

Computational Models - Lecture 11¹

Handout Mode

Iftach Haitner and Yishay Mansour.

Tel Aviv University.

May 7/9, 2012

¹Based on frames by Benny Chor, Tel Aviv University, modifying frames by Maurice Herlihy, Brown University.

Talk Outline

- Reminder – deterministic and nondeterministic time classes
- Additional NP languages
- The class $co-NP$
- P Verses 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 – NonDeterministic 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})

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

Reminder – NonDeterministic Time, cont.

Definition 7 (verifier)

A **verifier** for a language L , is an algorithm V with $L = \{w : V \text{ accepts } \langle w, c \rangle \text{ for some string } c\}$.

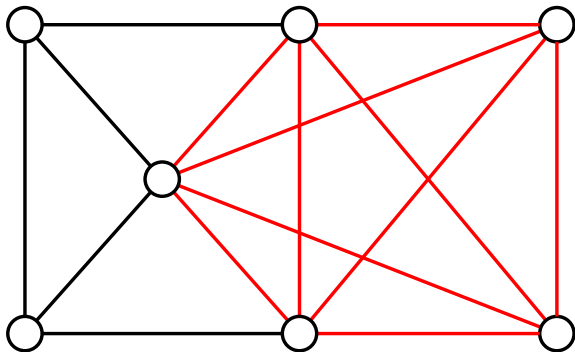
Theorem 8

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

Section 1

Additional 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 9

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 10

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

Proof's idea: The clique is the certificate.

Algorithm 11 (V)

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

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

Otherwise **Reject**.

SUBSET-SUM

- A collection of non-negative integers x_1, \dots, x_k
- A target number t

Question: does some subcollection add up to t ?

Example 12

- $\{4, 11, 16, 21, 27\}, 25) \in \text{SUBSET-SUM}$ because $4 + 21 = 25$.
- $\{4, 11, 16, 21, 27\}, 26) \notin \text{SUBSET-SUM}$ (why?)

SUBSET-SUM cont.

$$\text{SUBSET-SUM} = \{ \langle S = \{x_1, \dots, x_k\}, t \rangle : \exists \{y_1, \dots, y_\ell\} \subseteq S : \sum_{j=1}^{\ell} y_j = t \}$$

Collections are sets: repetitions not allowed.

Theorem 13

$\text{SUBSET-SUM} \in \mathcal{NP}$

Proof's idea: The **subset** is the **certificate**.

Algorithm 14 (V)

On input $(\langle S = \{x_1, \dots, x_k\}, t \rangle, c)$:

Accept if the following holds (otherwise **Reject**):

- 1 c is a collection of numbers summing to t .
- 2 c is a subset of S

Theorem 15

There is a pseudo-polynomial algorithm for SUBSET-SUM

Graph characterization of SUBSET-SUM

Definition 16

For $\langle S = \{x_1, \dots, x_k\}, t \rangle$, let $G(S, t) = (V, E)$

- $V = V_0 \cup V_1 \dots \cup V_k$, where $V_i = \{v_{i,0}, \dots, v_{i,t}\}$
- $E = E_1 \cup \dots \cup E_k$, where
 $E_i = \{(v_{i-1,j}, v_{i,j}), (v_{i-1,j}, v_{i,j+x_i}) : j \in \{0, \dots, t\}\}$

Claim 17

$v_{i,j}$ is reachable from $v_{0,0}$ in $G(S, t)$, iff
 $\langle \{x_1, \dots, x_i\}, j \rangle \in \text{SUBSET-SUM}$.

Proof?

Pseudo-polynomial algorithm for SUBSET-SUM

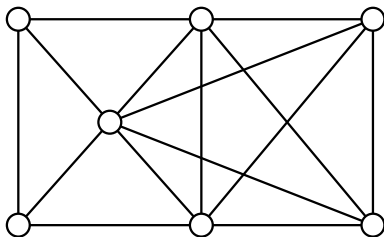
Algorithm 18

Input $\langle S, t \rangle$

Return **TRUE** iff $v_{k,t}$ is reachable from $v_{0,0}$ in $G(S, t)$.

- Correctness
- Size of $G(S, t)$ is $O(|S| \cdot t)$, and therefore the resulting algorithm for **SUBSET-SUM** runs in time $\text{poly}(|S| \cdot t)$.
Not $\text{poly}(\log_2 t)$ but $\text{poly}(t)$ (i.e., pseudo-polynomial)

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**.

KNAPSACK

- A collection of non-negative integers x_1, \dots, x_k (size)
- A collection of non-negative integers y_1, \dots, y_k (benefit)
- A capacity B
- A target number t

Question: does some $S \subset [1, k]$ has $\sum_{i \in S} x_i \leq B$ and $\sum_{i \in S} y_i \geq t$?

Example 22

- $(\{4, 11, 16, 21, 27\}, \{7, 3, 9, 5, 35\}, B = 20, t = 16) \in \text{KNAPSACK}$
because $S = \{1, 3\}$.

Theorem 23

$KNAPSACK \in \mathcal{NP}$

Proof's idea: The **subset** is the **certificate**.

Algorithm 24 (V)

On input $(\langle \{x_1, \dots, x_k\}, \{y_1, \dots, y_k\}, B, t \rangle, c)$:

Accept if the following holds (otherwise **Reject**):

- 1 $\sum_{i \in c} x_i \leq B$
- 2 $\sum_{i \in c} y_i \geq t$.
- 3 c is a subset of $[1, \dots, k]$

Pseudo-polynomial algorithm for KNAPSACK

Theorem 25

There is a pseudo-polynomial algorithm for KNAPSACK

- Running time $O(k \cdot t)$
- Not $\text{poly}(\log_2 t)$ but $\text{poly}(t)$ (i.e., pseudo-polynomial)

Algorithm 26 (KNAPSACK)

$A(0, 0) = 0$ and $A(0, p) = \infty$ for $p \neq 0$.

For $i = 1$ to k DO

For $p = 0$ to t DO

$A(i, p) = \min\{A(i - 1, p), x_i + A(i - 1, p - y_i)\}$

END

Test $t \leq \max\{p \mid A(k, p) \leq B\}$.

Section 2

The Class coNP

The Class co-NP

CLIQUE and SUBSET-SUM seem **not** to be members of NP .

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

Definition 27 (co-NP)

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

So far, **no one** knows if co-NP is distinct from NP .

Claim 28

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

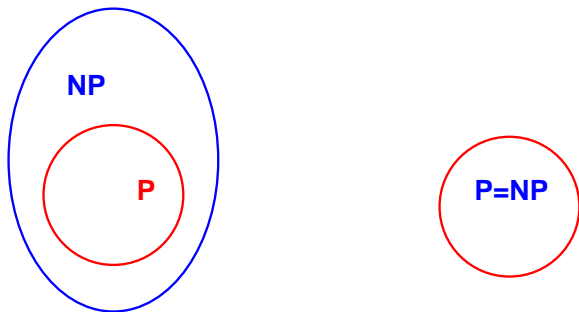
Proof?

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

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 (**why?**)

Observations

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

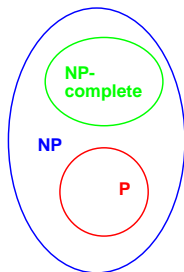
If $A \in \mathcal{NP} \setminus \mathcal{P}$ then $B \in \mathcal{NP} \setminus \mathcal{P}$.

If $\text{co-}\mathcal{NP} \neq \mathcal{NP}$ then $\mathcal{P} \neq \mathcal{NP}$.

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 29

Are there NP-complete languages?

Polynomial-Time Computable Functions

Definition 30 (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 31

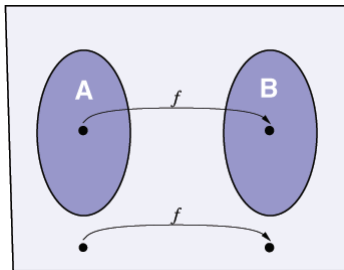
A language A is **polynomial time mapping reducible** to B , denoted $A \leq_P B$, if exists poly-time computable f such that

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

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 32

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 33 (Decider M_A for A)

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

- M_A decides A
- 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, $A \in \mathcal{P}$

Question 34

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

$$L = H_{TM}$$

If $x = 0^n 1^n$ then $f(x) = M_{stop}$

Otherwise $f(x) = M_{run}$.

NP Completeness, Formal Definition

Definition 35 (\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 36 (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 37

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

Proof: Immediately follows by **Thm 32**. ♣

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

Question 38

Is \mathcal{NPC} empty?

\mathcal{NP} Is Not Empty

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

Theorem 39

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

Proof:

- Clearly $A_{\text{NP}} \in \mathcal{NP}$
- 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., $V(w, \cdot)$ halts within $p(|w|)$ steps, for every $w \in \Sigma^*$).
- Define $f(x) = \langle V, x, 1^{p(|x|)} \rangle$.
- Clearly f is poly-time computable and $x \in L \iff f(x) \in A_{\text{NP}}$.



Finding Additional NP-complete Languages

Theorem 40

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.

Section 5

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 41 (satisfiable formula)

A formula is **satisfiable**, if some assignment of 0s and 1s to the 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 42 (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 43

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:

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

$3SAT \in \mathcal{NPC}$

- It is trivial that $3SAT \leq_P SAT$
- Clearly $3SAT \in \mathcal{NP}$
- We later show 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 44

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

Question 45

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

A Graph characterization for 2CNF

Definition 46

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

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

Claim 47

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

Proof?

Claim 48

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

Proof?

A Graph characterization for 2CNF, cont

Definition 49

A variable x in a 2CNF ϕ is **bad**, if there are 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.

- ϕ has bad $\implies \phi$ is unsatisfiable. Proof? by **Claim 47**
- ϕ has **no** bad variables $\implies \phi$ is satisfiable. Proof?

Algorithm 51

- 1 Pick **unassigned** literal $\ell \in V$ that has **no path** from ℓ to $\bar{\ell}$
- 2 Assign **1** to all literals reachable from ℓ , and **0** to their negations.
- 3 Repeat.

Can there be conflicts?

- Same iteration? No since ϕ has **no** bad variables
- Different iterations? No by **Claim 48**.

Poly-time algorithm for 2SAT

Algorithm 52 (TwoSatSolver)

Input: 2CNF ϕ

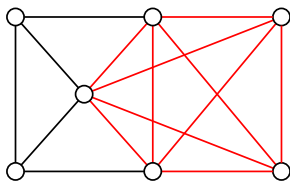
Return **TRUE** if there exist **no** bad variables in ϕ :

- Efficiency?
- Correctness?

Section 6

Clique

3SAT \leq_P CLIQUE



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

Claim 53

3SAT \leq_P CLIQUE

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

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

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

Proving $3SAT \leq_P CLIQUE$

Let ϕ be a 3CNF formula with k clauses.

On going example:

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

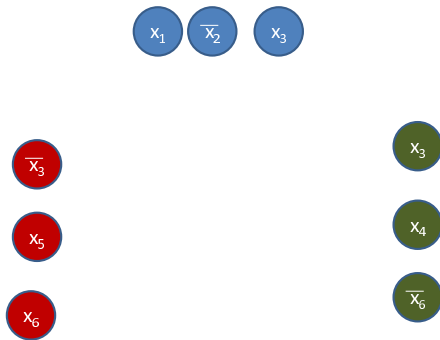
We define a graph $G = G(\phi)$ as follows:

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

Nodes of G

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

Add a node per **literal**

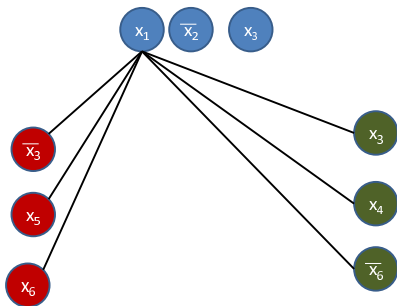


Edges of G

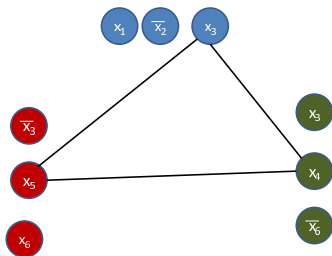
$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_3} \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$$

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

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



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

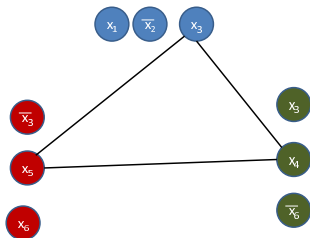


Proof: Suppose ϕ is satisfiable by an assignment ψ .
With respect to ψ :

- At least one literal is assigned to 1 in every clause (why?)
- Select a 1-literal in every tuple;
These literals can be joined by edges (why?)
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. (why?)
- 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 (why?)
Yielding a satisfying assignment to ϕ .



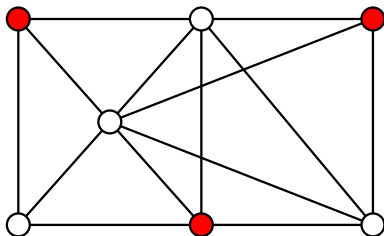
Recap

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

Section 7

Independent Set

Independent Set



Definition 54

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 55

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

Claim 56

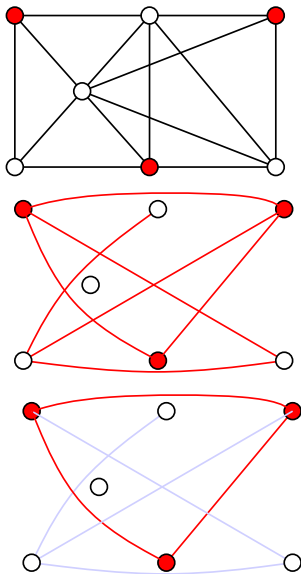
If U is an **independent set** in G , then U is a **clique** in G^c .

The reduction from **CLIQUE** to **IND-SET** simply compute the complement of the graph. ♣

Remark 57

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

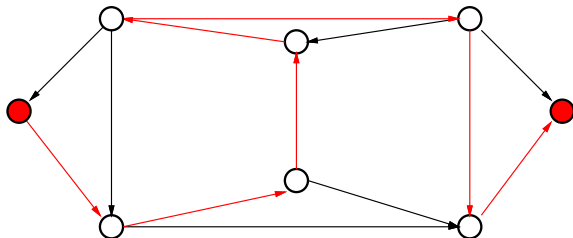
Independent Set \rightarrow Clique



Section 8

Hamiltonian Paths and Cycles

Reminder – Hamiltonian Path



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

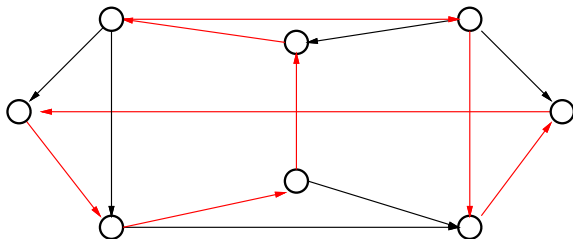
$$\text{HAMPATH} = \{ \langle G, s, t \rangle : G \text{ has Hamiltonian path from } s \text{ to } t \}$$

Theorem 58

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

Not today ...

Hamiltonian Cycle



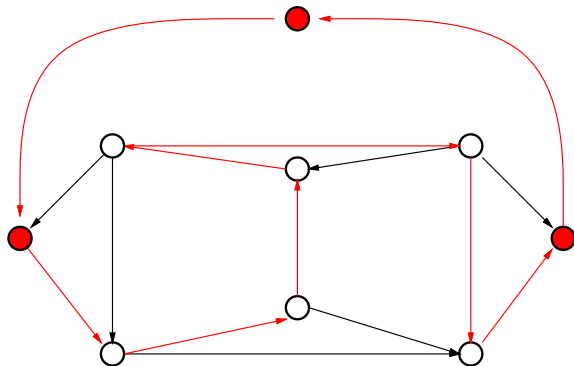
A **Hamiltonian Cycle** in a graph is an **Hamiltonian path** that ends up where it starts (i.e., $s = t$).

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

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.

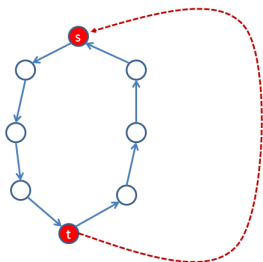
Proof's idea:



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 59

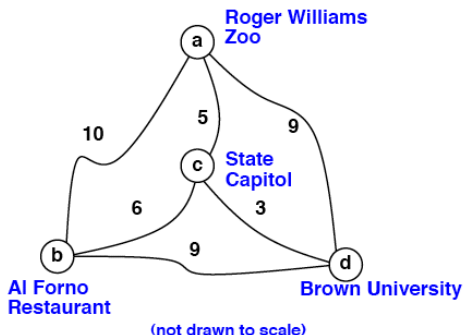
HAMCYCLE \leq_P HAMPATH.

Left as an easy (recommended) exercise.

Section 9

Traveling Salesman

Traveling Salesman



Input: set of cities C and set of inter-city distances D

Goal: find a hamiltonian cycle (TS tour) of total distance at most k

TRAVELING-SALESMAN =

$\{\langle C, D, k \rangle : (C, D) \text{ has a TS tour of total distance } \leq k\}$

HAMCYCLE \leq_P TRAVELING-SALESMAN

Proof: Given a directed graph $G = (V, E)$, construct traveling salesman instance.

- The cities are identical to the nodes of the original graph, $C = V$.
- The distance of going from v_1 to v_2 is 1 if $(v_1, v_2) \in E$, and 2 otherwise.
- The bound on the total distance of a tour is $k = |V|$.
- **correctness:**
 - \implies Suppose G has a Hamiltonian cycle.
The distance assigned by the reduction to all edges in this cycle is 1. Thus in (C, D) there is a traveling salesman tour of total distance $|V| = k$, namely $(C, D, k) \in \text{TRAVELING-SALESMAN}$.
 - \Leftarrow Suppose (C, D) has a traveling salesman tour of total distance $|V| = k$.
Tour cannot contain any edge of distance 2. Therefore it gives a Hamiltonian cycle in G .
- **Efficiency:** reduction done in quadratic time (filling up distances for all edges of the complete graph).



Chains of Reductions: NPC Problems

