

# Computational Models — Lecture 12<sup>1</sup>

## Handout Mode

Ronitt Rubinfeld and Iftach Haitner.

Tel Aviv University.

June 8/10, 2015

---

<sup>1</sup>Based on frames by Benny Chor, Tel Aviv University, modifying frames by Maurice Herlihy, Brown University. Also with modifications by Yishay Mansour.

## Talk Outline

- ▶ Reminder – deterministic and nondeterministic time classes
- ▶  $\mathcal{NP}$  and verifiability
- ▶ Additional NP languages
- ▶ The class  $co\text{-}\mathcal{NP}$
- ▶  $\mathcal{P}$  Verses  $\mathcal{NP}$
- ▶ NP-completeness
- ▶ Satisfiability
- ▶ Reductions
  
- Sipser's book, [7.4–7.5](#)

## Reminder – Deterministic Time

### Definition 1 (deterministic Time)

A **deterministic** TM  $M$  runs in time  $t$ , where  $t: \mathbb{N} \mapsto \mathbb{N}$ , if for **every** input  $w$ , the number of steps that  $M(w)$  uses is **at most**  $t(|w|)$ .

### Definition 2 (DTIME)

For  $t: \mathbb{N} \mapsto \mathbb{N}$ , let

$\text{DTIME}(t) = \{L \subseteq \Sigma^* : L \text{ is decided by an } O(t)\text{-time single tape TM}\}$

The bound on the running time is required to hold also for input **not** in the language (i.e.,  $L$ ).

### Definition 3 ( $\mathcal{P}$ )

$\mathcal{P} = \bigcup_{c \geq 0} \text{DTIME}(n^c)$

## Reminder – Non-Deterministic Time

### Definition 4 (nondeterministic Time)

A **non-deterministic** TM  $N$  runs in time  $t$ , where  $t: \mathbb{N} \mapsto \mathbb{N}$ , if for **every** input  $w$ , the **maximum** number of steps that  $N(w)$  uses on **any branch** of its computation tree, is **at most**  $t(|w|)$ .

Notice that also **non-accepting** branches must **reject** within the required time.

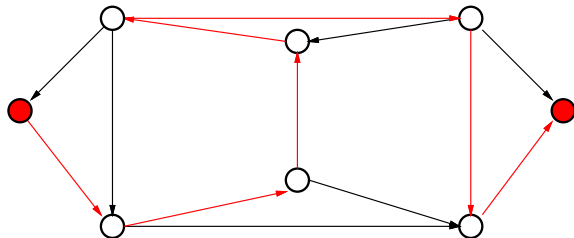
### Definition 5 (NTIME)

For  $t: \mathbb{N} \mapsto \mathbb{N}$  let  $\text{NTIME}(t) =$   
 $\{L \subseteq \Sigma^* : L \text{ is decided by an } O(t)\text{-time single tape NTM}\}$

### Definition 6 ( $\mathcal{NP}$ )

$$\mathcal{NP} = \bigcup_{c \geq 0} \text{NTIME}(n^c)$$

## Hamiltonian Path



A **Hamiltonian path** in a directed  $G$  visits each node **exactly** once.

$$\text{HAMPATH} = \{ \langle G, s, t \rangle : G \text{ has Hamiltonian path from } s \text{ to } t \}$$

### Question 7

How hard is it to decide **HAMPATH**?

Easy to obtain **exponential time** algorithm:

- ▶ Generate each potential path
- ▶ Check whether it is Hamiltonian

## HAMPATH $\in \mathcal{NP}$

Here is an NTM that decides HAMPATH in polynomial time.

### Algorithm 8 ( $N$ )

On input  $\langle G = (V, E), s, t \rangle$ ,

1. **Guess** a list of numbers  $p_1, \dots, p_m$ , where  $m = |V|$  and  $1 \leq p_i \leq m$ .
2. **Accept** if **all** the following hold (otherwise **Reject**):
  - ▶ No repetitions in list
  - ▶  $p_1 = s$  and  $p_m = t$ .
  - ▶  $(p_i, p_{i+1}) \in E$  for every  $1 \leq i \leq m - 1$

- ▶ How does a TM **guess** a string?

### Claim 9

$N$  runs in polynomial time

## Verifiability of HAMPATH

This problem has one very interesting feature: **polynomial verifiability**:

*We don't know a fast way to **find** a Hamiltonian path, but we can **check** whether a **given path** is Hamiltonian in polynomial time.*

**Verifying** correctness of a path is much **easier** than **determining** whether one exists

# Verifiability

## Definition 10 (verifier)

Algorithm  $V$  is a **verifier** for a language  $L$ , if

- ▶  $x \in L \implies \exists c \in \{0, 1\}^*$  s.t.  $V(x, c) = 1$ .
- ▶  $x \notin L \implies \nexists c \in \Sigma^*$  s.t.  $V(x, c) = 1$ .

- ▶ The verifier uses the additional information  $c$  to verify  $x \in L$ .
- ▶ If  $V$  accepts  $(x, c)$  (i.e., outputs 1), the string  $c$  is called a **certificate** (also known as, **proof** or **witness**) for  $x$ .
- ▶ A **polynomial verifier** runs in polynomial time in  $|x|$  (i.e., in the length of its **left-hand-side** input parameter).
- ▶ A language  $L$  is **polynomially verifiable**, if it has a polynomial verifier.
- Certificate for  $\langle G, s, t \rangle \in \text{HAMPATH}$  is Hamiltonian path from  $s$  to  $t$ .  
Easy to **verify** in time polynomial in  $|\langle G, s, t \rangle|$  whether given path is Hamiltonian.
- ▶ **Not** all languages are known to be polynomially verifiable.



# NP and Verifiability

## Theorem 11

A language is in  $\mathcal{NP}$  iff it has a *polynomial time verifier*.

Proof's idea:

- ▶ Verifier emulates NTM by using accepting branch as certificate.
- ▶ The NTM emulates the verifier by guessing the certificate.

Verifiability  $\implies \mathcal{NP}$

### Claim 12

If  $L$  has a poly-time verifier, then it is decided by some polynomial-time NTM.

Proof: Let  $V$  be poly-time verifier for  $L$  of running time  $p(n)$  for some  $p \in \text{poly}$ .

### Algorithm 13 ( $N$ )

On input  $x \in \{0, 1\}^n$ :

1. **Guess** a string  $c$  of length  $p(n)$ .
2. Emulate  $V$  on  $\langle x, c \rangle$
3. **Accept** if  $V$  accepts; Otherwise **Reject**.



- ▶ Why is it suffices to guess a string of length  $p(n)$ ?

## $\mathcal{NP} \implies$ Verifiability

### Claim 14

If  $L$  is decided by a polynomial-time NTM  $N$ , then  $L$  has a poly-time verifier.

Proof: Assume for simplicity that at each step of  $N$ , the number of possible non-deterministic moves is at most two.

### Algorithm 15 (V)

On input  $(x, c)$ :

1. Emulate  $N(x)$ , treating each symbol of  $c$  as a description of the non-deterministic choice in each step of  $N$ .
2. **Accept** if this branch accepts; Otherwise **Reject**.

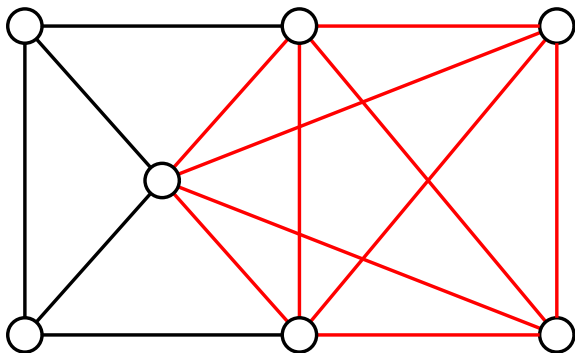


Without the simplifying assumption?

# Section 1

## **A few more NP languages**

# CLIQUE



A **clique** in a graph is a subgraph where every two nodes are connected by an edge.

A  **$k$ -clique** is a clique of size  $k$ .

## Question 16

What is the **largest  $k$ -clique** in the figure?

## CLIQUE cont.

$\text{CLIQUE} = \{ \langle G, k \rangle : G \text{ is an undirected graph with a } k\text{-clique} \}$

### Theorem 17

$\text{CLIQUE} \in \mathcal{NP}$

Proof's idea: The clique is the certificate.

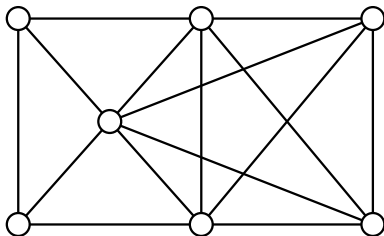
### Algorithm 18 (V)

On input  $(\langle G, k \rangle, c)$

Accept if  $c$  is a  $k$ -clique subgraph of  $G$ ;

Otherwise Reject.

## Independent Set



An **independent set** in a graph is a set of vertexes, no two of which are linked by an edge.

A  **$k$ -IS** is an independent set of size  $k$ .

### Question 19

What is the **largest  $k$ -IS** in the figure?

## IND-SET cont.

$\text{IND-SET} = \{\langle G, k \rangle : G \text{ contains an independent set of size } k\}$

### Theorem 20

$\text{IND-SET} \in \mathcal{NP}$

Proof's idea: The independent set is the certificate.

### Algorithm 21 (V)

On input  $(\langle G, k \rangle, c)$

Accept if  $c$  is a  $k$ -IS of  $G$  (no edges between nodes in  $c$ , and  $|c| = k$ );

Otherwise Reject.



## Section 2

# The Class $\text{coNP}$

## The Class $co-\mathcal{NP}$

$\overline{\text{CLIQUE}} = \{\langle G, k \rangle : G \text{ is an undirected graph with no } k\text{-clique}\}$  seems **not** to be member of  $\mathcal{NP}$ .

It is harder to efficiently verify that something does **not** exist than to efficiently verify that something **does** exist.

### Definition 22 ( $co-\mathcal{NP}$ )

$$co-\mathcal{NP} = \{L : \bar{L} \in \mathcal{NP}\}.$$

So far, **no one** knows if  $co-\mathcal{NP}$  is distinct from  $\mathcal{NP}$ .

### Claim 23

$$\mathcal{P} \subseteq co-\mathcal{NP}.$$

Proof?

$$L \in \mathcal{P} \implies \bar{L} \in \mathcal{P} \implies \bar{L} \in \mathcal{NP} \implies L \in co-\mathcal{NP}$$

## Be very careful!

Is Primality in  $\mathcal{NP}$ ?  $\text{co-}\mathcal{NP}$ ?

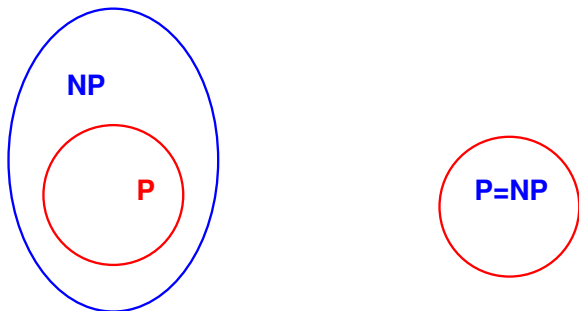
How would you prove that a number is prime without trying all divisors?

Actually it is in P! (not obvious at all)

## Section 3

### **P vs. NP**

# $\mathcal{P}$ Vs. $\mathcal{NP}$



The question  $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$  is one of the great unsolved mysteries in contemporary mathematics.

- ▶ Most computer scientists believe the two classes are **not** equal
- ▶ Most bogus proofs show them equal (?)

## $\mathcal{P}$ Vs. $\mathcal{NP}$ , cont.

If  $\mathcal{P}$  differs from  $\mathcal{NP}$ , then the distinction between  $\mathcal{P}$  and  $\mathcal{NP} \setminus \mathcal{P}$  is meaningful and important.

- ▶ languages in  $\mathcal{P}$  are **tractable**
- ▶ languages in  $\mathcal{NP} \setminus \mathcal{P}$  are **intractable**

Until we can prove that  $\mathcal{P} \neq \mathcal{NP}$ , there is no hope of proving that a **specific** language lies in  $\mathcal{NP} \setminus \mathcal{P}$ .

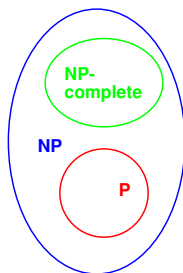
Nevertheless, we **can** prove statements of the form

*If  $A \in \mathcal{NP} \setminus \mathcal{P}$ , then  $B \in \mathcal{NP} \setminus \mathcal{P}$ .*

## Section 4

# NP Completeness

# NP Completeness



The class of **NP-complete** languages are

- ▶ “hardest” languages in  $\mathcal{NP}$
- ▶ “least likely” to be in  $\mathcal{P}$
- ▶ If any NP-complete  $L \in \mathcal{P}$ , then  $\mathcal{NP} = \mathcal{P}$ .

Such languages, “carry on their backs” the burden of all of  $\mathcal{NP}$ .

## Question 24

Are there NP-complete languages?



# Polynomial-Time Computable Functions

## Definition 25 (poly-time computable functions)

A function  $f : \Sigma^* \mapsto \Sigma^*$  is **polynomial-time computable**, if there is a poly-time **deterministic** TM that

- ▶ starts with input  $w$ , and
- ▶ halts with  $f(w)$  on tape.

# Polynomial-Time Reducibility

## Definition 26

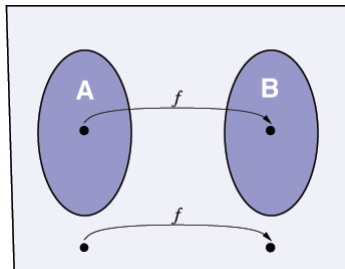
A language  $A$  is **polynomial time mapping reducible** to  $B$ , denoted  $A \leq_P B$ , if exists poly-time computable  $f$  such that

$$w \in A \iff f(w) \in B.$$

for every  $w \in \Sigma^*$ .

The function  $f$  is called a **polynomial-time reduction** from  $A$  to  $B$ .

The mapping  $f$  **efficiently** converts questions about membership in  $A$  to membership in  $B$ .



## Reductions to $\mathcal{P}$

### Theorem 27

If  $A \leq_P B$  and  $B \in \mathcal{P}$  then  $A \in \mathcal{P}$ .

Proof:

- ▶ Let  $f$  the reduction from  $A$  to  $B$ , computed by TM  $M_f$ .  
On input  $x$ , the TM  $M_f$  makes at most  $c_f \cdot |x|^{a_f}$  steps.
- ▶ Let  $M_B$  be the poly-time decider for  $B$ .  
On input  $y$ , the TM  $M_B$  makes at most  $c_B \cdot |y|^{a_B}$  steps.

### Algorithm 28 (Decider $M_A$ for $A$ )

On input  $x$ , return  $M_B(f(x))$

- ▶  $M_A$  decides  $A$
- ▶ Since  $|f(x)| \leq c_f |x|^{a_f}$ , running time of  $M_B(x)$ , is at most  
 $c_B \cdot (c_f \cdot |x|^{a_f})^{a_B} = (c_B \cdot c_f^{a_B}) \cdot |x|^{a_f \cdot a_B} \in \text{poly}(|x|)$

Hence,  $A \in \mathcal{P}$

## What $A \leq_P B$ tells us about B?

### Question 29

Assume that  $\{0^n 1^n : n \geq 0\} \leq_P L$ . Does it yield that  $L \in \mathcal{P}$ ?

**Answer:** No. (Reduction in the wrong direction!)

Let  $L = H_{TM}$  and define  $f(x) = \begin{cases} M_{stop}, & x \in \{0^n 1^n : n \in \mathbb{N}\} \\ M_{no-stop}, & \text{otherwise.} \end{cases}$

$A \leq_P B$  does tell us that B is “at least as hard” as A.

## NP Completeness, Formal Definition

### Definition 30 ( $\mathcal{NP}$ -complete)

A language  $B$  is **NP-complete**, if

- ▶  $B \in \mathcal{NP}$ , and
- ▶ Every  $A \in \mathcal{NP}$  is **poly-time** reducible to  $B$  (i.e.,  $A \leq_P B$ )

We let  $\mathcal{NPC}$  denote the class of all NP-complete languages.

Compare to

### Definition 31 (RE-Complete)

A language  $B$  is **RE-complete**, if

- ▶  $B \in \mathcal{RE}$ , and
- ▶ Every  $A \in \mathcal{RE}$  is **mapping** reducible to  $B$ .

## Why NP Completeness?

### Theorem 32

If  $B \in \mathcal{NPC}$  and  $B \in \mathcal{P}$ , then  $\mathcal{P} = \mathcal{NP}$ .

Proof: Immediately follows by **Thm 27**. ♣

To show  $\mathcal{P} = \mathcal{NP}$  (and make an instant fortune, see [www.claymath.org/millennium/P\\_vs\\_NP/](http://www.claymath.org/millennium/P_vs_NP/)), suffices to find a polynomial-time algorithm for **any** NP-complete problem.

### Question 33

Is  $\mathcal{NPC}$  empty?

## $\mathcal{NPC}$ Is Not Empty

$A_{NP} = \{ \langle M, x, 1^n \rangle : M \text{ is a TM} \wedge \exists c \in \Sigma^* \text{ s.t. } M(x, c) \text{ accepts within } n \text{ steps} \}$ .

### Theorem 34

$A_{NP} \in \mathcal{NPC}$

Proof:

- ▶ Clearly  $A_{NP} \in \mathcal{NP}$ . (?)
- ▶ Let  $L \in \mathcal{NP}$ , let  $V$  be a verifier for  $L$  and let  $p \in \text{poly}$  be a bound on the running time of  $V$  (i.e.,  $V(x, \cdot)$  halts within  $p(|x|)$  steps, for every  $x \in \Sigma^*$ ).
- ▶ Define  $f(x) = \langle V, x, 1^{p(|x|)} \rangle$ .
- ▶ Clearly  $f$  is poly-time computable and  $x \in L \iff f(x) \in A_{NP}$ .



## Finding Additional NP-complete Languages

### Theorem 35

Assume that

1.  $B \in \mathcal{NP}$
2.  $A \in \mathcal{NPC}$  and  $A \leq_P B$

then  $B \in \mathcal{NPC}$ .

Proof: Home exercise ... ♣

We would like to find  $L \in \mathcal{NPC}$  that is “natural” and “easy” to reduce to.

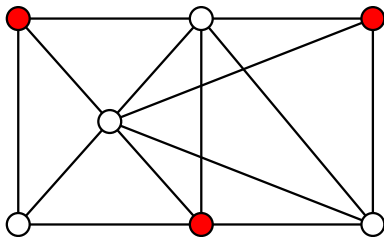
The most useful such language is the **language of satisfied formulas**. But, we will start with **clique**.



## Section 5

# **A first example of an NP-completeness reduction: Independent Set**

## Independent Set



### Definition 36

An **independent set** in a graph is a set of vertexes, no two of which are linked by an edge.

The language of independent sets:

$$\text{IND-SET} = \{ \langle G, k \rangle : G \text{ contains an independent set of size } k \}$$

Clearly  $\text{IND-SET} \in \mathcal{NP}$ . We show that  $\text{CLIQUE} \leq_P \text{IND-SET}$  and deduce that  $\text{IND-SET} \in \mathcal{NPC}$

## CLIQUE $\leq_P$ IND-SET

Proof:

### Definition 37

The **complement** of a graph  $G = (V, E)$  is a graph  $G^c = (V, E^c)$ , where  $E^c = \{(v_1, v_2) : v_1, v_2 \in V \text{ and } (v_1, v_2) \notin E\}$ .

The reduction  $f(G, k)$  from **CLIQUE** to **IND-SET** simply computes the complement of the graph and outputs  $(G^c, k)$ .

$f$  satisfies:

- ▶  $U$  is a **clique** in  $G$   $\iff$   $U$  is a **independent set** in  $G^c$ .
- ▶ computable in polynomial time!



### Remark 38

Same reduction shows that **IND-SET**  $\leq_P$  **CLIQUE**

## Section 6

# Satisfiability

# Boolean Variables

- ▶ A **Boolean** variable assumes values
  - ▶ **TRUE** (written **1**), and **FALSE** (written **0**).
- ▶ Boolean operations:
  - ▶ and:  $\wedge$
  - ▶ or:  $\vee$
  - ▶ not:  $\neg$
- ▶ Examples:

$$0 \wedge 1 = 0$$

$$0 \vee 1 = 1$$

$$\overline{0} = 1$$

## Boolean Formulas and SAT

A **Boolean** formula is an expression involving Boolean variables and operations.

$$\phi = (\bar{x} \wedge y) \vee (x \wedge \bar{z})$$

### Definition 39 (satisfiable formula)

A formula is **satisfiable**, if some **Boolean** assignment to its variables, makes the formula evaluate to **1**.

The formula  $\phi = (\bar{x} \wedge y) \vee (x \wedge \bar{z})$  is satisfiable by the assignment

$$x = 0$$

$$y = 1$$

$$z = 0$$

The language of satisfied formulas:

$$\text{SAT} = \{ \langle \phi \rangle : \phi \text{ is a satisfiable Boolean formula} \}$$

$SAT \in \mathcal{NP}$

$SAT = \{\langle \phi \rangle : \phi \text{ is satisfiable Boolean formula}\}$

**Theorem 40 (Cook-Levin (early 70s))**

$SAT \in \mathcal{NP}$ .

- ▶ The “most important”  $\mathcal{NP}$ -complete language.
- ▶ It is easy to see that  $SAT \in \mathcal{NP}$
- ▶ For the proof of other part wait for next week ...

## The Language 3SAT

It is useful to consider a special version of SAT

- ▶ A **literal** is a variable or negated variable:  $x$  or  $\bar{x}$ .
- ▶ A **clause** is several literals joined by  $\vee$ s:  $(x_1 \vee \bar{x}_2 \vee \bar{x}_3)$
- ▶ A Boolean formula is in **conjunctive normal form** (CNF) if it consists of **clauses**, connected with  $\wedge$ s.

For example:  $(x_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee x_4) \wedge (x_3 \vee \bar{x}_5 \vee x_6) \wedge (x_3 \vee \bar{x}_6)$

### Definition 41

A Boolean formula is in **k-CNF form**, if it is a **CNF** formula, and all clauses have **k** literals.

- ▶ Example of 3CNF:  $(x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (x_3 \vee \bar{x}_5 \vee x_6) \wedge (x_3 \vee \bar{x}_6 \vee x_4)$

The language of satisfied **3CNF** formulas:

$$3SAT = \{ \langle \phi \rangle : \phi \text{ is satisfiable 3CNF formula} \}$$



## $3SAT \in \mathcal{NPC}$

- ▶ Clearly  $3SAT \leq_P SAT$ .  
and  $3SAT \in \mathcal{NP}$ .
- ▶ We show next class that  $SAT \leq_P 3SAT$ , yielding that  $3SAT \in \mathcal{NPC}$ .
- ▶ Since  $3SAT$  has more **structure** than  $SAT$ , it is simpler to reduce it to other languages.
- ▶ In the following we prove that several  $\mathcal{NP}$  languages are  $\mathcal{NPC}$ , by showing a reduction from  $3SAT$  to them.

1SAT, 2SAT  $\in \mathcal{P}$

### Question 42

Is 1SAT  $\in \mathcal{P}$  ?

### Question 43

Is 2SAT  $\in \mathcal{P}$  ?

## A Graph characterization for 2CNF

### Definition 44

For 2CNF formula  $\phi$  with variables  $x_1, \dots, x_k$ , define a directed graph  $G(\phi) = (V, E)$  as

- ▶  $V = \{x_1, \bar{x}_1, \dots, x_k, \bar{x}_k\}$
- ▶  $E = \{(\bar{\ell}, h), (\bar{h}, \ell) : (\ell \vee h) \in \phi\}$  (taking  $\bar{\bar{x}} = x$ ).

### Claim 45

$G(\phi)$  contains path from  $\ell$  to  $h \implies$  in any satisfying assignment of  $\phi$  with  $\ell = 1$ , it holds that  $h = 1$ .

Proof?

### Claim 46

$G(\phi)$  contains path from  $\ell$  to  $h \implies G(\phi)$  contains path from  $\bar{h}$  to  $\bar{\ell}$

Proof?

## A Graph characterization for 2CNF, cont

### Definition 47

A variable  $x$  in 2CNF  $\phi$  is **bad**, if  $\exists$  paths in  $G(\phi)$  from  $x$  to  $\bar{x}$  and from  $\bar{x}$  to  $x$ .

### Claim 48

A 2CNF formula is satisfiable **iff** it contains no bad variables.

- ▶  $\phi$  has bad variable  $\implies \phi$  is unsatisfiable. Proof? by **Claim 45**
- ▶  $\phi$  has **no** bad variables  $\implies \phi$  is satisfiable. Proof?

## A Graph characterization for 2CNF, cont

### Definition 49

A variable  $x$  in 2CNF  $\phi$  is **bad**, if  $\exists$  paths in  $G(\phi)$  from  $x$  to  $\bar{x}$  and from  $\bar{x}$  to  $x$ .

### Claim 50

A 2CNF formula is satisfiable iff it contains no bad variables.

### Algorithm 51

While  $\phi$  has unassigned variables:

1. Pick **unassigned** literal  $l \in V$  that has **no path** from  $l$  to  $\bar{l}$
2. Assign **1** to all literals reachable from  $l$ , and **0** to their negations.
3. Recompute graph for "reduced formula"

Can there be conflicts (i.e.,  $x$  and  $\bar{x}$  assigned the same value)?

- ▶ Same iteration? Assume  $l \rightsquigarrow h \rightsquigarrow \bar{h}$ .  $\implies h \rightsquigarrow \bar{h} \rightsquigarrow \bar{l} \implies l \rightsquigarrow \bar{l}$ .
- ▶ Different iterations? Removing edges cannot add bad variables.

Final assignment satisfies  $\phi$ . Assume  $h$  is set to 1, then **all** clauses it participates in are satisfied.

## Poly-time algorithm for 2SAT

### Algorithm 52 (TwoSatSolver)

Input:  $2CNF \phi$

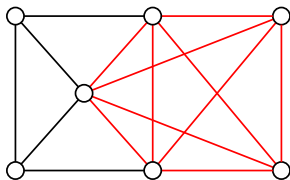
Return **TRUE** if there exist **no** bad variables in  $\phi$ :

- ▶ Efficiency?
- ▶ Correctness?

# Section 7

## **Clique**

## 3SAT $\leq_P$ CLIQUE



CLIQUE =  $\{ \langle G, k \rangle : G \text{ is an undirected graph with a } k\text{-clique} \}$

### Claim 53

3SAT  $\leq_P$  CLIQUE.

Since CLIQUE  $\in \mathcal{NP}$ , it follows that CLIQUE  $\in \mathcal{NPC}$ .

Proof's idea: We'll construct a poly-time reduction  $f$  that maps 3CNF formulae  $\phi$  to pairs  $\langle G, k \rangle$  of graphs and numbers.

The function  $f$  will have the property that  $\phi$  is satisfiable, iff  $G$  has a clique of size  $k$ .



## Proving $3SAT \leq_P CLIQUE$

On input  $\phi$ , a 3CNF formula, the mapping reduction is defined as follows:

let  $k$  be the number of clauses in  $\phi$ .

We construct a graph  $G = G(\phi)$ , see below, and output  $(G, k)$ .

The graph  $G$  is defined as follows:

- ▶ Nodes in  $G$  are organized into triples  $t_1, \dots, t_k$ .
- ▶ Each triple corresponds to a clause of  $\phi$
- ▶ Each node in a triple corresponds to a literal in corresponding clause.

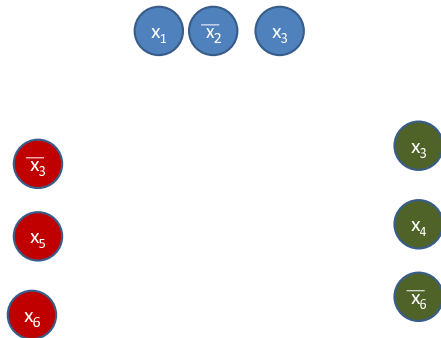
Ongoing example:

$$(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_3 \vee x_5 \vee x_6) \wedge (x_3 \vee \bar{x}_6 \vee x_4)$$

## Nodes of G

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_3} \vee x_5 \vee x_6) \wedge (x_3 \vee x_4 \vee \overline{x_6})$$

Add a node per **literal**

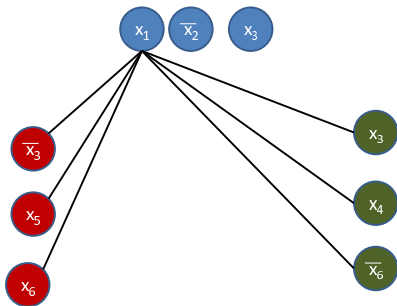


## Edges of G

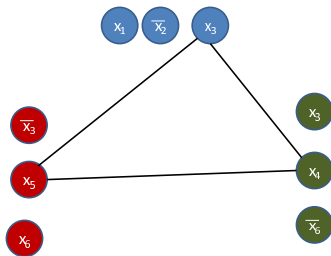
Add edges between **all** vertex pairs, **except**

- ▶ within **same** triple
- ▶ between **contradictory** literals (e.g.,  $x_3$  and  $\overline{x_3}$ )

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_3} \vee x_5 \vee x_6) \wedge (x_3 \vee x_4 \vee \overline{x_6})$$



$\phi \in 3\text{SAT} \implies \langle G, k \rangle \in \text{CLIQUE}$

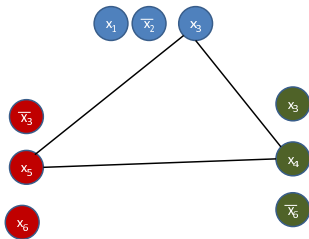


Proof: Suppose  $\phi$  is satisfiable by an assignment  $\psi$ .  
With respect to  $\psi$ :

- ▶ At least one literal is assigned to 1 in every clause (?)
- ▶ Select a 1-literal in every tuple;  
These literals can be joined by edges (?)  
Yielding a  $k$ -clique in  $G$ . ♣




$\langle G, k \rangle \in \text{CLIQUE} \implies \phi \in \text{3SAT}$



Proof: Suppose  $G$  has a  $k$ -clique.

- ▶ No two of the clique nodes are in the same triple. (?)
- ▶  $G$  has  $k$  vertices and  $k$  clauses, so each triple has exactly one clique node.
- ▶ Assign 1 to each node in clique;  
Assignment has no contradictions (?)  
Yielding a satisfying assignment to  $\phi$ .

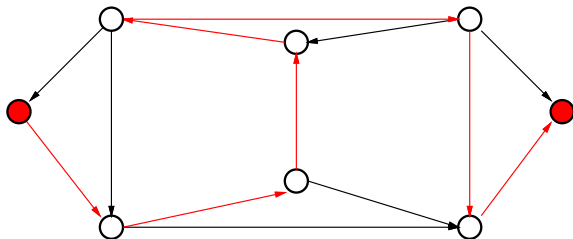
## Recap

- ▶ We've constructed a poly-time computable function  $f$ .
- ▶ We saw that  $f$  has the property that  $\phi \in 3SAT$  iff  $f(\phi) \in CLIQUE$ .
- ▶ Therefore,  $f$  is a poly-time reduction from  $3SAT$  to  $CLIQUE$   
 $\implies 3SAT \leq_P CLIQUE.$  

## Section 8

# Hamiltonian Paths and Cycles

## Reminder – Hamiltonian Path



A **Hamiltonian path** in a directed  $G$  visits each node **exactly** once.

$$\text{HAMPATH} = \{ \langle G, s, t \rangle : G \text{ has Hamiltonian path from } s \text{ to } t \}$$

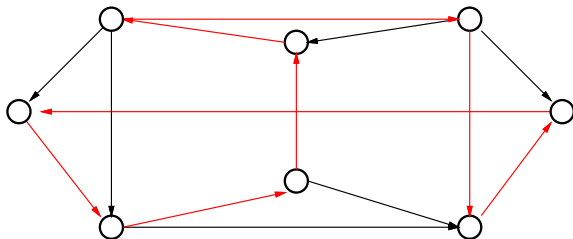
### Theorem 54

$\text{HAMPATH} \in \mathcal{NPC}$ .

Not today ...



## Hamiltonian Cycle



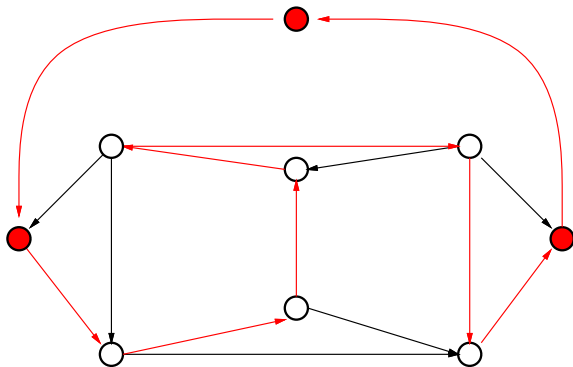
A **Hamiltonian Cycle** in a graph is an **Hamiltonian path** that ends up where it starts (i.e.,  $s = t$ ).

$$\text{HAMCYCLE} = \{ \langle G \rangle : G \text{ has Hamiltonian cycle} \}$$

Clearly  $\text{HAMCYCLE} \in \mathcal{NP}$ . We will show that  $\text{HAMPATH} \leq_P \text{HAMCYCLE}$ , and deduce that  $\text{HAMCYCLE} \in \mathcal{NPC}$

# HAMPATH $\leq_p$ HAMCYCLE.

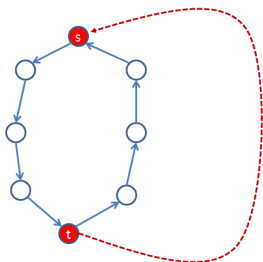
Proof's idea:



Hey, is the new vertex really needed? Why not just add an edge from  $t$  to  $s$ ?

## HAMPATH $\leq_P$ HAMCYCLE.

Why the new vertex really needed? Why not just add an edge from  $t$  to  $s$ ?



HAMCYCLE  $\leq_P$  HAMPATH.

**Claim 55**

HAMCYCLE  $\leq_P$  HAMPATH.

Left as an easy (recommended) exercise.

## Undirected Hamiltonian Circuit

$\text{UHAMCYCLE} = \{\langle G \rangle : G \text{ is undirected \& has Hamiltonian cycle}\}$

where Hamiltonian cycle/path in an **undirect** graph, is defined analogously to the direct case.

Clearly  $\text{UHAMCYCLE} \in \mathcal{NP}$ .

It is not hard to show (see Sipser 7.55, for a similar proof) that  $\text{HAMCYCLE} \leq_P \text{UHAMCYCLE}$ , and deduce that  $\text{UHAMCYCLE} \in \mathcal{NPC}$

## Chains of Reductions: NPC Problems

