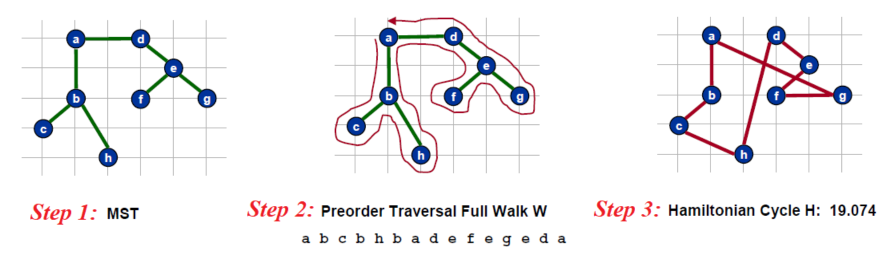


THM 35.2 in CLRS

If $c(\cdot, \cdot)$ satisfies the triangle inequality, there are poly-time algorithms with $\leqslant$ 2-approximation ratios

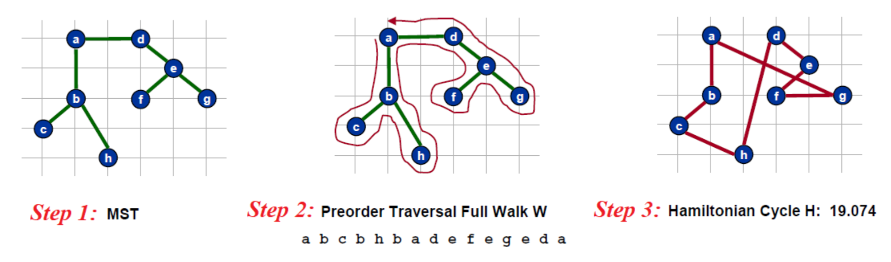
Remark: The triangle inequality is necessary, otherwise no constant approx unless $\mathrm{P}{=}\mathrm{NP}$— Section 35.2.2 in CLRS

2-Approx of TSP



1. Step 1: Find a MST T of G
2. Step 2: Find the preorder walk W of T
3. Step 3: Output the Hamilton cycle H of W

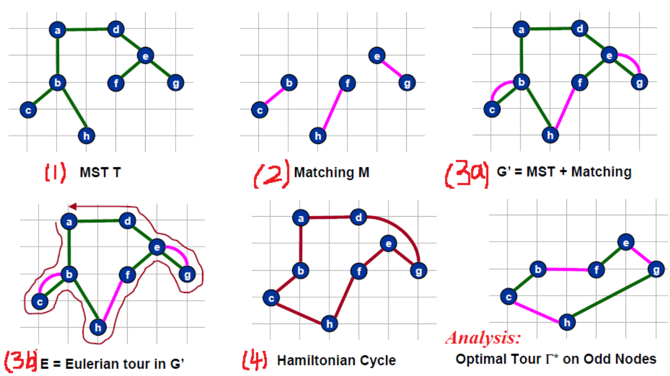
2-Approx of TSP



1. Step 1: Find a MST T of G
2. Step 2: Find the preorder walk W of T
3. Step 3: Output the Hamilton cycle H of W
4. Analysis: $c(H) \leqslant c(W) = 2c(T)$ and $c(T) \leqslant c(H^*)$ for the optimal solution

1.5-Approx of TSP

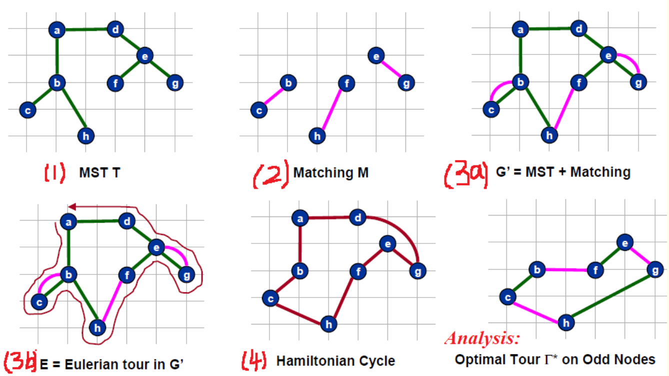
Christofides Algorithm — short-cut via the Eulerian Tour



1. Find a MST T

1.5-Approx of TSP

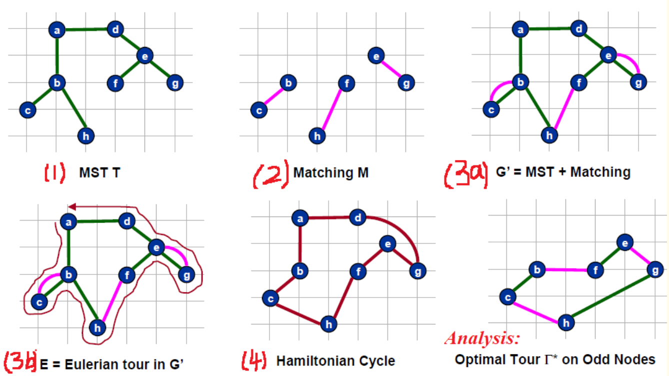
Christofides Algorithm — short-cut via the Eulerian Tour



1. Find a MST T
2. $M :=$ min-cost matching of odd degree nodes in T
3. Find the Eulerian tour E on $T \cup M$
4. Output the Hamilton cycle H of E

1.5-Approx of TSP

Christofides Algorithm — short-cut via the Eulerian Tour



1. Find a MST T
2. $M :=$ min-cost matching of odd degree nodes in T
3. Find the Eulerian tour E on $T \cup M$
4. Output the Hamilton cycle H of E
5. Analysis: $c(H) \leqslant c(M) + c(T)$ and $c(M) \leqslant c(H^*)/2$

Approximating MIS

Here is a strange claim about approximating max-independent-set

Claim

For some $\alpha < 1$, if $\exists$ an α-approximation algorithm in P for MIS, then $\sqrt{\alpha}$-approximating MIS is also in P.

1. Reduction: Given a graph $G = (V, E)$, consider its 2-fold OR-power $H = (V^2, E_H)$ s.t.

$$(u_1, u_2) \sim (v_1, v_2) \text{ iff } u_1 \sim v_1 \text{ or } u_2 \sim v_2$$

Approximating MIS

Here is a strange claim about approximating max-independent-set

Claim

For some $\alpha < 1$, if $\exists$ an α-approximation algorithm in P for MIS, then $\sqrt{\alpha}$-approximating MIS is also in P.

1. Reduction: Given a graph $G = (V, E)$, consider its 2-fold OR-power $H = (V^2, E_H)$ s.t.

$$(u_1, u_2) \sim (v_1, v_2) \text{ iff } u_1 \sim v_1 \text{ or } u_2 \sim v_2$$

2. Let S be an independent set of H and $S_1 := \{u_1 | \exists u_2 \ s.t. \ (u_1, u_2) \in S\}$ and vice versa for S_2
3. OBS: $S_1 \times S_2$ is also independent

Approximating MIS

Here is a strange claim about approximating max-independent-set

Claim

For some $\alpha < 1$, if $\exists$ an α-approximation algorithm in P for MIS, then $\sqrt{\alpha}$-approximating MIS is also in P.

1. Reduction: Given a graph $G = (V, E)$, consider its 2-fold OR-power $H = (V^2, E_H)$ s.t.

$$(u_1, u_2) \sim (v_1, v_2) \text{ iff } u_1 \sim v_1 \text{ or } u_2 \sim v_2$$

2. Let S be an independent set of H and $S_1 := \{u_1 | \exists u_2 \ s.t. \ (u_1, u_2) \in S\}$ and vice versa for S_2
3. OBS: $S_1 \times S_2$ is also independent
4. Next, how to finish the proof?

Approximating MIS

In fact, we could consider k-fold OR-power for any k!

Claim

For some $\alpha < 1$, if there is an α-approximation algorithm in P for MIS, then $\alpha^{1/k}$-approximating MIS is also in $\mathrm{P}$for any k.

1. If there is a 0.001-approximation, setting $k = 800$, we obtain a 0.99-approximation.
2. How shall we interpret this result?

Approximating MIS

In fact, we could consider k-fold OR-power for any k!

Claim

For some $\alpha < 1$, if there is an α-approximation algorithm in P for MIS, then $\alpha^{1/k}$-approximating MIS is also in $\mathrm{P}$for any k.

1. If there is a 0.001-approximation, setting $k = 800$, we obtain a 0.99-approximation.
2. How shall we interpret this result?
3. Researchers believe this is saying no constant-approximation for MIS unless NP=P
4. In fact, we can show: No $n^{0.999}$-approximation for MIS unless NP=P

Approximating MIS

In fact, we could consider k-fold OR-power for any k!

Claim

For some $\alpha < 1$, if there is an α-approximation algorithm in P for MIS, then $\alpha^{1/k}$-approximating MIS is also in $\mathrm{P}$for any k.

1. If there is a 0.001-approximation, setting $k = 800$, we obtain a 0.99-approximation.
2. How shall we interpret this result?
3. Researchers believe this is saying no constant-approximation for MIS unless NP=P
4. In fact, we can show: No $n^{0.999}$-approximation for MIS unless NP=P
5. 3 steps: NP-hard to distinguish $\exists$ an independent set of size $n/10$ or MIS $\leqslant 0.99 \cdot n/10$; then apply $O(\log n)$-fold OR-power; finally, sparsify the product to $n^{O(1)}$ size

Summary

1. Undecidable problem: Halting, Accepting, Rejecting, $\dots$
2. Many problems can not be solved in poly-time (at least we believe so) — NP captures the most interesting class

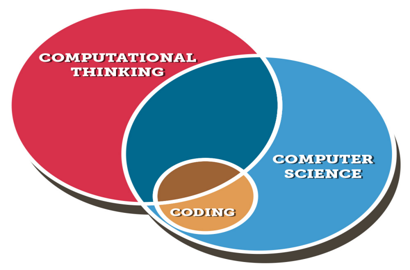
Summary

1. Undecidable problem: Halting, Accepting, Rejecting, ...
2. Many problems can not be solved in poly-time (at least we believe so) — NP captures the most interesting class
3. Complete problems stand for the hardest problem in a class like linear programming in P
4. A great notion is NP-complete: $\mathrm{P} \neq \mathrm{NP} \Leftrightarrow \forall$ NP-complete problem Q, $Q \notin \mathrm{P}$

Summary

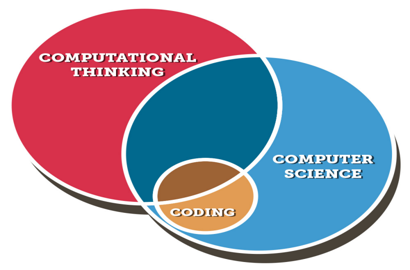
1. Undecidable problem: Halting, Accepting, Rejecting, . . .
2. Many problems can not be solved in poly-time (at least we believe so) — NP captures the most interesting class
3. Complete problems stand for the hardest problem in a class like linear programming in P
4. A great notion is NP-complete: $\mathrm{P} \neq \mathrm{NP} \Leftrightarrow \forall$ NP-complete problem Q, $Q \notin \mathrm{P}$
5. Many natural problems are NP-complete: SAT, CLIQUE, SUBSET-SUM, Vertex-Cover, . . .
6. There are many ways to cure NP-complete ☺

For this course



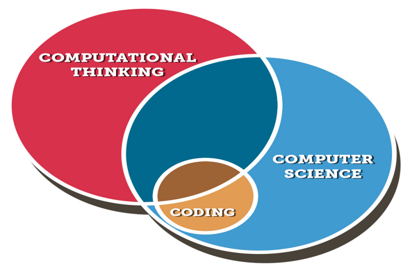
1. As a science, CS is much more than coding — creations, good definitions, rigorous analyses, ...

For this course



1. As a science, CS is much more than coding — creations, good definitions, rigorous analyses, . . .
2. Design and analysis of algorithms are the backbone of programs

For this course



1. As a science, CS is much more than coding — creations, good definitions, rigorous analyses, . . .
2. Design and analysis of algorithms are the backbone of programs

3. Hope you enjoy this course and learn something (at least) ☺

Questions?