Complexity

Richard Baron

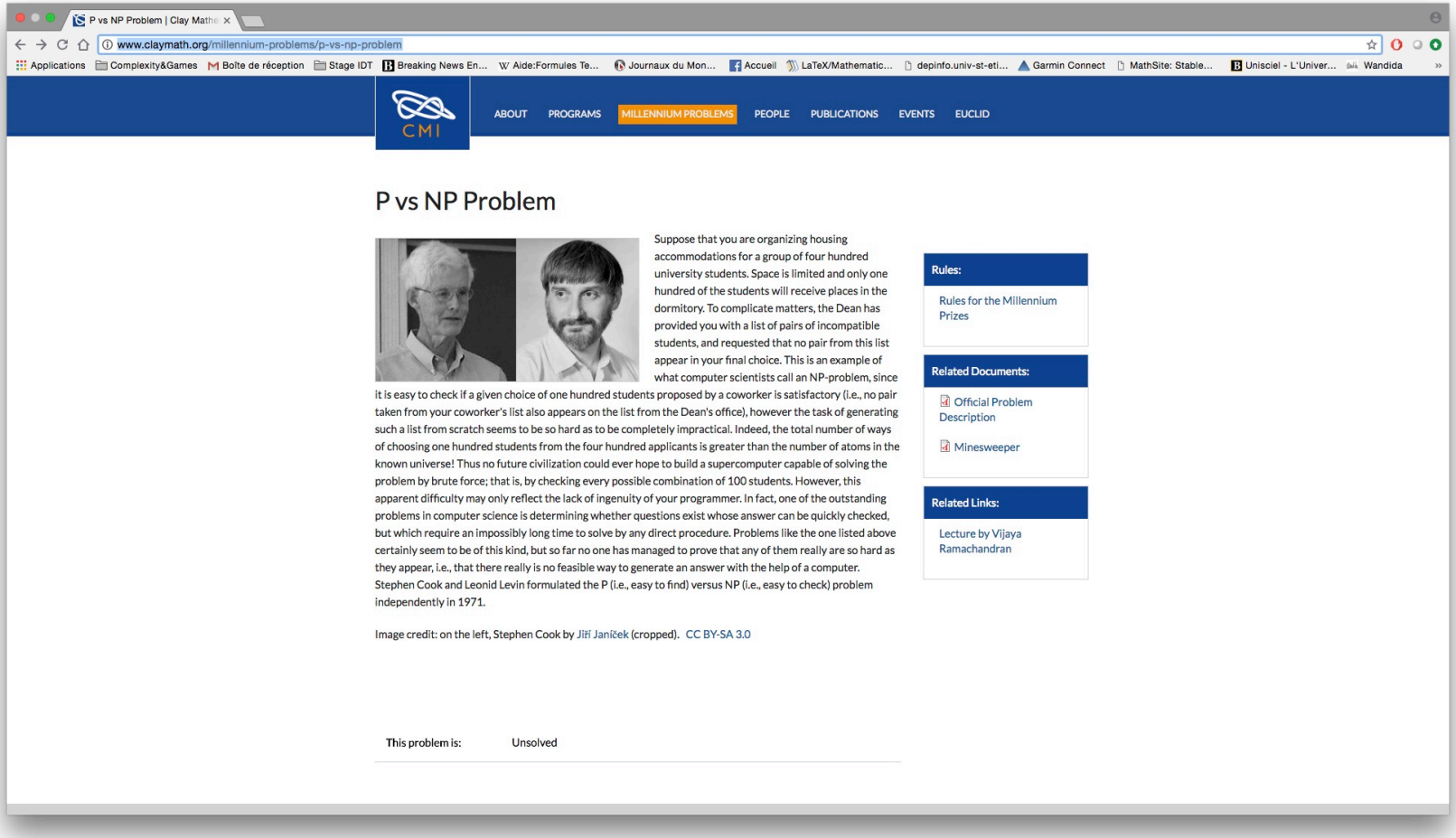
Year 2020-2021, Covid+1

Prerequisites

- Basic Maths
  - logic proofs
  - sets
  - exponentiation (and reverse)
- Propositional calculus
- Languages, Automaton
- Graphs (weighted, encoding)

Motivation

Clay Mathematics Institute : $1 million



Bibliography

- Thomas Cormen, Charles Leiserson, Ronald Rivest and Clifford Stein, Introduction to algorithms, Dunod 2nd edition (See course Advanced Algorithms)
- Michael R. Garey and David S. Johnson, Computers and Intractability, A guide to the theory of NP-Completeness, WH Freeman
- Christos Papadimitriou, Computational complexity, Addison Wesley
- Michael Sipser, Introduction to theory of computation, Thomson Course Technology, 2nd edition
- John Hopcroft, Rajeev Motwani and Jeffrey Ullman, Introduction to Automata theory, Languages and Computation, Pearson, 2nd edition

General notion : A Problem

Definition 1

(Decision problem) : A decision problem is defined by a set of instances I, and a subset $P \subseteq I$ called positive.

Intuitively, positive instances are "yes" answer to a given question

Example 1

Prime numbers : $I = \mathbb{N}$, $P = \{n \in \mathbb{N} | n \text{ is prime}\}$
Connected graphs : $I = \{G = (V, E) | G \text{ is a finite graph}\}$,
$P = \{G = (V, E) | G \text{ is connected}\}$
Automata acceptance : $I = \{(A, w) | A \text{ automaton}, w \text{ word}\}$,
$P = \{(A, w) | A \text{ accepts } w\}$

Numerous Real Problems are Optimization Problems

OPTIMIZATION PROBLEM
DATA: set X, objective function $f : X \mapsto \mathbb{Z}$
QUESTION: maximize/minimize f on X ($\max_{x \in X} f(x)$ or $\min_{x \in X} f(x)$)

(Associated) DECISION PROBLEM
DATA: set X, objective function $f : X \mapsto \mathbb{Z}$, $K \in \mathbb{Z}$
QUESTION: Is there $x \in X$ such that $f(x) \geq K$? (or $f(x) \leq K$ for the minimization prob.)

Notion

OPTIMIZATION is at least as difficult as DECISION

Proof:

Assume not: DECISION more difficult than OPTIMIZATION
e.g. algorithme A_{DEC} to solve DECISION in $O(n^k)$
and algorithm A_{OPT} to solve OPTIMIZATION in $O(n^{k'})$ with $k' < k$

then:

- apply A_{OPT}
- compare the output with K

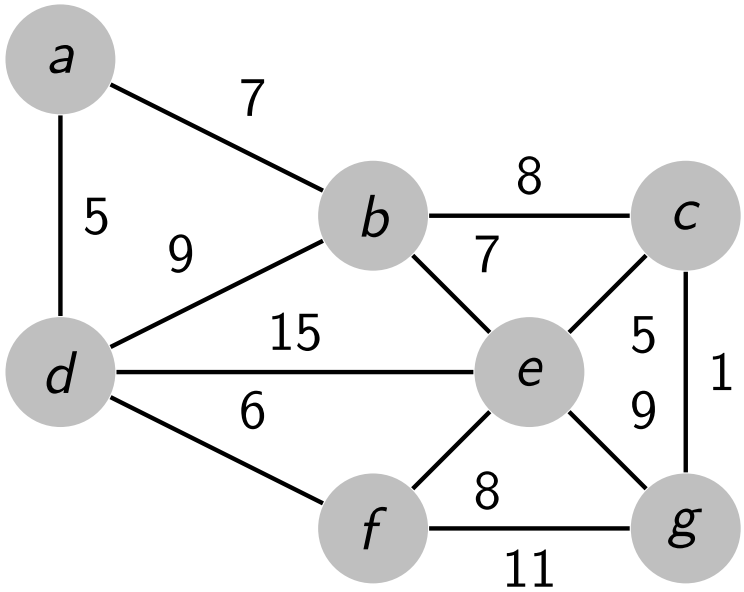
would be an algorithm in $O(n^{k'})$ for DECISION

In other words:

solve OPTIMIZATION $\Longrightarrow$ answer to DECISION

Converse: NO (e.g. TSP)

TRAVELLING SALESMAN PROBLEM
DATA: a weighted graph and a integer B
QUESTION: Is there a *tour* in this graph which length is less than B ?



Assume B=100. A_{DEC} outputs YES.
What answer for A_{OPT} ?

From Problems to Languages

Definition 2

Let Σ be an alphabet. An encoding (or encoding scheme) is a one-to-one function from I to Σ^*. The encoding of $x \in I$ is denoted $\langle x \rangle$. The language associated with P is $L_P = \{\langle x \rangle | x \in P\}$.

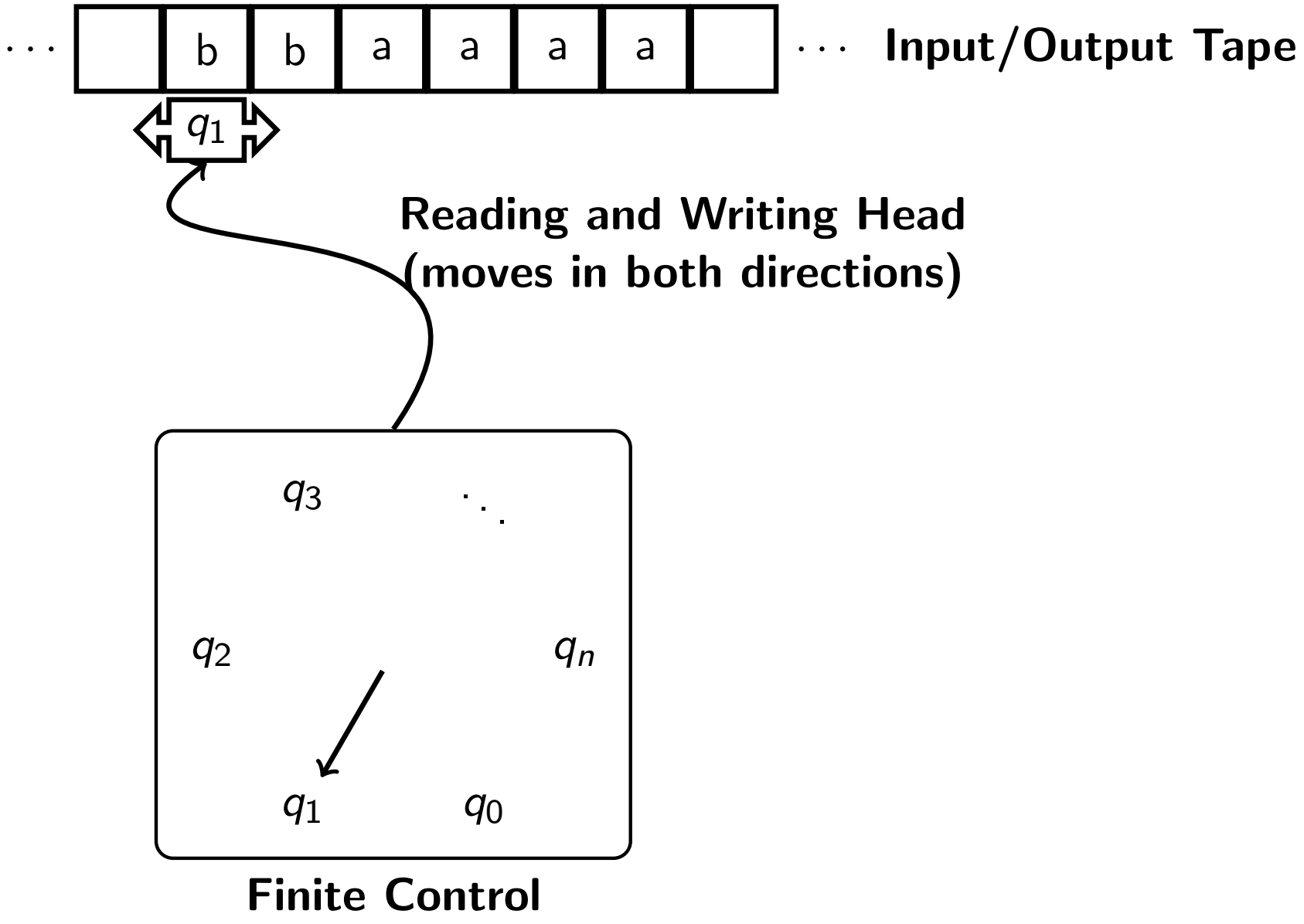
Example 2

Numbers : $\langle n \rangle$ = decimal or binary encoding of n.
Graphs (...)

Turing machine

Two elements : (a finite control + a infinite tape) linked by a read/write head
Tape divided into squares, or cells.
Cell's elements indexed by $\mathbb{N}$

Turing machine



Definition 3

A Turing machine is defined by the sextuplet $(Q, \Sigma, \Gamma, E, q_0, F, \#)$*:*

- *a non empty set of states* Q*;*
- *an input alphabet* Σ *with* $\# \notin \Sigma$ *;*
- *a tape alphabet* Γ *containing all the symbols that can be written on the tape. Of course,* $\Sigma \subseteq \Gamma$ *and* $\# \in \Gamma$*;*
- *a set of transitions* E *of the form* (p, a, q, b, x)*, with* $p \in Q, q \in Q, a \in \Gamma, b \in \Gamma, x \in \{\triangleleft, \triangleright\}$*. The transition may be represented by* $p, a \rightarrow q, b, x$*;*
- *an initial state* p_0*,* $p_0 \in Q$*;*
- *a set of final or accepting states* $F \subset Q$*;*
- *a blank symbol* $\#$ *that appears in all but a finite number of cells of the tape, those that hold input symbols.*

Compare with Automaton : $(Q, \Sigma, \delta, q_0, F)$

Turing machine

$\lhd$ (sometimes L): move of the read/write head to the left (one cell)
$\rhd$ (sometimes R): move of the read/write head to the right (one cell)
Transitions sometimes denoted by $p, a \to q, b, x$
! Keep in mind, a transition is element of $Q \times \Gamma \times Q \times \Gamma \times \{\lhd, \rhd\}$
Deterministic Turing Machine : at most **one** transition for each couple (p, a)
Non-deterministic Turing Machine : **several** possible transitions

Example 3

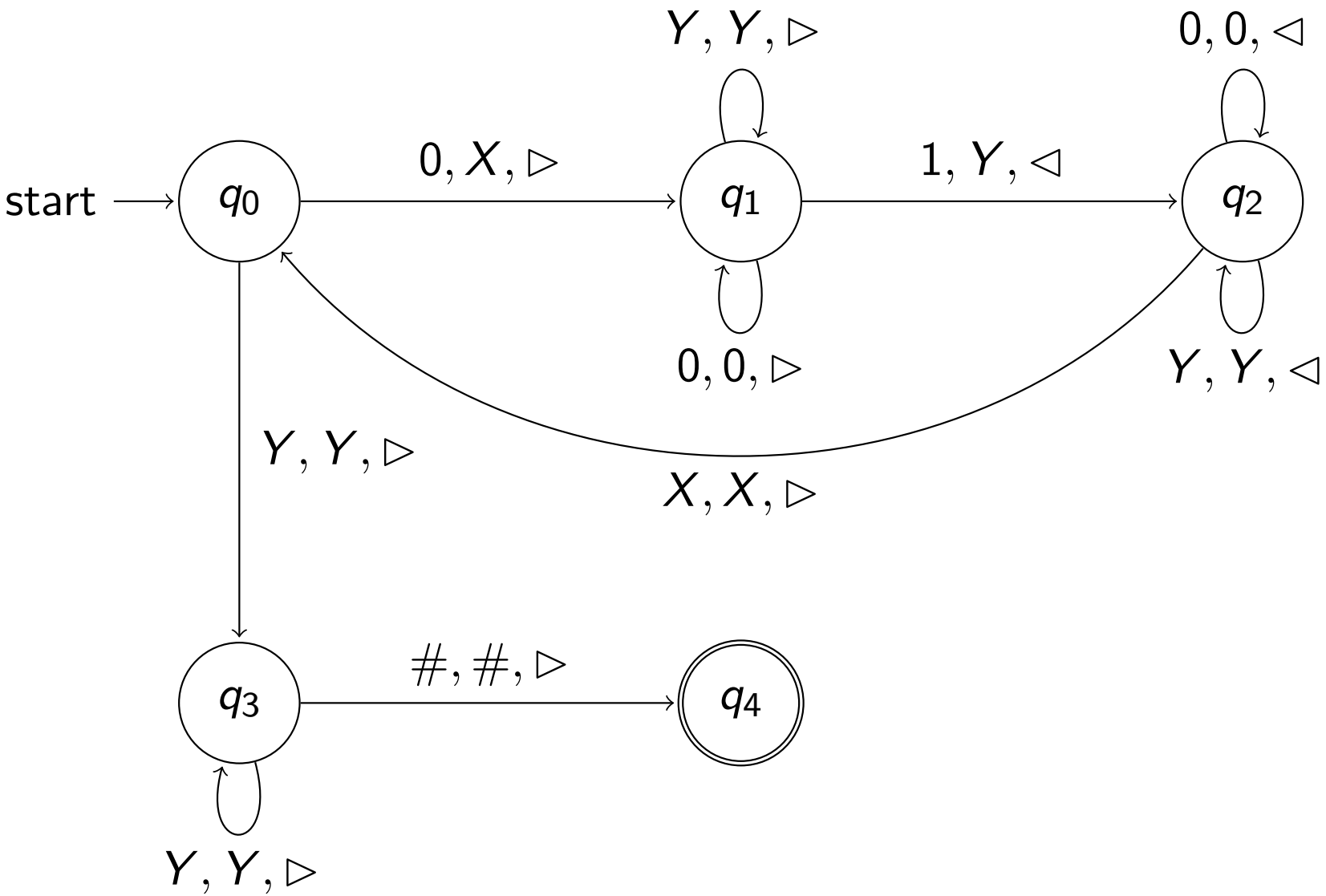
$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, \#\}, \delta, q_0, \{q_4\}, \#)$
with δ *defined by:*

	Symbol				
State	*0*	*1*	*X*	*Y*	#
q_0	$q_1, X, \triangleright$	—	—	$q_3, Y, \triangleright$	—
q_1	$q_1, 0, \triangleright$	$q_2, Y, \triangleleft$	—	$q_1, Y, \triangleright$	—
q_2	$q_2, 0, \triangleleft$	—	$q_0, X, \triangleright$	$q_2, Y, \triangleleft$	—
q_3	—	—	—	$q_3, Y, \triangleright$	$q_4, \#, \triangleright$
q_4	—	—	—	—	—

Turing machine

With $w = 0011$
$q_0 0011$, $X q_1 011$, $X 0 q_1 11$, $X q_2 0 Y 1$, $q_2 X 0 Y 1$, $X q_0 0 Y 1$, $XX q_1 Y 1$, $XXY q_1 Y 1$, $XX q_2 YY$, $X q_2 XYY$, $XX q_0 YY$, $XXY q_3 Y$, $XXYY q_3 \#$, $XXYY \# q_4 \#$
With $w = 011$
$q_0 011$, $X q_1 11$, $q_2 XY 1$, $X q_0 Y 1$, $XY q_3 1$

Turing machine



Turing machine

A configuration defined by :

1. state of the machine (element of Q);
2. symbols of the tape;
3. position of the read/write head.

Configuration uqv : Machine in state q, u word on the tape (before head), v word beginning at the position of the head.
One step (move) of a computation :

1. change the state;
2. write a new symbol on the tape;
3. move the read/write head of one position.

Definition 4

A step of a computation is a pair (C, C') denoted $C \to C'$ such that :

- *either $C = ucpav$ and$C' = uqcbv$ and the transition is $p, a \to q, b, \lhd$*
- *either $C = upav$ and$C' = ubqv$ and the transition is $p, a \to q, b, \rhd$*

Definition 5

A computation is a serie of successives configurations $C_0 \to C_1 \to \ldots \to C_k$. A computation is accepting if C_0 is an initial configuration, that is $C_0 = q_0 w$ with $w \in \Sigma^*$ and C_k is final, that is $C_k = uqv$ with $q \in F$.

Turing Machine :

- Accepting. input = word, output = Yes/No
- Computing. input = word, output : word(s)

Definition 6

$w \in \Sigma^*$ is accepted by a Turing machine $\mathcal{M}$ if there is an accepting computation with initial configuration $q_0 w$. The set of accepted words of $\mathcal{M}$ is denoted $L(\mathcal{M})$.

Note: Alternatively, F may be partitioned into F_Y and F_N or equivalently (Sipser) $F = \{q_{\text{accept}}, q_{\text{reject}}\}$

Definition 7

A two-way infinite tape machine is formally identical to the preceding model but with cell's tape indexed by $\mathbb{Z}$ (not $\mathbb{N}$)

Proposition 3.1

A two-way infinite tape machine is equivalent to a one-way infinite tape machine. Reciprocally, A one-way infinite tape machine is equivalent to a two-way infinite tape machine.

Proof:

Initial tape (bi-infinite) for M:

	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	
...	#	#	X	a	b	Y	A	b	a	#	#	#	...

Simulated by M':

0	1	2	3	4	5	6	
Y	A	b	a	#	#	#	...
$	b	a	X	#	#	#	...

and:

- $\Gamma' = \Gamma \times (\Gamma \cup \{\$\})$ (head read both symbols of new cells)
- $\Sigma' = \Sigma \times \{\$, \#\}$ (input is encoded on **upper part** of cells)
- $\#' = (\#, \#)$
- $Q' = Q \times \{\uparrow, \downarrow\}$
- $q_0' = (q_0, \uparrow)$
- $F' = F \times \{\uparrow, \downarrow\}$

Initial tape (bi-infinite) for M:

	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	
...	#	#	X	a	b	Y	A	b	a	#	#	#	...

Simulated by M':

0	1	2	3	4	5	6	
Y	A	b	a	#	#	#	...
$	b	a	X	#	#	#	...

New transitions in E' ?

For all $(p, a, q, b, \triangleright) \in E$

Then we create:

$((p, \uparrow), (a, .), (q, \uparrow), (b, .), \triangleright)$ **and** $((p, \downarrow), (., a), (q, \downarrow), (., b), \triangleleft)$ in E' (. stands for **any** (all) symbol of Γ)

Initial tape (bi-infinite) for M:

	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	
...	#	#	X	a	b	Y	A	b	a	#	#	#	...

Simulated by M':

	0	1	2	3	4	5	6	
	Y	A	b	a	#	#	#	...
	$	b	a	X	#	#	#	...

New transitions in E' ?

For all $(p, a, q, b, \lhd) \in E$

Then we create (if $. \neq \$$ (see Special cases below)):

$((p, \uparrow), (a, .), (q, \uparrow), (b, .), \lhd)$ **and** $((p, \downarrow), (., a), (q, \downarrow), (., b), \rhd)$ in E' (. stands for any symbol of Γ)

two-way infinite Turing machine

Initial tape (bi-infinite) for M:

	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	
...	#	#	X	a	b	Y	A	b	a	#	#	#	...

Simulated by M':

0	1	2	3	4	5	6	
Y	A	b	a	#	#	#	...
$	b	a	X	#	#	#	...

Problem: Special cases for cell 0

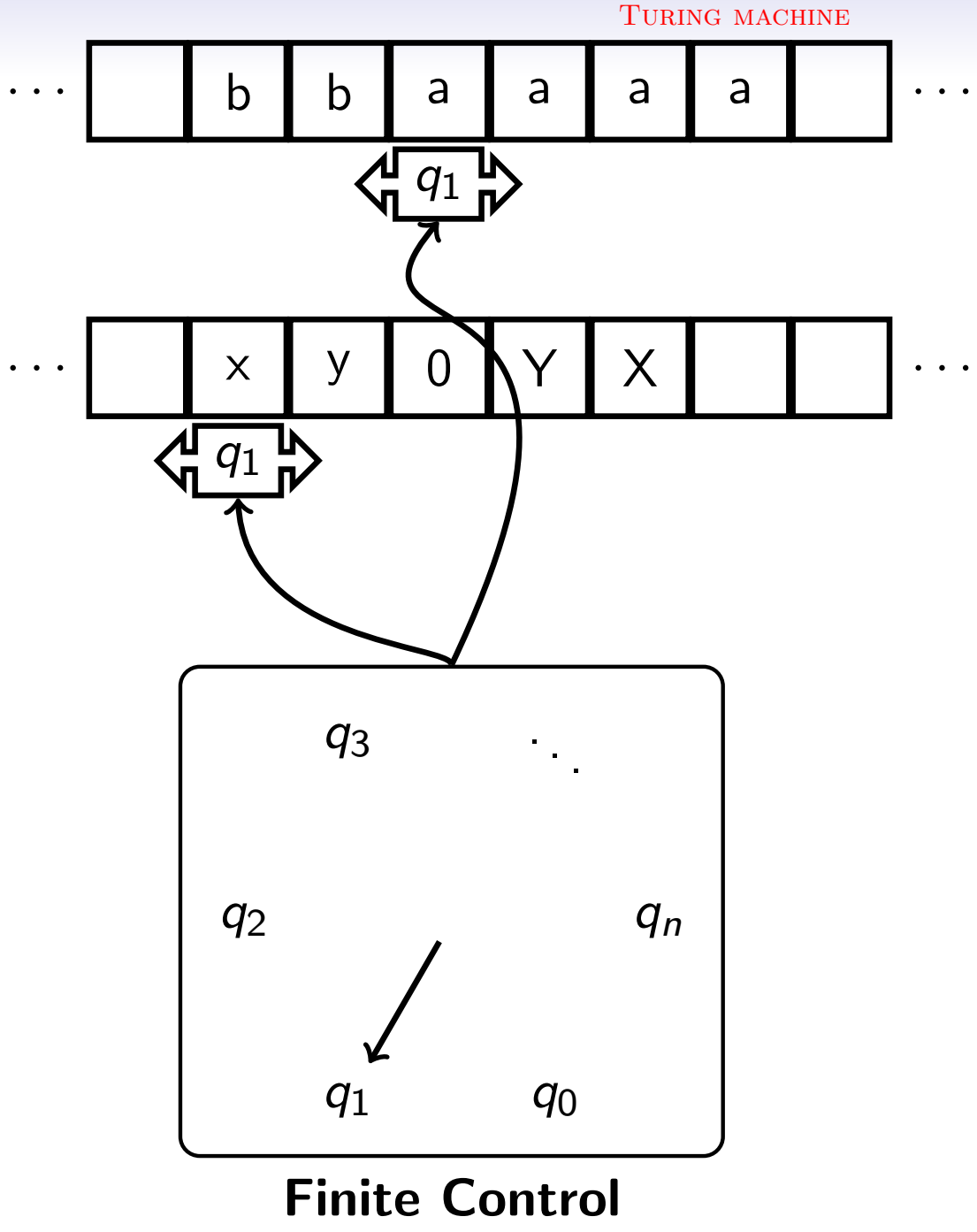
- For all $(p, a, q, b, \lhd) \in E$ then create $((p, \uparrow), (a, \$), (q, \downarrow), (b, \$), \rhd) \in E'$
- For all $(p, a, q, b, \lhd) \in E$ then Create $((p, \uparrow), (a, \$), (q, \downarrow), (b, \$), \rhd) \in E'$
- Create $((p, \downarrow), (., \$), (p, \uparrow), (., \$), \triangledown) \in E'$ (. stands for any symbol of Γ) ($\triangledown = \rhd + \lhd$ and . stands for any symbol of Γ) (follows a $((p, \downarrow), (., x), (q, \downarrow), (., y), \lhd)$ step on cell 1 where . stands for any symbol of Γ)

Definition 8

A multitape machine has k tapes, with k corresponding read/write head. A transition is an element of the set
$Q \times \Gamma^k \times Q \times \Gamma^k \times \{\triangleright, \triangleleft, \triangledown\}^k$

Proposition 3.2

Every multitape Turing machine $\mathcal{M}$ is equivalent to one-tape Turing machine $\mathcal{M}'$ that accepts the same inputs.


Turing machine
b
b
a
a
a
a
q_1
x
y
0
Y
X
q_1
q_3
q_2
q_n
q_1
q_0
Finite Control

Turing machine

solution 1:

- $\mathcal{M}'$ stores contents of k tapes separated by new symbol $\$ \notin \Gamma$
- to keep track of heads locations, $\mathcal{M}'$ inserts a symbol $\uparrow$ before every symbols located under one head
- $\mathcal{M}'$ scans the whole content
- $\mathcal{M}'$ applies the required transition of state
- $\mathcal{M}'$ makes second pass to update (k) contents

solution 2: instead of $\uparrow$, new "dotted" symbols are added to Γ and indicate location of one head

solution 3: k tapes are replaced by one, where each cell contains a symbol of Γ^k (same idea as the bi-infinite machine). Additionally, k symbols 0/1 are added to keep track of heads locations

Solution 1

Tape 1			↓					
	0	1	2	3	4	5	6	
	b	b	a	a	a	a	#	...
Tape 2	↓							
	0	1	2	3	4	5	6	
	x	y	x	y	x	#	#	...

Simulated by

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
b	b	↑	a	a	a	a	$	↑	x	y	x	y	x	#	...

Solution 1

		↓					
b	b	a	a	a	a	#	. . .
↓							
x	y	x	y	x	#	#	. . .

For M':

b	b	↑	a	a	a	a	$	↑	x	y	x	y	x	#	. . .

Transition to:

			↓				
b	b	b	a	a	a	#	. . .
	↓						
X	y	x	y	x	#	#	. . .

For M':

b	b	b	↑	a	a	a	$	X	↑	y	x	y	x	#	. . .

Solution 1

		↓					
b	b	a	a	a	a	#	. . .
↓							
x	y	x	y	x	#	#	. . .

For M':

b	b	↑	a	a	a	a	$	↑	x	y	x	y	x	#	. . .

Transition to:

			↓				
b	b	b	a	a	a	#	. . .
	↓						
X	y	x	y	x	#	#	. . .

For M':

b	b	b	↑	a	a	a	$	X	↑	y	x	y	x	#	. . .

Solution 1

		↓					
b	b	a	a	a	a	#	. . .
↓							
x	y	x	y	x	#	#	. . .

For M':

b	b	↑	a	a	a	a	$	↑	x	y	x	y	x	#	. . .

Transition to:

			↓				
b	b	b	a	a	a	#	. . .
	↓						
X	y	x	y	x	#	#	. . .

For M':

b	b	b	↑	a	a	a	$	X	↑	y	x	y	x	#	. . .

Solution 1

		↓					
b	b	a	a	a	a	#	...
↓							
x	y	x	y	x	#	#	...

For M':

b	b	↑	a	a	a	a	$	↑	x	y	x	y	x	#	...

Transition to:

			↓				
b	b	b	a	a	a	#	...
	↓						
X	y	x	y	x	#	#	...

For M':

b	b	b	↑	a	a	a	$	X	↑	y	x	y	x	#	...

Solution 2

Tape 1			↓					
	0	1	2	3	4	5	6	
	b	b	a	a	a	a	#	
Tape 2	↓							
	0	1	2	3	4	5	6	
	x	y	x	y	x	#	#	...

Simulated by

0	1	2	3	4	5	6	7	8	9	10	11	12	13
b	b	$\dot{a}$	a	a	a	$	$\dot{x}$	y	x	y	x	#	...

Solution 2

		↓					
b	b	a	a	a	a	#	. . .
↓							
x	y	x	y	x	#	#	. . .

For M':

b	b	$\dot{a}$	a	a	a	$	$\dot{x}$	y	x	y	x	#	. . .

Transition to:

			↓				
b	b	b	a	a	a	#	. . .
	↓						
X	y	x	y	x	#	#	. . .

For M':

b	b	b	$\dot{a}$	a	a	$	X	$\dot{y}$	x	y	x	#	. . .

Solution 2

		↓					
b	b	a	a	a	a	#	. . .
↓							
x	y	x	y	x	#	#	. . .

For M':

b	b	$\dot{a}$	a	a	a	$	$\dot{x}$	y	x	y	x	#	. . .

Transition to:

			↓				
b	b	b	a	a	a	#	. . .
	↓						
X	y	x	y	x	#	#	. . .

For M':

b	b	b	$\dot{a}$	a	a	$	X	$\dot{y}$	x	y	x	#	. . .

Solution 2

		↓					
b	b	a	a	a	a	#	...
↓							
x	y	x	y	x	#	#	...

For M':

b	b	$\dot{a}$	a	a	a	$	$\dot{x}$	y	x	y	x	#	...

Transition to:

			↓				
b	b	b	a	a	a	#	...
	↓						
X	y	x	y	x	#	#	...

For M':

b	b	b	$\dot{a}$	a	a	$	X	$\dot{y}$	x	y	x	#	...

Solution 2

		↓					
b	b	a	a	a	a	#	. . .
↓							
x	y	x	y	x	#	#	. . .

For M':

b	b	$\dot{a}$	a	a	a	$	$\dot{x}$	y	x	y	x	#	. . .

Transition to:

			↓				
b	b	b	a	a	a	#	. . .
	↓						
X	y	x	y	x	#	#	. . .

For M':

b	b	b	$\dot{a}$	a	a	$	X	$\dot{y}$	x	y	x	#	. . .

Solution 3

			↓					
Tape 1	0	1	2	3	4	5	6	
	b	b	a	a	a	a	#	

	↓						
Tape 2	0	1	2	3	4	5	6	
	x	y	x	y	x	#	#	...

Simulated by

	↓						
0	1	2	3	4	5	6	
b	b	a	a	a	a	#	...
0	0	1	0	0	0	0	
x	y	x	y	x	#	#	...
1	0	0	0	0	0	0	

M' reads symbol b0y0

Solution 3

		↓					
b	b	a	a	a	a	#	. . .
↓							
x	y	x	y	x	#	#	. . .

For M':

b	b	a	a	a	a	#	. . .
0	0	1	0	0	0	0	
x	y	x	y	x	#	#	. . .
1	0	0	0	0	0	0	

Transition to:

			↓				
b	b	b	a	a	a	#	. . .
	↓						
X	y	x	y	x	#	#	. . .

For M':

b	b	b	a	a	a	#	. . .
0	0	0	1	0	0	0	
X	y	x	y	x	#	#	. . .
0	1	0	0	0	0	0	

Solution 3

		↓					
b	b	a	a	a	a	#	. . .
↓							
x	y	x	y	x	#	#	. . .

For M':

b	b	a	a	a	a	#	. . .
0	0	1	0	0	0	0	
x	y	x	y	x	#	#	. . .
1	0	0	0	0	0	0	

Transition to:

			↓				
b	b	b	a	a	a	#	. . .
	↓						
X	y	x	y	x	#	#	. . .

For M':

b	b	b	a	a	a	#	. . .
0	0	0	1	0	0	0	
X	y	x	y	x	#	#	. . .
0	1	0	0	0	0	0	

Solution 3

		↓					
b	b	a	a	a	a	#	...
↓							
x	y	x	y	x	#	#	...

For M':

b	b	a	a	a	a	#	...
0	0	1	0	0	0	0	
x	y	x	y	x	#	#	...
1	0	0	0	0	0	0	

Transition to:

			↓				
b	b	b	a	a	a	#	...
	↓						
X	y	x	y	x	#	#	...

For M':

b	b	b	a	a	a	#	...
0	0	0	1	0	0	0	
X	y	x	y	x	#	#	...
0	1	0	0	0	0	0	

Solution 3

		↓					
b	b	a	a	a	a	#	...
↓							
x	y	x	y	x	#	#	...

For M':

b	b	a	a	a	a	#	...
0	0	1	0	0	0	0	
x	y	x	y	x	#	#	...
1	0	0	0	0	0	0	

Transition to:

			↓				
b	b	b	a	a	a	#	...
	↓						
X	y	x	y	x	#	#	...

For M':

b	b	b	a	a	a	#	...
0	0	0	1	0	0	0	
X	y	x	y	x	#	#	...
0	1	0	0	0	0	0	

Proposition 3.3

For every Turing machine $\mathcal{M}$ there exists a Turing machine $\mathcal{M}'$ such that :

1. *$L(\mathcal{M})$=$L(\mathcal{M}')$ and $\mathcal{M}$ halts if and only if $\mathcal{M}'$ halts ;*
2. *$\mathcal{M}'$ has two states q_+ and q_- such that :*
   - *$F' = \{q_+\}$*
   - *$\mathcal{M}'$ always halts in q_+ and q_-*
   - *$\mathcal{M}'$ halts only in either q_+ or q_-*

Proof

Let $\mathcal{M} = (Q, \Sigma, \Gamma, E, q_0, F, \#)$

Define $\mathcal{M}' = (Q', \Sigma, \Gamma', E', q_0', F', \#)$ with:

- $Q' = (Q \setminus F) \cup \{q_0', q_1', q_+, q_-\}$
- $\Sigma' = \Sigma$
- $\Gamma' = \Gamma \cup \{\bar{a} |\ a \in \Gamma\}$
- initial state q_0'
- $F' = \{q_+\}$
- blanck symbol $\#$

Proof

Now define E' the set of transitions for $\mathcal{M}'$:
$E' = E_0 \cup E_1 \cup E_2 \cup E_3$ depending on the state p of $\mathcal{M}'$

- $p = q'_0$ then $E_0 = \{q'_0, a \to q'_1, \bar{a}, \triangleright \,|a \in \Gamma\}$
- $p = q'_1$ then $E_1 = \{q'_1, a \to q_0, a, \triangleleft \,|a \in \Gamma\}$
- $p = q_+$ or $p = q_-$ NO transition ($\mathcal{M}'$ must halt)

- $p \in Q \setminus F$

$$E_2 = \{p, a \to q, b, x | p, a \to q, b, x \in E, q \notin F\}$$
$$\cup \{p, \bar{a} \to q, \bar{b}, \triangleright \, | p, a \to q, b, \triangleright \in E, q \notin F\}$$
$$\cup \{p, a \to q_+, b, x | p, a \to q, b, x \in E, q \in F\}$$
$$\cup \{p, \bar{a} \to q_+, \bar{b}, \triangleright \, | p, a \to q, b, \triangleright \in E, q \in F\}$$

$$E_3 = \{p, a \to q_-, a, \triangleright \mid \text{if } (p, a) \in U\}$$
$$\cup \{p, \bar{a} \to q_-, \bar{a}, \triangleright \mid \text{if } (p, a) \in U\}$$
$$\cup \{p, \bar{a} \to q_-, \bar{a}, \triangleright \mid \text{if } (p, a) \in V\}$$

where $U = \{(p, a) \in Q \times \Gamma | (p, a, q, b, x) \notin E, x \in \{\triangleleft, \triangleright\}\}$ and $V = \{(p, a) \in Q \times \Gamma | (p, a, q, b, \triangleright) \notin E\}$

Notice: with $(p, a, q, b, \triangleright) \notin E\}$, $(p, a, q, b, \triangleleft) \in E$ is still possible ! Last line of E_3 catches these cases

- $p \in Q \setminus F$

$$E_2 = \{p, a \to q, b, x | p, a \to q, b, x \in E, q \notin F\}$$
$$\cup \{p, \bar{a} \to q, \bar{b}, \rhd \, | p, a \to q, b, \rhd \in E, q \notin F\}$$
$$\cup \{p, a \to q_+, b, x | p, a \to q, b, x \in E, q \in F\}$$
$$\cup \{p, \bar{a} \to q_+, \bar{b}, \rhd \, | p, a \to q, b, \rhd \in E, q \in F\}$$

$$E_3 = \{p, a \to q_-, a, \rhd \mid \text{if } (p, a) \in U\}$$
$$\cup \{p, \bar{a} \to q_-, \bar{a}, \rhd \mid \text{if } (p, a) \in U\}$$
$$\cup \{p, \bar{a} \to q_-, \bar{a}, \rhd \mid \text{if } (p, a) \in V\}$$

where $U = \{(p, a) \in Q \times \Gamma | (p, a, q, b, x) \notin E, x \in \{\lhd, \rhd\}\}$ and
$V = \{(p, a) \in Q \times \Gamma | (p, a, q, b, \rhd) \notin E\}$
Notice: with $(p, a, q, b, \rhd) \notin E\}$, $(p, a, q, b, \lhd) \in E$ is still possible
! Last line of E_3 catches these cases

Definition 9

A Non-deterministic Turing machine is defined by the sextuplet $(Q, \Sigma, \Gamma, \Delta, q_0, F, \#)$*:*

- *a non empty set of states* Q*;*
- *an input alphabet* Σ *with* $\# \notin \Sigma$ *;*
- *a tape alphabet* Γ *containing all the symbols that can be written on the tape. Of course,* $\Sigma \subseteq \Gamma$ *and* $\# \in \Gamma$*;*
- *an transition relation* $\Delta \subseteq Q \times \Gamma \times Q \times \Gamma \times \{\triangleleft, \triangleright\}$
- *an initial state* p_0*,* $p_0 \in Q$*;*
- *a set of final or accepting states* $F \subset Q$*;*
- *a blank symbol* $\#$ *that appears in all but a finite number of cells of the tape, those that hold input symbols.*

Non-deterministic : accepting if **at least** one accepting computation

Example: Let M a Turing machine be defined by:

$$(Q = \{q_0, q_1, q_2\}, \Sigma = \{0, 1\}, \Gamma = \{0, 1, B\}, \delta, q_0, F = \{q_2\}, B)$$

Transition function δ is defined as follows:

	Symbol		
State	0	1	B
q_0	$q_1, 1, \triangleright$	$q_1, 0, \triangleright$	—
q_1	—	$q_1, 1, \triangleright$	$q_2, B, \triangleright$
	—	$q_0, 1, \triangleleft$	
q_2	—	—	—

What if $C_0 = q_0111$?

Turing machine

$q_0111 \rightarrow \quad 0q_111 \rightarrow \quad 01q_11 \rightarrow \quad 011q_1B \rightarrow \quad 011Bq_2B$

$q_0111 \rightarrow \quad 0q_111 \rightarrow \quad 01q_11 \rightarrow \quad 0q_011 \rightarrow \quad 00q_11 \rightarrow \quad 001q_1B \rightarrow \quad 001Bq_2B$

$q_0111 \rightarrow \quad 0q_111 \rightarrow \quad \mathbf{01q_11} \rightarrow \quad 0q_011 \rightarrow \quad 00q_11 \rightarrow \quad 0q_001 \rightarrow \quad \mathbf{01q_11} \rightarrow \quad \text{loop}$

$q_0111 \rightarrow \quad 0q_111 \rightarrow \quad q_0011 \rightarrow \quad 1q_111 \rightarrow \quad 11q_11 \rightarrow \quad 111q_1B \rightarrow \quad 111Bq_2B$

$q_0111 \rightarrow \quad 0q_111 \rightarrow \quad q_0011 \rightarrow \quad 1q_111 \rightarrow \quad 11q_11 \rightarrow \quad 1q_011 \rightarrow \quad 10q_11 \rightarrow \quad 101q_1B \rightarrow \quad 101Bq_2B$

$q_0111 \rightarrow \quad 0q_111 \rightarrow \quad q_0011 \rightarrow \quad 1q_111 \rightarrow \quad 11q_11 \rightarrow \quad 1q_011 \rightarrow \quad 10q_11 \rightarrow \quad 1q_001B \rightarrow \quad \mathbf{11q_11} \rightarrow \quad \text{loop}$

$q_0111 \rightarrow \quad 0q_111 \rightarrow \quad q_0011 \rightarrow \quad 1q_111 \rightarrow \quad q_0111 \rightarrow \quad \mathbf{0q_111} \rightarrow \quad \text{loop}$

Proposition 3.4

Every non-deterministic Turing machine $\mathcal{M}$ is equivalent to a deterministic Turing machine $\mathcal{M}'$. If $\mathcal{M}$ has no infinite computation, then $\mathcal{M}'$ has no infinite computation.

Proof:

Idea: $\mathcal{M}'$ will simulate all possible computations of $\mathcal{M}$
BUT must not get trapped into infinite ones (if any)
$\mathcal{M}'$ has 3 tapes:

1. store input word x
2. encode the choices made during simulation
3. working tape

Define $r = \max_{q \in Q, a \in \Gamma} | \{(q, a, q', z, Z) \in \Delta\}|$
r is the greatest number of possible transitions of the NDTM $\mathcal{M}$, for all combinations of state q and symbol a on the tape (for all cells of transition table)

Step t of DTM $\mathcal{M}$' :

- erase content of 3rd tape, then copy input word x on this tape
- generate the string $y \in \{1, \ldots, r\}^*$ numbered t (in lexicographic order) $y = m_{i_1} m_{i_2} \ldots m_{i_l}$ and write it on 2nd tape
- simulate the computation of NDTM $\mathcal{M}$ for at most l steps (recall $l = |y|$). At step j $(1 \leq j \leq l)$ of this simulation, use m_{i_j} to select which transition (of NDTM $\mathcal{M}$) is to be applied. If less than m_{i_j} possible transitions, then skip to $t + 1$
- if simulation of NDTM $\mathcal{M}$ is such that $\mathcal{M}$ reaches an accepting state, then DTM $\mathcal{M}'$ accepts x. Else, $\mathcal{M}'$ skips to step $t + 1$.

Comment: $\{1, \ldots, r\}^*$ is the (infinite) set of all strings written with symbols of $\{1, \ldots, r\}$.
Lexicographic ordering of $\{1, \ldots, 4\}^*$ is

$$\{1, 2, 3, 4, 11, 12, 13, 14, 21, 22, 23, 24, 31, 32, \ldots \\ \ldots, 43, 44, 111, 112, 113, 114, 121, 122, \ldots\} \tag{1}$$

Number of operations of this simulation ?
Assume executions $\mathcal{M}$ is finite, bounded by $\bar{l}$ (number of steps)
Time for simulation by $\mathcal{M}'$?
A string $y = m_{i_1} m_{i_2} \dots m_{i_l}$ (thus $|y| = l$) encodes an execution (of $\mathcal{M}$) in **at most** l steps
Thus, total time for this simulation by $\mathcal{M}'$ is bounded by $T = \sum_{l=1}^{\bar{l}} l r^l$
Rewrite $T = r \sum_{l=1}^{\bar{l}} l r^{l-1}$
Let $f_n(r) = \sum_{l=1}^{n} l r^{l-1}$.

$$
\begin{aligned}
(1-r)f_n(r) &= \sum_{l=1}^{n} l r^{l-1} - \sum_{l=1}^{n} l r^{l} \\
&= \sum_{j=0}^{n-1} (j+1) r^{j} - \sum_{j=1}^{n} j r^{j} \\
&= r^0 + \sum_{j=1}^{n-1} \left((j+1) r^{j} - j r^{j} \right) - n r^{n} \\
&= r^0 + \sum_{j=1}^{n-1} r^{j} - n r^{n} \\
&= \sum_{j=0}^{n-1} r^{j} - n r^{n} \\
&= r^0 \frac{1-r^n}{1-r} - n r^{n}
\end{aligned}
$$

Then

$$f_n(r) = \frac{1 - r^n}{(1-r)^2} - n\frac{r^n}{1-r}$$

and

$$T = r\left[\frac{1 - r^{\bar{l}}}{(1-r)^2} - \bar{l}\frac{r^{\bar{l}}}{1-r}\right]$$

That is $T = O(r^{\bar{l}})$
Recalling:

$$\begin{aligned} r^{\alpha} &= \exp(\alpha \log(r)) \\ &= \exp(\alpha \log_2(r) \log(2)) \\ &= 2^{\alpha \log_2(r)} \end{aligned}$$

we have

$$T = O(r^{\bar{l}}) = O(2^{\bar{l} \log_2(r)}) = O(2^{\bar{l}})$$

Thesis 1

A function computable in any reasonable computational model is computable by a Turing machine

reasonable computational model?

- computation defined by a set of finite instructions;
- each instruction carried out in a finite amount of time...
- ... in a deterministic manner

Definition 10 (Complexity)

Let $\gamma = q_0 w \to C_0 \to C_1 \to \ldots \to C_m$ be a computation of a Turing machine $\mathcal{M}$ on an input word w.

1. *the time $t_{\mathcal{M}}(\gamma)$ of this computation is m*
2. *the space $s_{\mathcal{M}}(\gamma)$ is the number of cells visited by the head during the computation*

Complexity for an input w:

$$t_{\mathcal{M}}(w) = \max_{\gamma} t_{\mathcal{M}}(\gamma) \quad \text{et} \quad s_{\mathcal{M}}(w) = \max_{\gamma} s_{\mathcal{M}}(\gamma)$$

Complexity (at worst case):

$$t_{\mathcal{M}}(n) = \max_{|w|=n} t_{\mathcal{M}}(w) \quad \text{et} \quad s_{\mathcal{M}}(n) = \max_{|w|=n} s_{\mathcal{M}}(w)$$

Lemma 1

For every one tape Turing machine $\mathcal{M}$, there exists a constant K s.t.

$$s_{\mathcal{M}}(n) \leq t_{\mathcal{M}}(n) \quad et \quad t_{\mathcal{M}}(n) \leq 2^{Ks_{\mathcal{M}}(n)}$$

Proof: For a computation with input word w, any configuration C_i composed of symbols of Γ and Q (only one state), that is
$C_i \in (\Gamma \cup Q)^*$
Only required symbols: $C_i = 001Bq_1BBBB\ldots$

Proof: Finite computation $\implies$ No loop
Thus: number of configurations greater than number of steps (if not...)
Number of configurations of length at most $s_M(w)$: $|Q \cup \Gamma|^{s_M(w)+1}$ (think of $0001q_1B$)
That is: $|Q \cup \Gamma|^{s_M(w)+1} \geq t_M(w) + 1$ ($+1$ because of C_0)
we can choose $k \in \mathbb{R}$ such that $ks_M(w) > s_M(w) + 1$ and obtain: $t_M(w) \leq t_M(w) + 1 \leq |Q \cup \Gamma|^{ks_M(w)}$

$$\begin{aligned} t_M(w) &\leq \exp[ks_M(w)\log(|Q \cup \Gamma|] \\ &\leq \exp[s_M(w)k\log_2(|Q \cup \Gamma|)\log(2)] \\ &\leq 2^{K.s_M(w)} \end{aligned}$$

with $K = k\log_2(|Q \cup \Gamma|)$
Inequality holds for all w thus for $t_M(n)$ and $s_M(n)$

Bi-infinite tape TM: one transition of the bi-infinite tape TM = one transition for the simulating TM

Multi-tape TM:

Proposition 4.1

Every k-tape Turing machine $\mathcal{M}$ is equivalent to a one-tape Turing machine $\mathcal{M}'$ s.t.

$$t_{\mathcal{M}'}(n) = \mathcal{O}(t^2_{\mathcal{M}}(n))$$

Non-deterministic TM: r maximal cardinality of the sets $\delta(p, a) = \{(q, b, x) | p, a \to q, b, x \in E\}, \forall p \in Q, a \in \Gamma$

Proposition 4.2

Every (non-deterministic) Turing machine $\mathcal{M}$ is equivalent to a determistic Turing machine $\mathcal{M}'$ s.t.

$$t_{\mathcal{M}'}(n) = 2^{\mathcal{O}(t_{\mathcal{M}}(n))}$$

Definition 11

Given $f : \mathbb{N} \to \mathbb{R}^+$, we define :

- *$TIME(f(n))$ the set of problems decided by a deterministic Turing machine in time $\mathcal{O}(f(n))$*
- *$NTIME(f(n))$ the set of problems decided by a non-deterministic Turing machine in time $\mathcal{O}(f(n))$*

Definition of classes :

$$P = \bigcup_{k \geq 0} TIME(n^k)$$

$$NP = \bigcup_{k \geq 0} NTIME(n^k)$$

Proposition 4.3

$$P \subseteq NP$$

Alternative view

Cormen Leiserson

Definition 12

A polynomial-time verifier for a language L is a deterministic Turing machine $\mathcal{M}$ which accepts input $\langle w, c\rangle$ in polynomial time in $|w|$ such that $L = \{w | \exists c, \langle w, c\rangle \in L(\mathcal{M})\}$

Polynomial time in $|w| \Rightarrow$ size of c is polynomial

$c =$ certificate/witness

Algorithmically more intuitive

$|c| = 2^{|w|}$ No!

Proposition 4.4

Language L is in NP if and only if there exists a polynomial-time verifier for L.

Alternative view

Proof:

First, assume $L \in NP$. There is non-deterministic machine $\mathcal{M}$ which accepts L in polynomial time.

Then verifier $\mathcal{V}$ has input w, and c can be an encoding of the transitions of $\mathcal{M}$ when $\mathcal{M}$ accepts w.

$\mathcal{V}$ accepts w if $\mathcal{M}$ accepts w

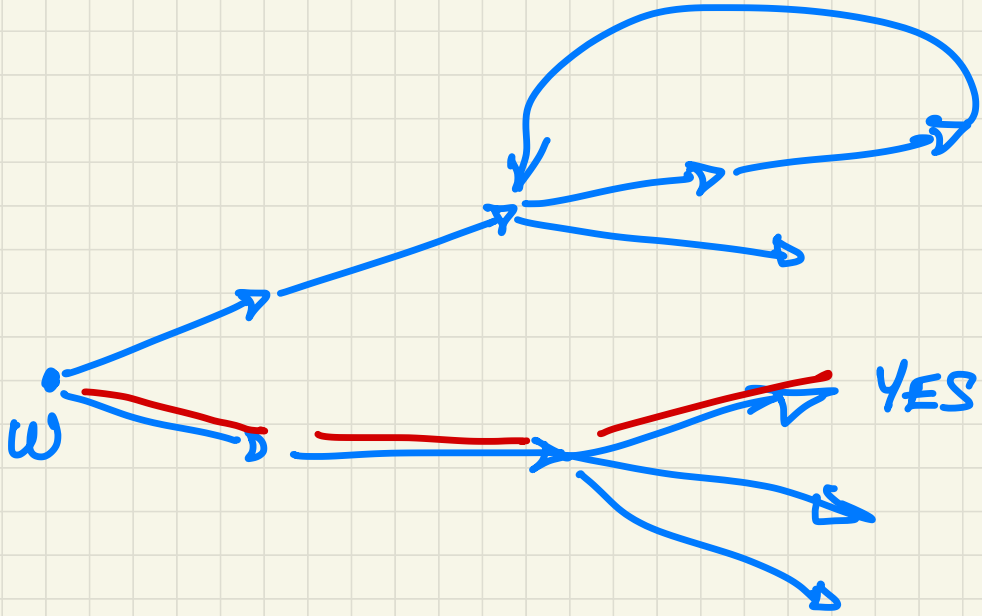
Second, assume there is a verifier $\mathcal{V}$ for L

Given w, a machine $\mathcal{M}$:

- picks a random c which size is polynomially bounded in $|w|$
- simulates $\mathcal{V}$ with input $< w, c >$

$\mathcal{M}$ accepts w if $\mathcal{V}$ accepts $< w, c >$

As $\mathcal{V}$ computes in polynomial time, $\mathcal{M}$ computes in polynomial time


w
YES

Polynomial reduction

Question : $NP \subseteq P$.
If so : $P = NP$
1 M$ question, see Millennium problem at Clay Mathematics Institute
Prove that $\exists L \in NP$ with $L \notin P$? Not until today.
Instead : L NP-complete ?

Definition 13 (Polynomial reduction)

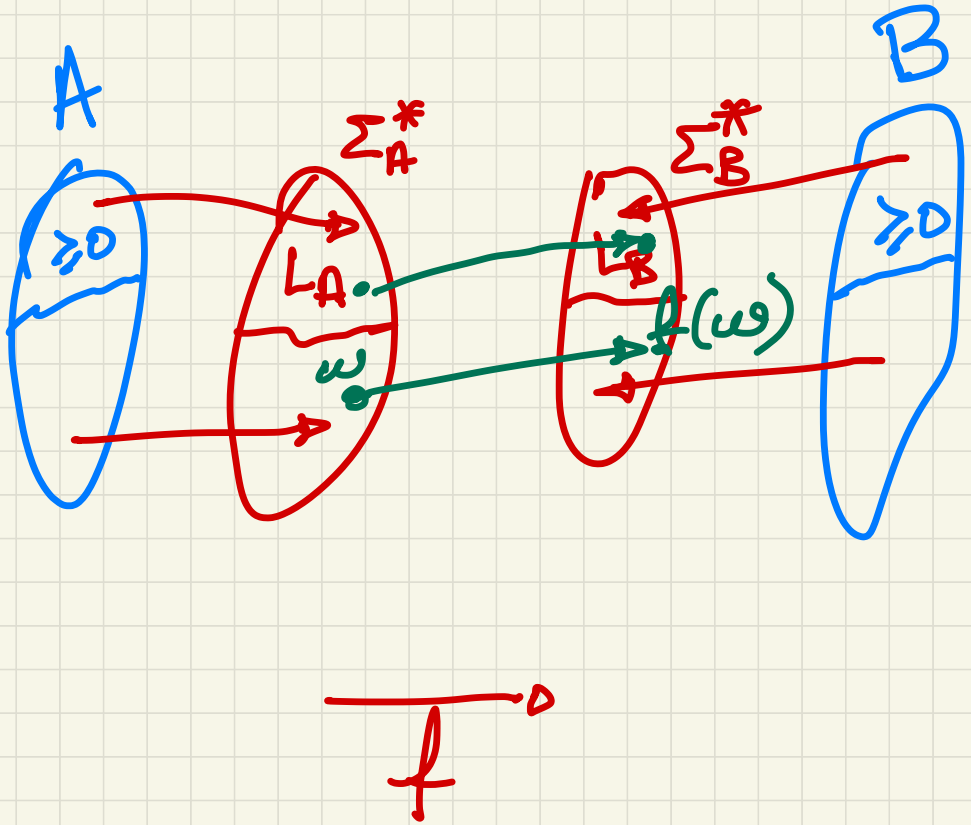
Let A and B be problems, with respective languages L_A and L_B on alphabets Σ_A and Σ_B. A polynomial reduction (transformation) from A to B is a polynomial-time computable function $f : \Sigma_A^* \to \Sigma_B^*$ such that :

$$w \in L_A \iff f(w) \in L_B;$$

This is denoted $A \leq_P B$.

$\leq_P$: transitive and reflexive

f: comp. by a T.M.
$f(x) = M(x)$


A
B
Σ_A^*
Σ_B^*
≥ 0
≥ 0
L_A
L_B
ω
$f(\omega)$
f

$\leq_P$ is transitive

Proof:

assume $L_1 \leq_P L_2$ and $L_2 \leq_P L_3$, then

$$\exists \text{ a DTM } \mathcal{M}, \forall x \in \Sigma_1^*, \mathcal{M}(x) \in L_2 \iff x \in L_1$$

with a polynomial p, s.t. $t_{\mathcal{M}}(x) \leq p(|x|)$.

$$\exists \text{ a DTM } \mathcal{M}', \forall x \in \Sigma_2^*, \mathcal{M}'(x) \in L_3 \iff x \in L_2$$

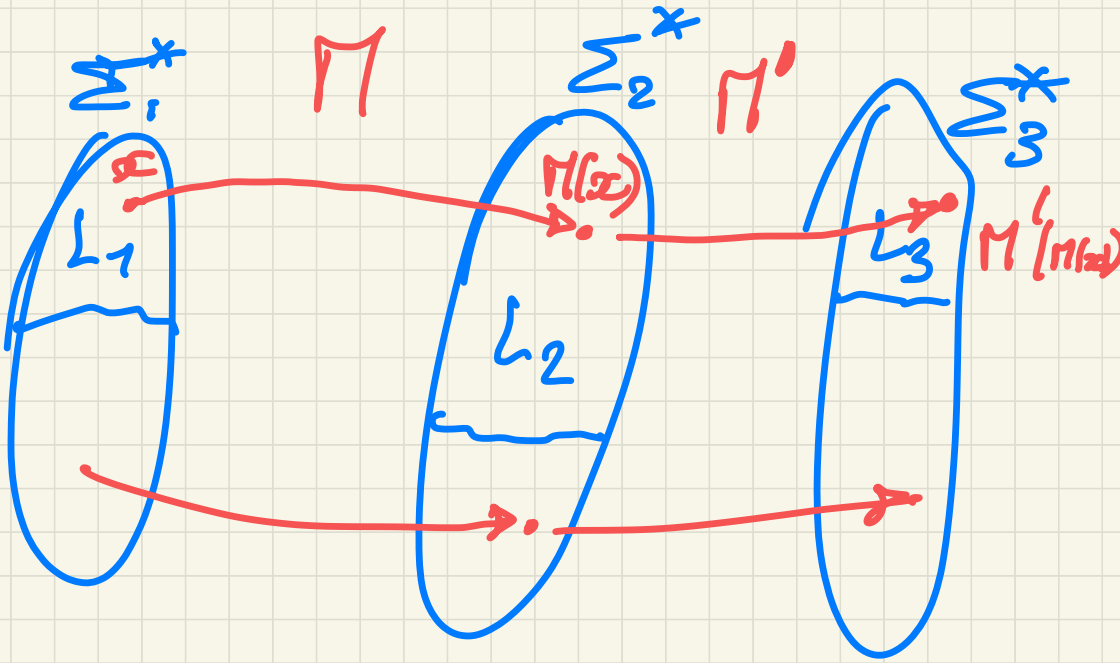
with a polynomial p', s.t. $t_{\mathcal{M}'}(x) \leq p'(|x|)$

Let $x \in \Sigma_1^*$,

$$\mathcal{M}'(\mathcal{M}(x)) \in L_3 \iff \mathcal{M}(x) \in L_2 \iff x \in L_1$$

and $t_{\mathcal{M}'}(\mathcal{M}(x)) \leq p'(|\mathcal{M}(x)|) \leq p'(p(|x|)) = p''(|x|)$

using $|\mathcal{M}(x)| \leq t_{\mathcal{M}}(x) \leq p(|x|)$ and p' is increasing



p' increasing

$n \leq n' \Rightarrow p'(n) \leq p'(n')$

$n \in \mathbb{N}$

$n' \in \mathbb{N}$

if $x \leq_P y$ and $y \in P$ then $x \in P$

Proposition 5.1

If $A \leq_P B$ and $B \in P$, then $A \in P$.

$M''(x) = \text{YES} \iff x \notin L_A$

Proof:

if $A \leq_P B$ then:

$$\exists \text{ a DTM } \mathcal{M}, \forall x \in \Sigma_A^*, \mathcal{M}(x) \in L_B \iff x \in L_A$$

and a polynomial p, s.t. $t_{\mathcal{M}}(x) \leq p(|x|)$
Furthermore, if $B \in P$ then

$$\exists \text{ a DTM } \mathcal{M}', \forall z \in \Sigma_B^*, \mathcal{M}'(z) = \text{YES} \iff z \in L_B$$

and polynom p', s.t. $t_{\mathcal{M}'}(z) \leq p'(|z|)$.
Let $x \in L_A$. Then $\mathcal{M}(x) \in L_B$. Converse is also true, and

$M''(x)$

$$|\mathcal{M}(x)| \leq t_{\mathcal{M}}(x) \leq p(|x|) \tag{2}$$

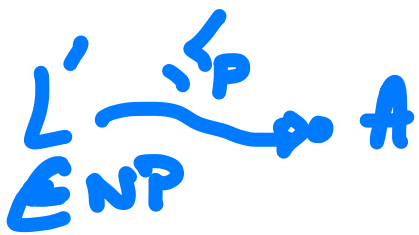
$\mathcal{M}'(\mathcal{M}(x)) = \text{YES} \iff \mathcal{M}(x) \in L_B \iff x \in L_A$. and
$t_{\mathcal{M}'}(\mathcal{M}(x)) \leq p'(|\mathcal{M}(x)|) \leq p'(p(|x|)) = p''(|x|)$.

Definition 14

A problem A is NP-hard if for every problem $B \in NP$, $B \leq_P A$ (every B is reducible to A). If additionally $A \in NP$ then A is NP-complete.

Proposition 5.2

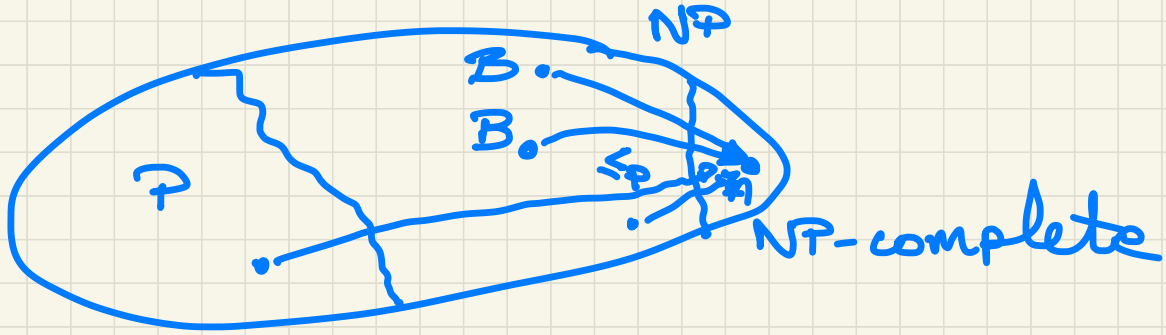
If A is NP-hard and $A \leq_P B$ then B is NP-hard



Proof:

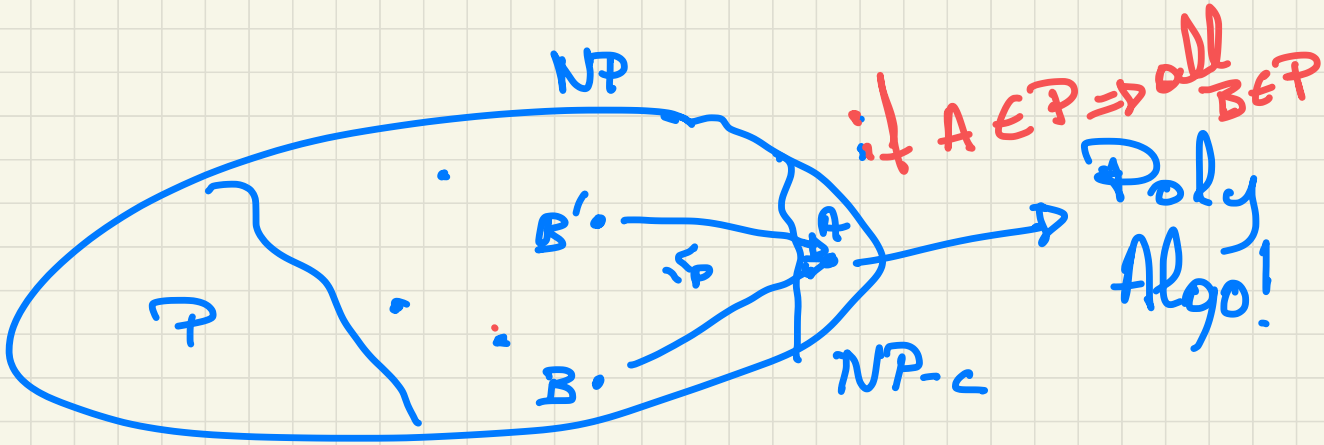
If A is NP-hard then $\forall L' \in NP,\ L' \leq_P A$

By transitivity of $\leq_P$, $A \leq_P B$ implies $\forall L' \in NP,\ L' \leq_P B$



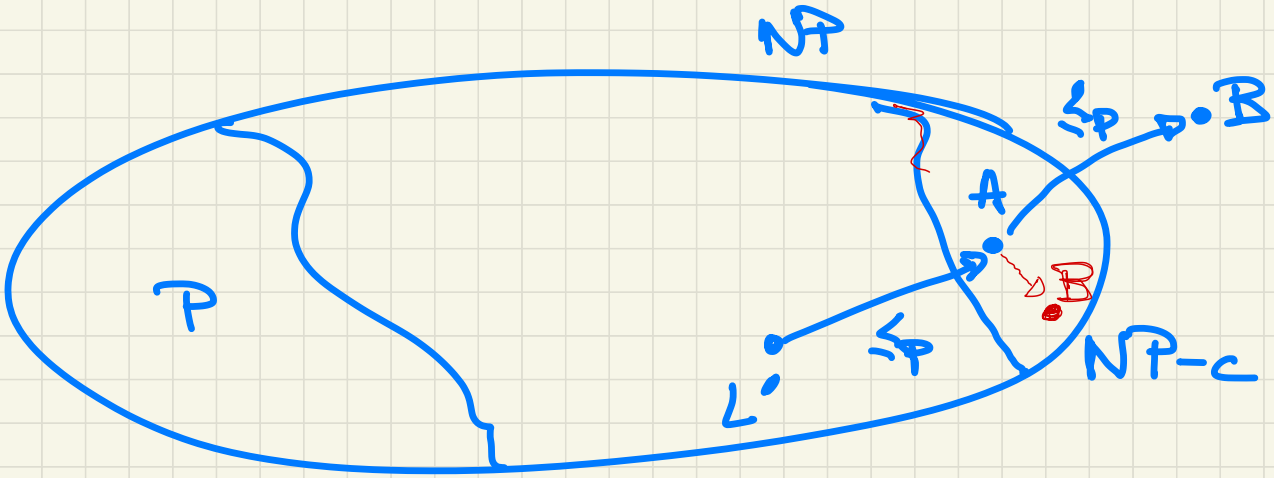
NP-complete

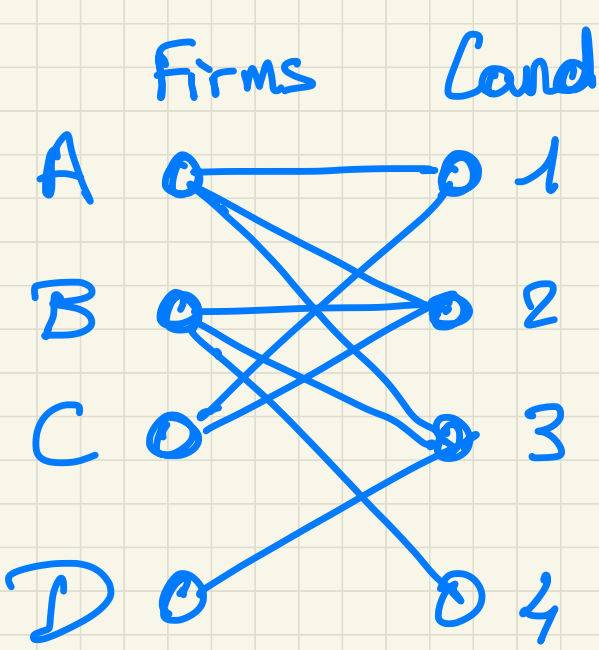
- NP-hard
- $\in$ NP



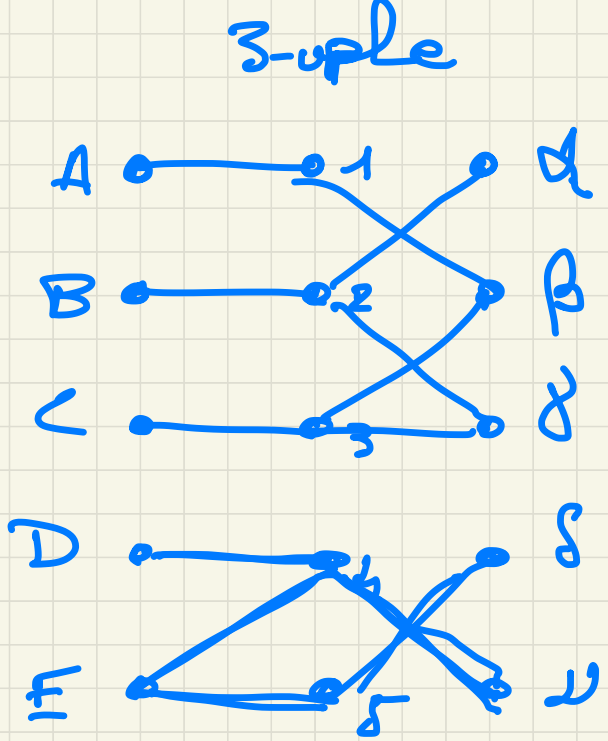
If $A \in P \Rightarrow$ all $B \in P$

$P \neq NP$?


NP
P
A
B
B
L
NP-C

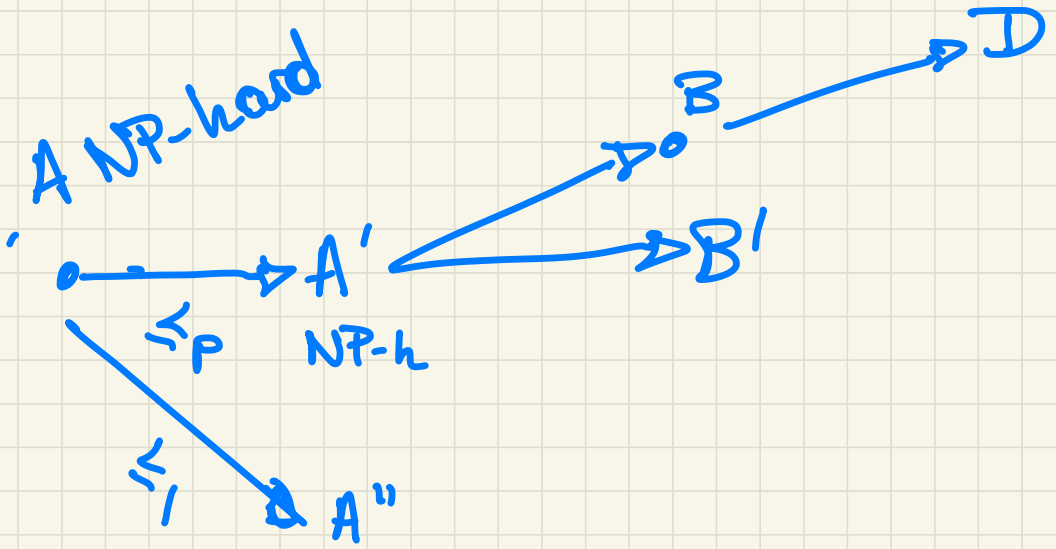


Matching



3D-Matching

$\in$ NP-c



Karp

$\#L \in NP,\ L \leq_p B$ (NPh)

HC $\leq_P$ TSP
Proof :
Standard formulation for Decision Problems

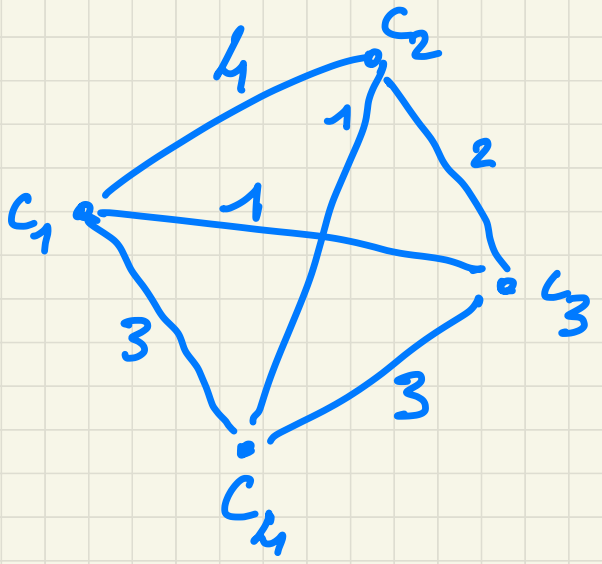
TRAVELING SALESMAN PROBLEM (TSP)
INSTANCE : a finite set $C = \{c_1, c_2, \ldots, c_m\}$ of cities, distances $d(c_i, c_j) \in \mathbb{N}$ for every $(c_i, c_j) \in C \times C$, a bound $B \in \mathbb{N}$.
QUESTION : is there a tour of all the cities of C having a length of at most B, that is, a permutation π de $\{1, \ldots, m\}$ s.t.

$$\sum_{i=1}^{m-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(m)}, c_{\pi(1)}) \leq B$$

C

+

$d(c_i, c_j)$



$B = 7$

Positive mutation : Tour $c_1 c_3 c_2 c_4$

$\pi(1) = 1 \quad \pi(2) = 3$

$\pi(3) = 2 \quad \pi(4) = 4$

Example

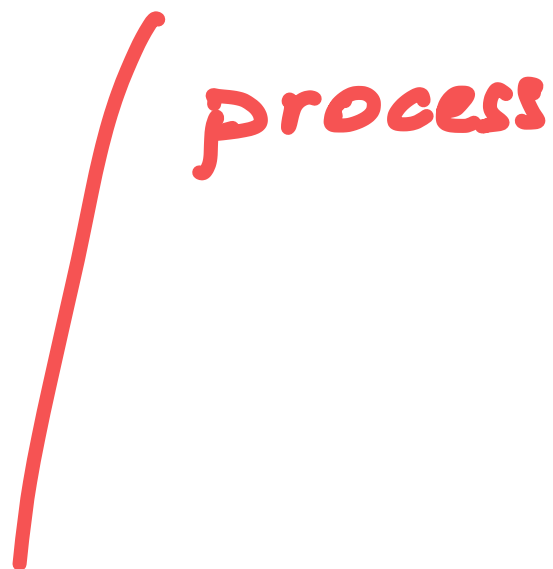
HAMILTONIAN CIRCUIT (HC)
INSTANCE : a graph $G = (V, E)$ with $|V| = n$ (vertices are denoted v_i)
QUESTION : is there an hamiltonian circuit in G, that is, a permutation π of $\{1, \ldots, n\}$ s.t.
$\forall i \in \{1, \ldots, n-1\}, (v_{\pi(i)}, v_{\pi(i+1)}) \in E$ and $(v_{\pi(n)}, v_{\pi(1)}) \in E$

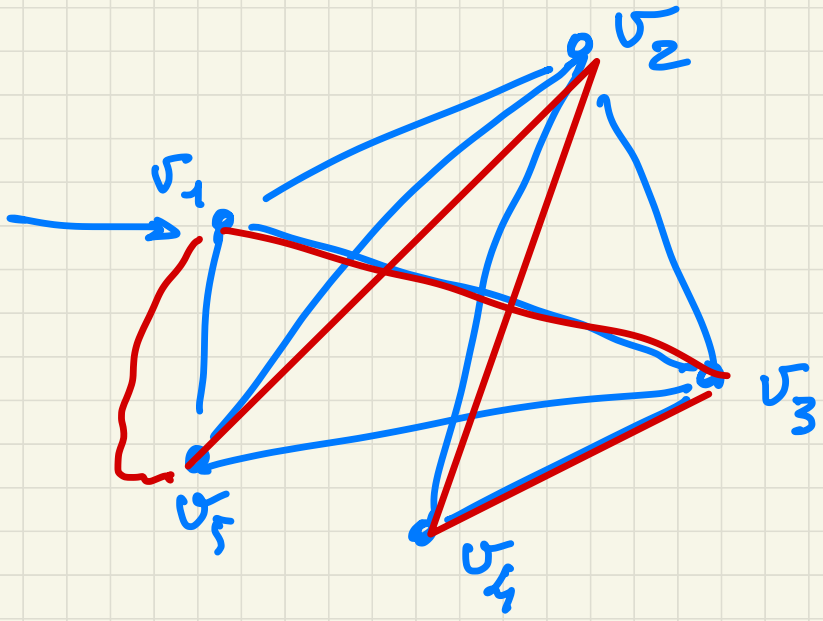
Methodology: **for any instance** of **HC** define ONE peculiar instance of **TSP**

Set $C = V$
and $\forall v_i, v_j \in V \times V$,

$$d(v_i, v_j) = \begin{cases} 1 & \text{si } (v_i, v_j) \in E \\ 2 & \text{sinon} \end{cases}$$

Finally $B = |V|$.





π?

$v_1\ v_3\ v_4\ v_2\ v_5$

$\pi(1) = 1$

$\pi(2) = 3$

$\pi(3) = 4$

$\pi(4) = 2$

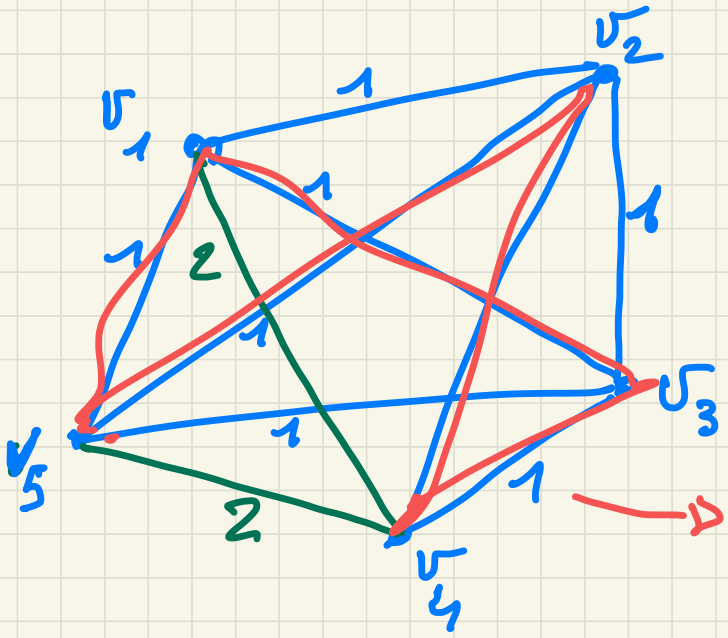
$\pi(5) = 5$

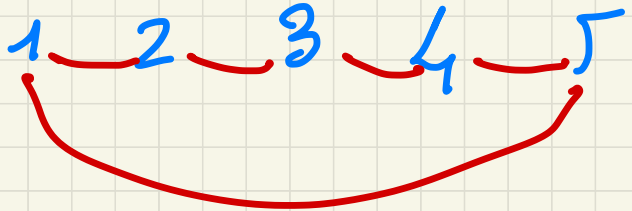
Positive instance of HC

$C = V$

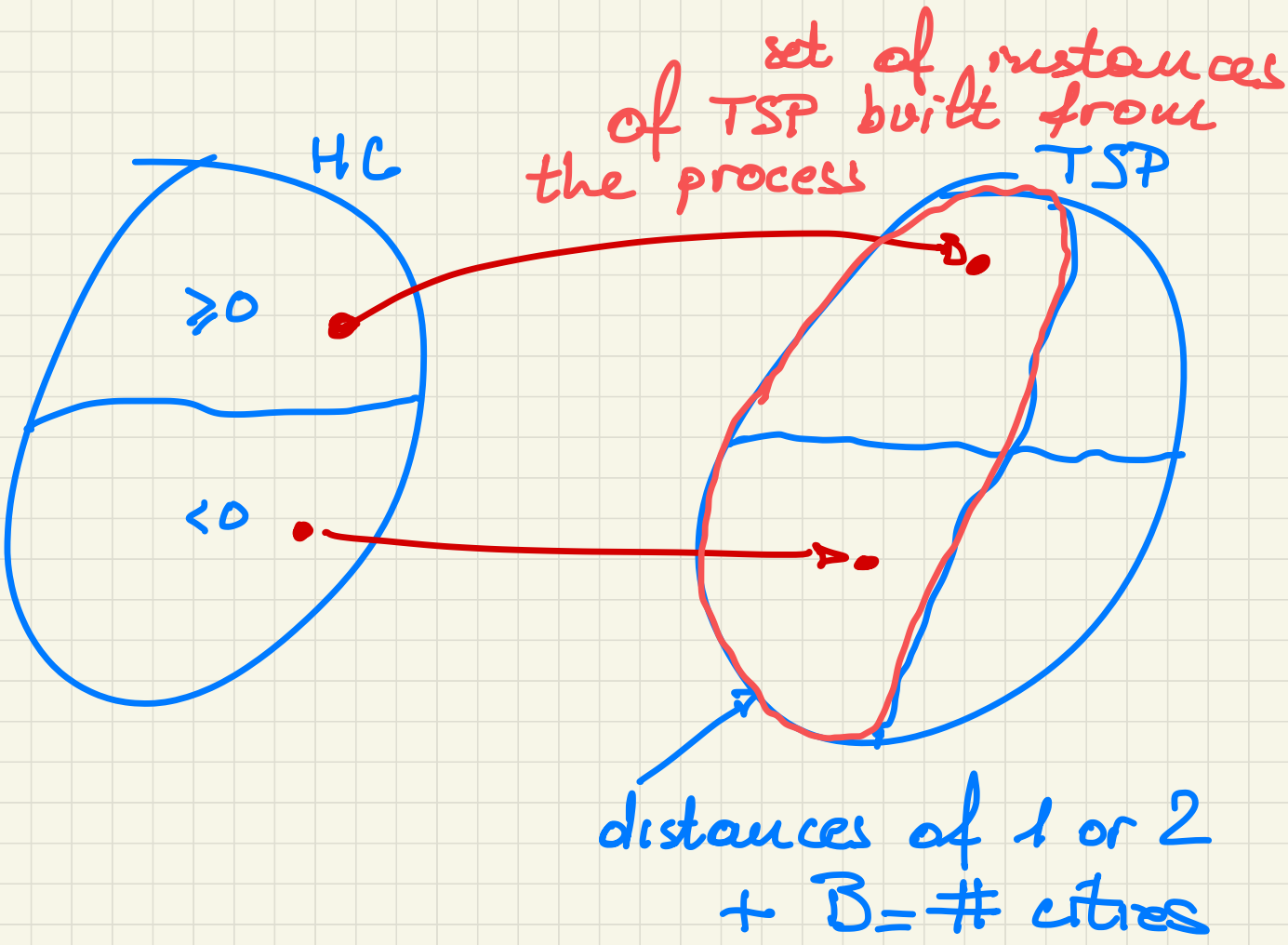
$d(.,.)$

$B = 5$


v_1
v_2
v_3
v_4
v_5
1
1
1
2
1
1
2
1
existing edges of original graph


1
2
3
4
5

graph

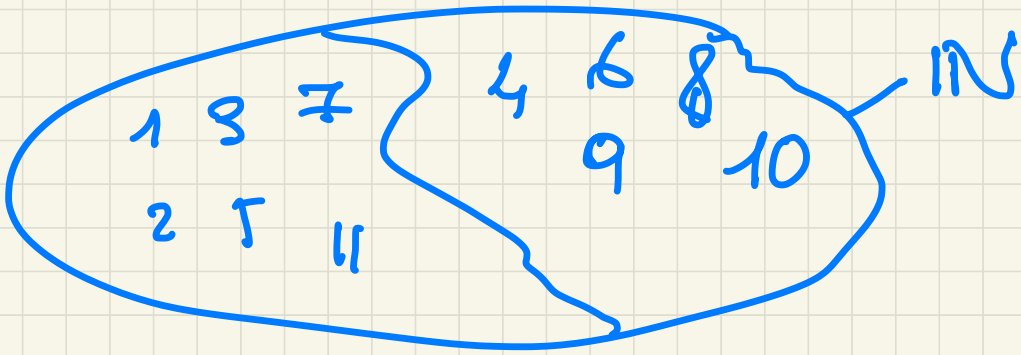

HC
≥0
<0
TSP
set of instances of TSP built from the process
distances of 1 or 2
+ B = # cities

Problem PRIME

PRIME

INSTANCE : n an integer

QUESTION : is n a prime number?



Step 1: Check that the construction is polynomial.
Here: $O(|V|^2)$
Step 2: Recall

Definition 15 (Polynomial reduction)

Let A and B be problems, with respective languages L_A and L_B on alphabets Σ_A and Σ_B. A polynomial reduction (transformation) from A to B is a polynomial-time computable function $f : \Sigma_A^* \to \Sigma_B^*$ such that :

$$w \in L_A \iff f(w) \in L_B$$

$$I \in HC_{?,0} \iff f(I) \in TSP_{?,0}$$

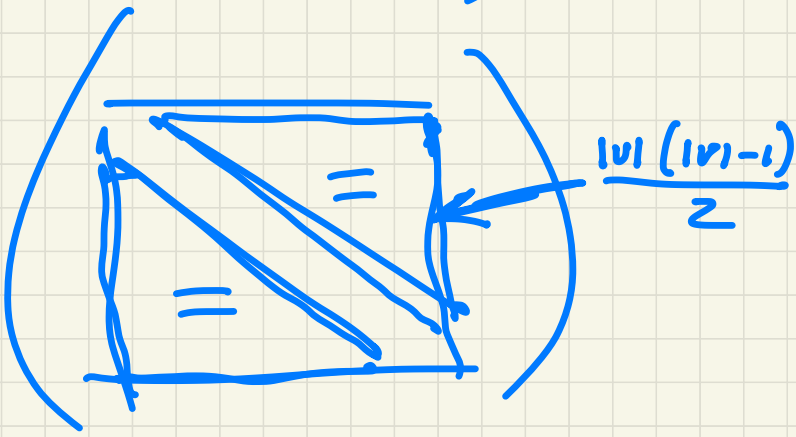
This is denoted $A \leq_P B$.

Polynomial construction

- $C = V$ — $O(|V|)$
- $d(c_i, c_j)$ $\forall (c_i, c_j) \in V \times V$ — $O(|V|^2)$
- $B = |V|$ — $O(|V|)$

$O(|V|^2)$

polynomial



Example

Step 2.1:

Assume the instance of HC is positive

That is, there is a permutation π of the vertices of G which defines an hamiltonian circuit.

Let us take the same permutation π on THE **TSP** instance.

Circuit = only existing edges of G

Distance is $|V| = B$. = sum of $|V|$ distances with value 1

TSP instance is positive

Step 2.2:

Assume the instance of TSP is positive

The there is a permutation π de $\{1, \ldots, |V|\}$ such that

$$\sum_{i=1}^{|V|-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(|V|)}, c_{\pi(1)}) \leq B = |V|$$

by contradiction

Prove it uses only distances of 1.

⇒ correspond to existing edges

We denote the preceding equation: $\sum_{j=1}^{|V|} d_j \leq |V|$

Assume there is a distance different from 1

Thus, it exists $j_0 \in \{1, \ldots, |V|\}$ s.t. $d_{j_0} = 2$ and

$\sum_{j=1}^{|V|} d_j = d_{j_0} + \sum_{j=1, j\neq j_0}^{|V|} d_j \leq |V|$, thus $\sum_{j=1, j\neq j_0}^{|V|} d_j \leq |V| - 2$

d_{j_0}

But $\forall j, j \neq j_0 \; d_j \geq 1$ implies $\sum_{j=1, j\neq j_0}^{|V|} d_j \geq |V| - 1$

A contradiction

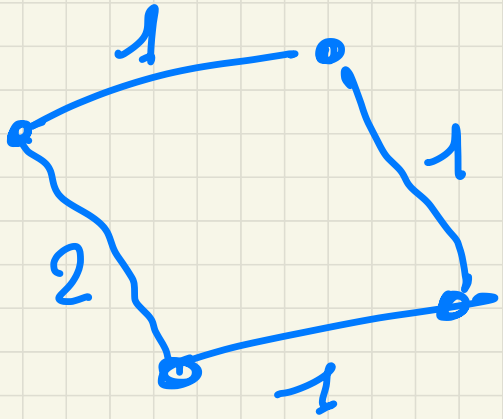
Thus, the tour uses only distances of 1, which correspond to existing edges of G

This defines a hamiltonian circuit in G — the one it was built from

HC instance is positive

$$|V| - 1 \leq \sum_{\substack{j=1 \\ j \neq j_0}}^{|V|} d_j \leq |V| - 2$$

Contradiction



Sum ≤ 4

Example

Steps 1, 2.1, 2.2 $\Longrightarrow$ polynomial transformation from **HC** to **TSP**

as **HC** is NP-complete, thus NP-hard, then **TSP is NP-hard**

Step 3: is **TSP** in NP ?

certificate ? a permutation

answer to the QUESTION

complexity of checking phase ?

$O(|C|)$, that is, polynomial

Provided a certificate C

- check it is a permutation of C

1 3 2 4 5 $v_1\ v_3\ v_2\ v_4\ v_5$

$O(|C|)$

- check $\sum$ distances $\leq B$

$O(|C|)$

finally $O(|C|)$

Methodology

Assuming A is NP-complete

Proving NP-completeness of B by $A \leq_P B$

Step 1 Prove that the construction (of instance of B) is polynomial

Step 2 Prove

- **Step 2.1** Positive instance of $A \implies$ Positive instance of B
- **Step 2.2** Positive instance of $B \implies$ Positive instance of A

Now, as A is NP-hard, then B is NP-hard (Proposition 6.2)

Step 3 Prove B is in NP

Another Example: CLIQUE

3-SAT $\leq_p$ CLIQUE

CLIQUE
INSTANCE : a graph $G = (V, E)$ and an integer $J \leq |V|$
QUESTION : is there a subset $V' \subseteq V$ with $|V'| \geq J$ and such that $\forall u \in V', v \in V', (u, v) \in E$

V' must be complet graph (of size at least J)

From **3-SAT** (NP-complete)
3-SAT
INSTANCE : a set C of k clauses (denoted $C_j, j \in \{1, \ldots, k\}$) each of which is a disjunction of 3 literals (denoted C_{j1}, C_{j2}, C_{j3}), a set U of boolean variables denoted x_i. Each literal can be either a boolean variable x_i or its negation $\bar{x}_i$ (also written $\neg x_i$).
QUESTION : is there a truth assignment for U, that is a function $f : U \mapsto \{True, False\}$ such that the k clauses of C are True.

C_1 C_2

$\phi = (x_1 \vee x_2 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3)$

C_{11} C_{12}

$U = \{x_1, x_2, x_3\}$

truth assignment

$f(x_1) =$ False

$f(x_2) =$ True

$f(x_3) =$ False

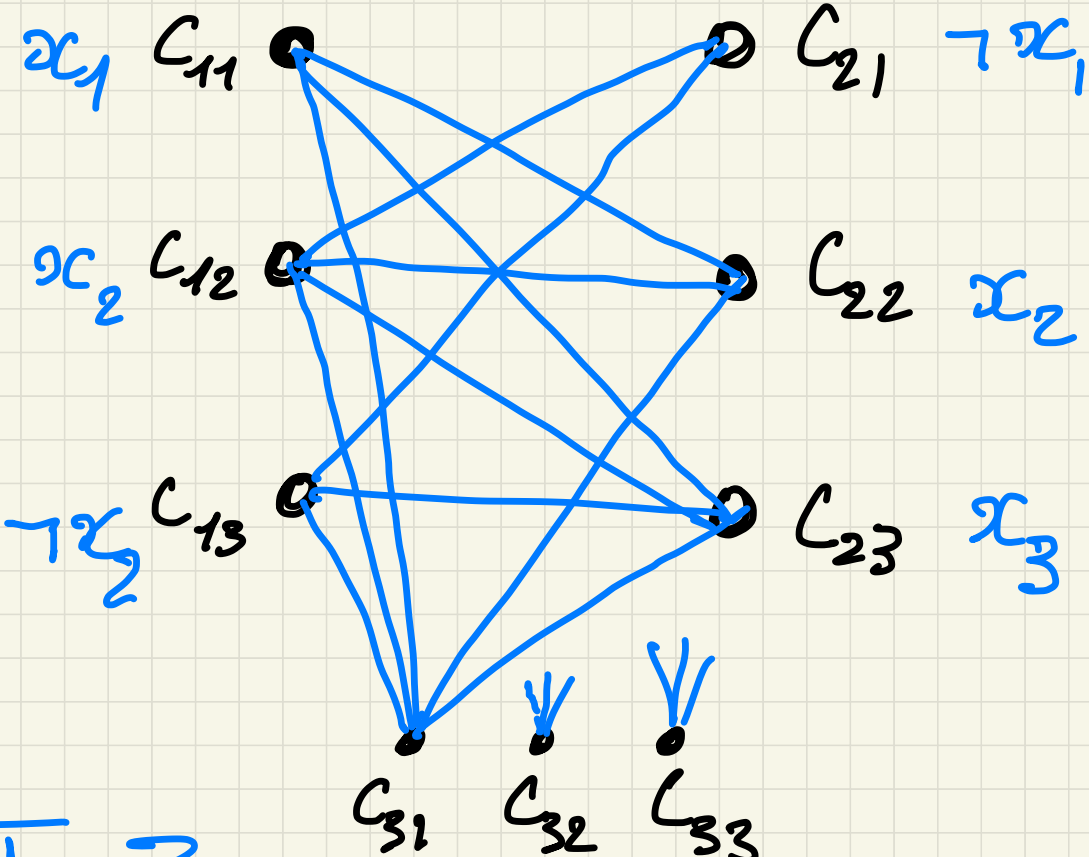
ϕ positive instance of 3-SAT

C_1 C_2

$\phi = (x_1 \vee x_2 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3)$

$\wedge (x_1 \vee \neg x_2 \vee \neg x_3)$

C_3



$J = 3$

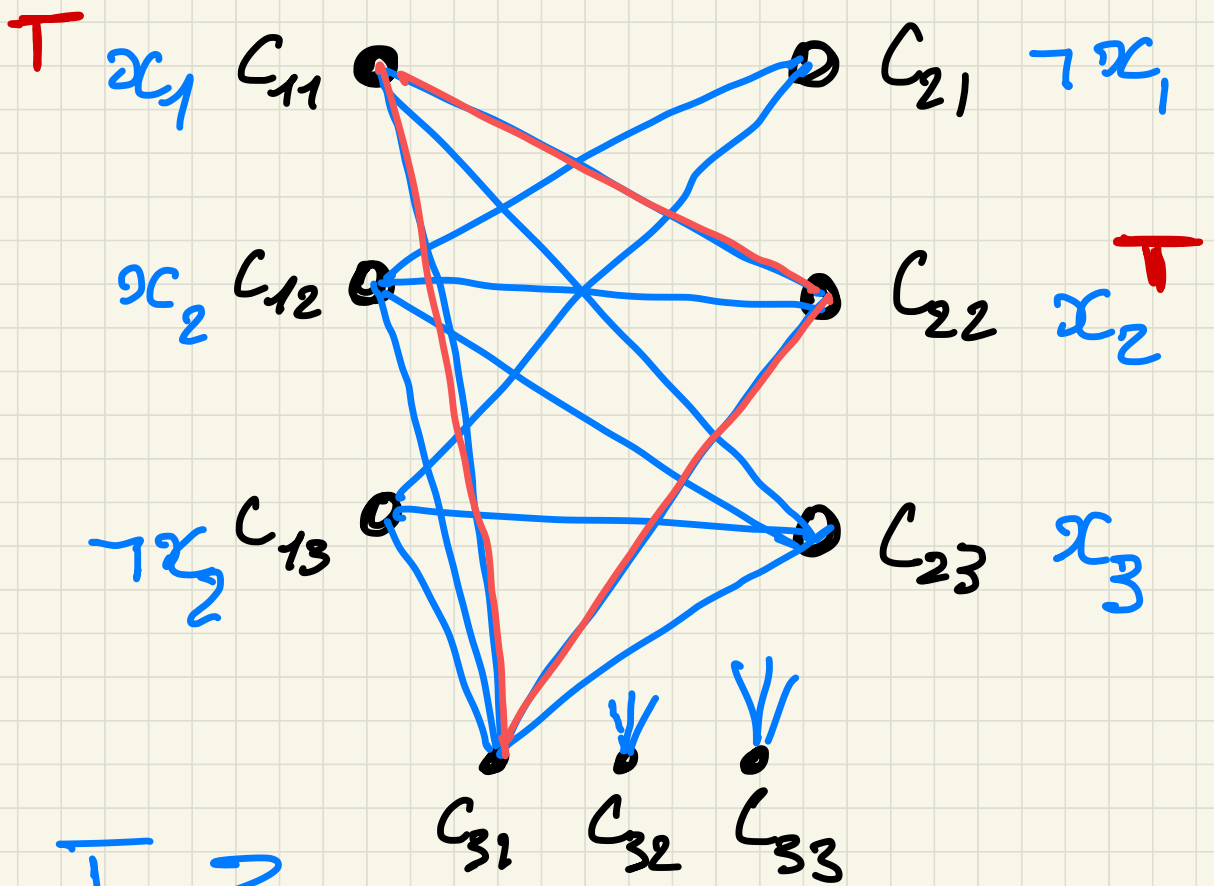
x_1 $\neg x_2$ $\neg x_3$

C_1 C_2

$\phi = (x_1 \vee x_2 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3)$

T T T

C_3



$J = 3$

T x_1 $\neg x_2$ $\neg x_3$

8 clauses

$x_1^T \quad x_2^T \quad x_3^F$

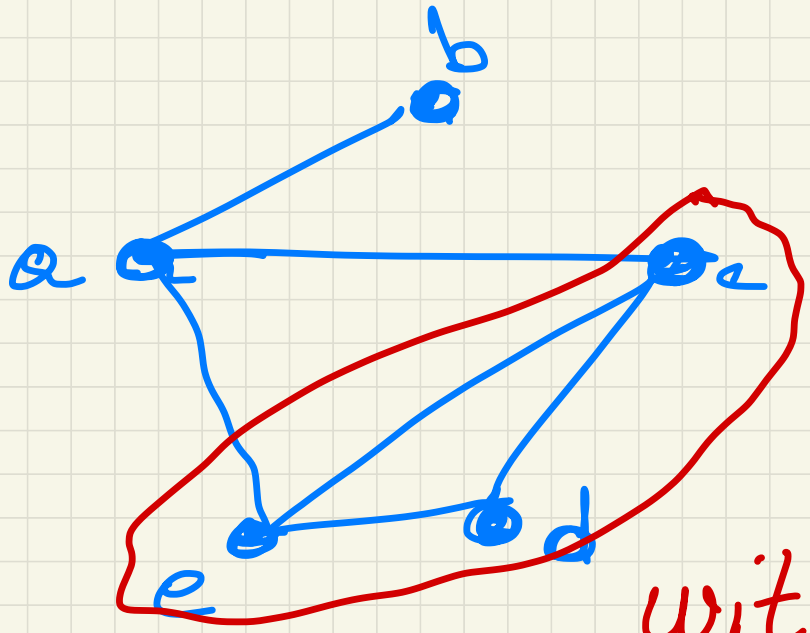
$x_1 \quad x_2 \quad \neg x_3$

$x_1 \quad \neg x_2 \quad x_3$

$\vdots$

$\neg x_1^F \quad \neg x_2^F \quad x_3^F$

$\neg x_1 \quad \neg x_2 \quad \neg x_3$



with $J = 3$ YES!

$J = 4$

~~Positive~~ or not

$|V'| \geq 4$ s.t.

$\forall u \in V', v \in V'$

$(u, v) \in E$

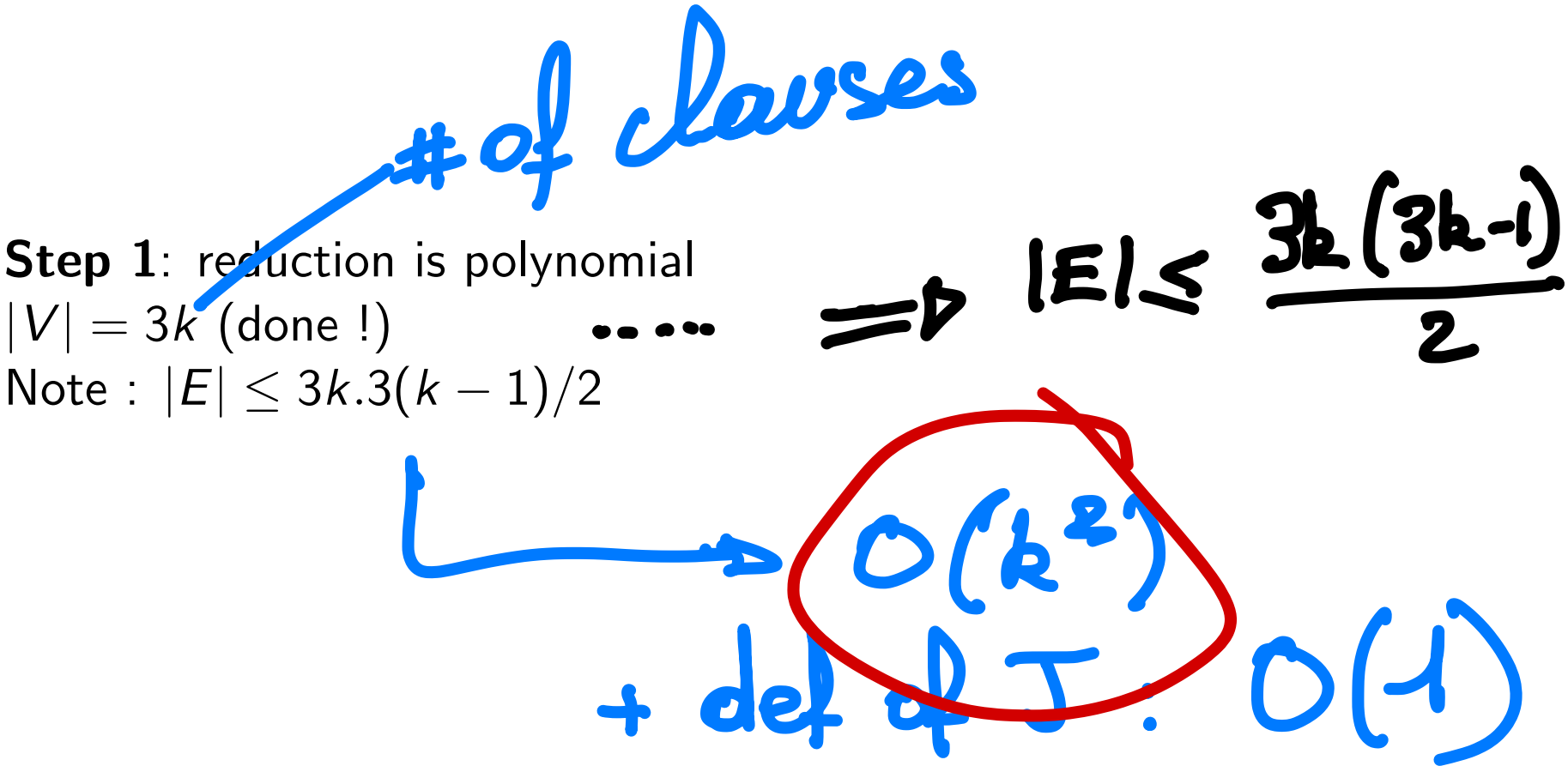
CLIQUE

Redution:

CLIQUE is fully defined by $G = (V, E)$ **and** J

- definition of V: if literal C_{jl} is x_i (resp. $\neg x_i$), add a vertex labelled with the literal x_i (resp. $\neg x_i$) in V
- definition of E: add an edge from a vertex corresponding to literal C_{jl} to another vertex corresponding to literal C_{hg} if
  - $h \neq j$. No edge between any two vertices corresponding to literals from a same clause
  - literals C_{jl} and C_{hg} are not negations of one another (e.g. if C_{jl} is x_i and C_{hg} is $\neg x_i$ then no edge)
- definition of J: $J = k$

CLIQUE


#of clauses
Step 1: reduction is polynomial
$|V| = 3k$ (done !)
Note : $|E| \leq 3k.3(k-1)/2$
$\Rightarrow |E| \leq \frac{3k(3k-1)}{2}$
$O(k^2)$
+ def of I : $O(1)$

Step 2: **3SAT** $\geq 0 \iff$ **CLIQUE** ≥ 0

Step 2.1: Assume instance of **3SAT** is positive
Then, there is a truth assignment which makes all clauses True
Thus, at least ONE literal per clause has a value True
V' = Pick one vertex per group of 3 corresponding to literals with value True
Let us show that V' is a CLIQUE of size J:

- clearly $|V'| = J$, as we picked exactly one literal per clause
- there is an edge between any two of them, as the corresponding literals can not be negation one from another (all of them has a value True).

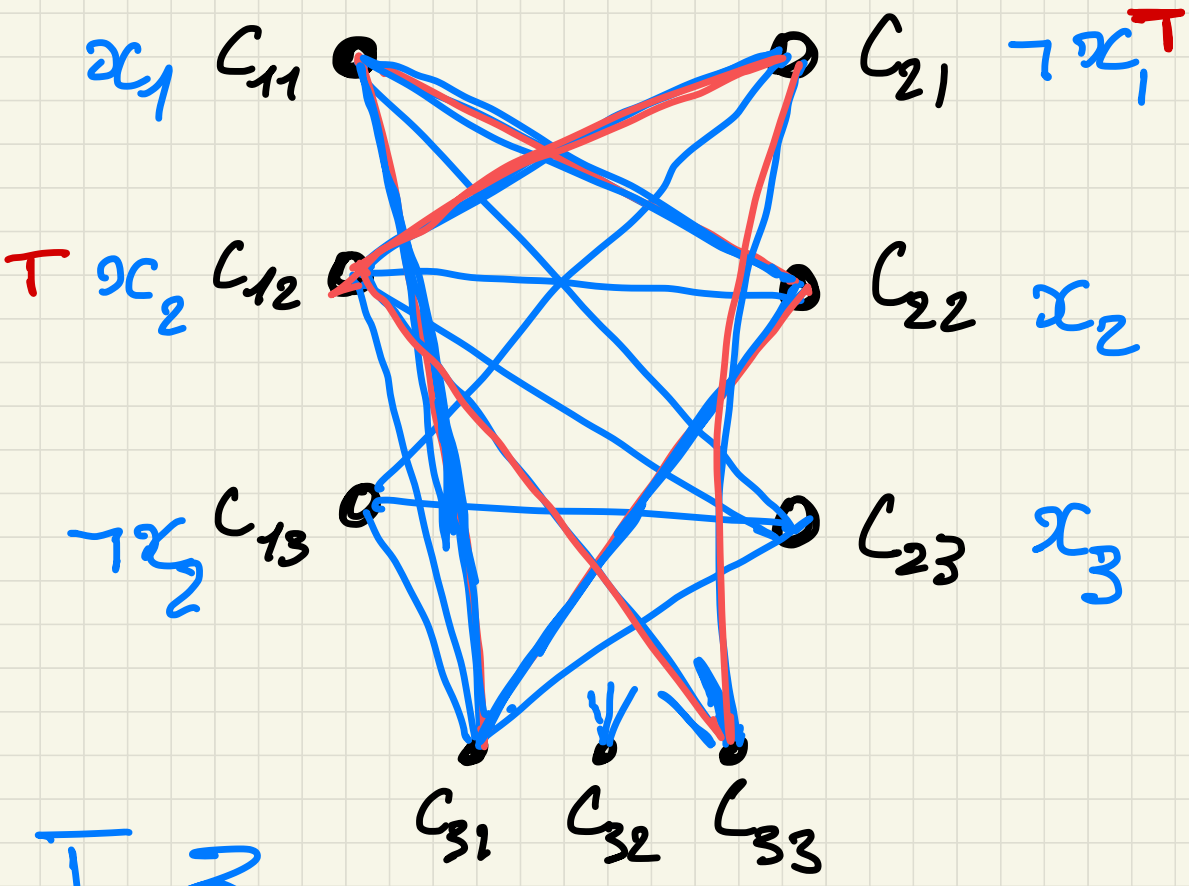
+ vertices of V' : picked into + groups (of 3)

C_1 C_2

$\phi = (x_1 \vee x_2 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3)$

T T T

C_3



$J = 3$

x_1 $\neg x_2$ $\neg x_3$

T

Step 2.2: Assume THE instance (the one we built) of **CLIQUE** is positive
There is $V' \subseteq V$ with $|V'| \geq J$ s.t. $\forall u \in V', v \in V', (u, v) \in E$
Set corresponding literals to a value True
Remark: a CLIQUE (in this graph) has **at most** $J = k$ vertices, every vertex picked inside a group of 3 (corresponding to one clause) (proof by contradiction: assume $k + 1$ vertices, pigeon-hole principle...)
Thus $|V'| = J$
Let us show this defines a truth assignment s.t. all clauses are True

- can not assign value True to x_i and $\neg x_i$ at the same time: no edge between corresponding vertices $\Rightarrow$ truth assignment
- as **exactly** one vertex per group is in V', the corresponding literal being True, every clause is True

QED

CLIQUE

From steps 1 and 2: **3SAT** $\leq_P$ **CLIQUE**
As **3SAT** is NP-complete (thus NP-hard), **CLIQUE** is NP-hard

Step 3: Show **CLIQUE** $\in$ **NP**
certificate $= V' \subseteq V$

- check $|V'| \geq J$: $O(|V|)$
- check $\forall u \in V', v \in V', (u, v) \in E$: $O(|V|^2)$

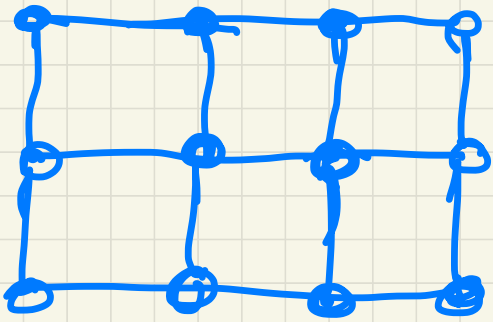
$O(|V'|)$

$\frac{|V|(|V|-1)}{2}$

CLIQUE $\in$ **NP**

From steps 1, 2 and 3: **CLIQUE** is **NP-complete**

Assuming we consider grid graphs



No clique of size > 2

INDEPENDENT SET

Incidentally

INDEPENDENT SET

INSTANCE : a graph $G = (V, E)$ and an integer $J \leq |V|$

QUESTION : is there a subset $V' \subseteq V$ with $|V'| \geq J$ and such that $\forall u \in V', v \in V', (u, v) \notin E$

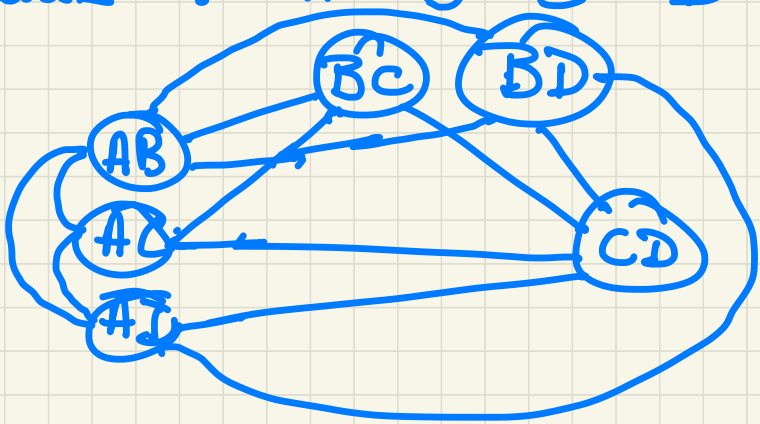
Application: Scheduling...

Reduction:

- $V' \subseteq V$ **CLIQUE** in $G = (V, E)$ if and only if V' **INDEPENDENT SET** (same size) in $G^C = (V, E^C)$ where $E^C = \{(u, v) | u \in V, v \in V, (u, v) \notin E\}$ + same J
- from **3-SAT**

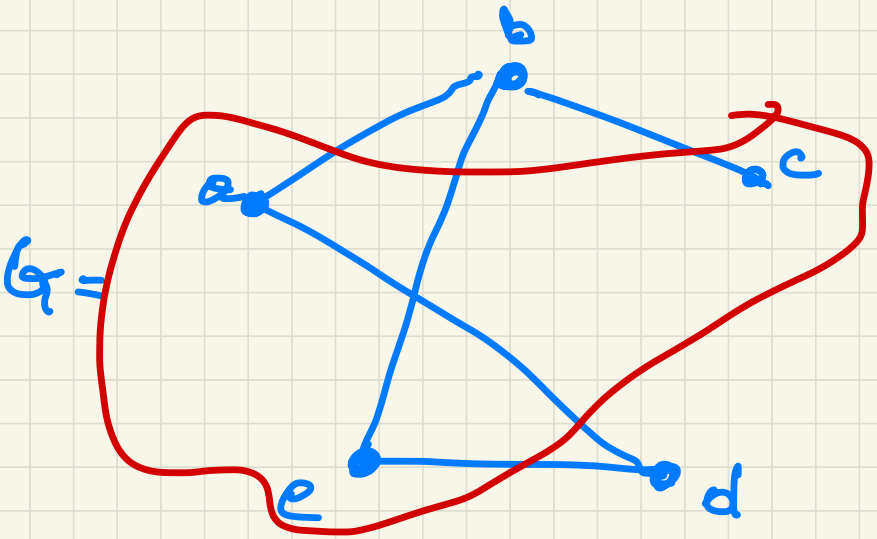
Applications

Teams : A B C D



Largest Independent Set

= max # of matchs at the same time



$J = 4$

$J = 3$

positive

$\leq_p$ $\quad w \in L_A \iff f(w) \in L_B$

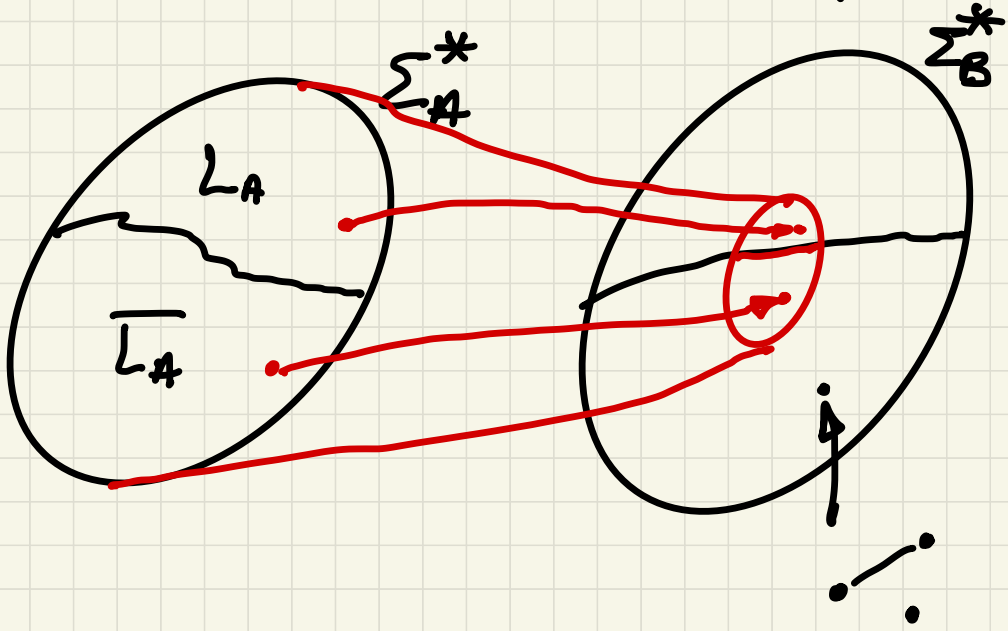
point 2 not f a bijection

f by assumption

a bijection

- $f(w)$ unique
- PROVIDED one $f(w)$

then w is unique



INDEPENDENT SET

From **CLIQUE**

By definition of G^C, we have

$\forall u \in V, v, \in V, \big((u,v) \in E \Leftrightarrow (u,v) \notin E^C\big)$

and consequently $\forall u \in V, v, \in V, \big((u,v) \notin E \Leftrightarrow (u,v) \in E^C\big)$

Then, if an instance of **INDEPENDENT SET** is positive

$\exists V' \subseteq V, |V'| \geq J$ s.t. $\forall u \in V', v \in V', (u,v) \notin E^C$ and thus $(u,v) \in E$ V' is also a "clique" of size at least J and instance of **CLIQUE** is positive

Converse proof from **CLIQUE** to **INDEPENDENT SET** uses the same argument (to do)

polynom. $O(|E|)$ reduction

$(a,b) \notin E^C \Rightarrow (a,b) \in E$

Step 1

CLIQUE $\leq_p$ I.S.

build G^c

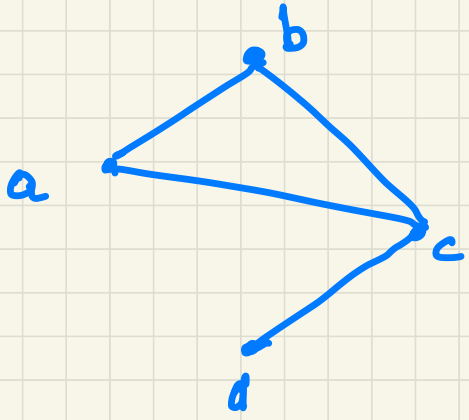
$O(|E|) \leq O(|V|^2)$

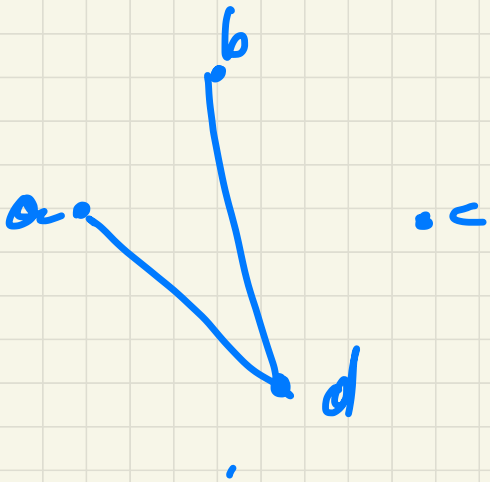
G

	a	b	c	d
a	0	1	1	0
b	1	0	1	0
c	1	1	0	1
d	0	0	1	0

G^c

0	0	0	1
0	0	0	1
0	0	0	0
1	1	0	0





CLIQUE

$G = (V, E)$

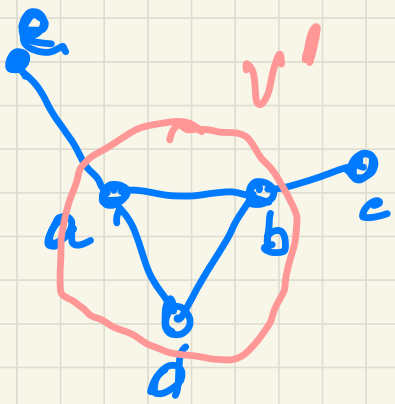
J

=

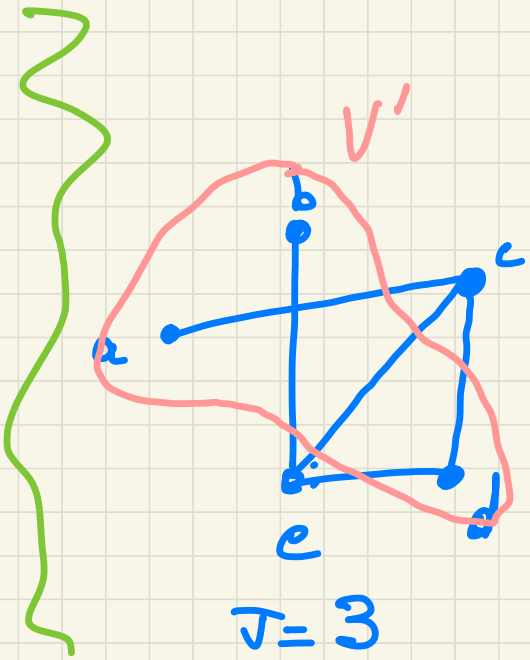
INDEPENDENT SET

$G^c = (V, E^c)$

J

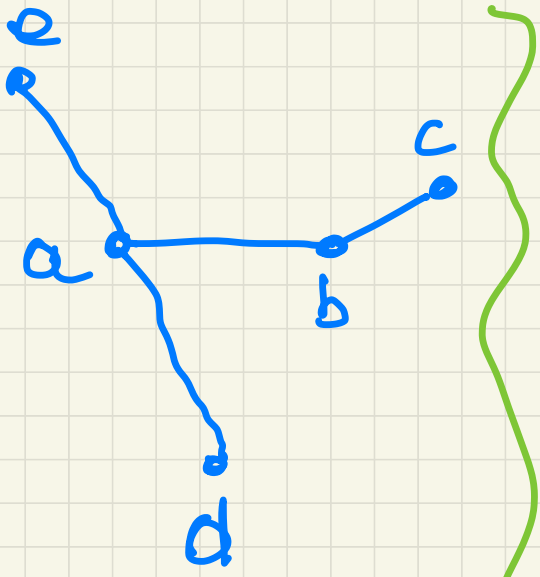


$J = 3$



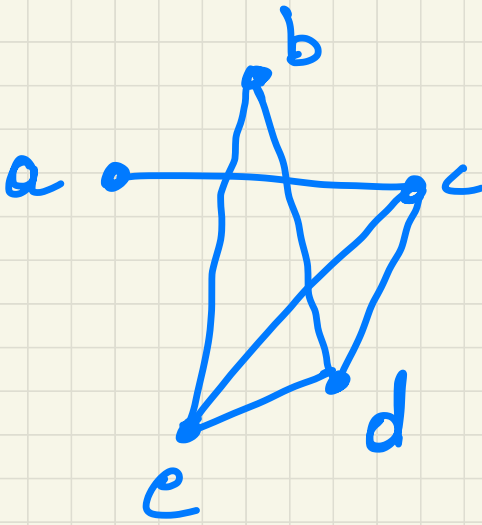
$J = 3$

$(V, E \cup E^c)$ is the complete graph of size $|V|$!



J = 3

Negative instance of CLIQUE



J = 3

negative instance of INDEPENDENT SET

Step 3

I.S. $\in$ NP

certificate ? V'

- $|V'| \geq J$ $\quad O(|V|)$
- $\forall u \in V', v \in V'$
  $(u,v) \notin E^c$ $\quad O(|E|) = O(|V|^2)$

$O(|V|^2)$

Yes, I.S. $\in$ NP

INDEPENDENT SET

ToDo for 11/19

From **3-SAT**

Given an instance of **3-SAT** (C a formula of k clauses), instance of **INDEPENDENT SET** ($G = (V, E)$ and J) is defined by:

1. V is composed of:
   - 1.1 three vertices labelled with c_{j1}, c_{j2}, c_{j3} corresponding to c_{j1}, c_{j2}, c_{j3} for every clause C_j;
2. E is composed of:
   - 2.1 three edges (c_{j1}, c_{j2}),(c_{j1}, c_{j3}),(c_{j2}, c_{j3}) for every triplet c_{j1}, c_{j2}, c_{j3} ;
   - 2.2 an edge between vertex c_{ji} and vertex c_{kl} if $j \neq k$ and literals c_{ji} and c_{kl} **are** negation of one another ;
3. $J = k$

Example: assume

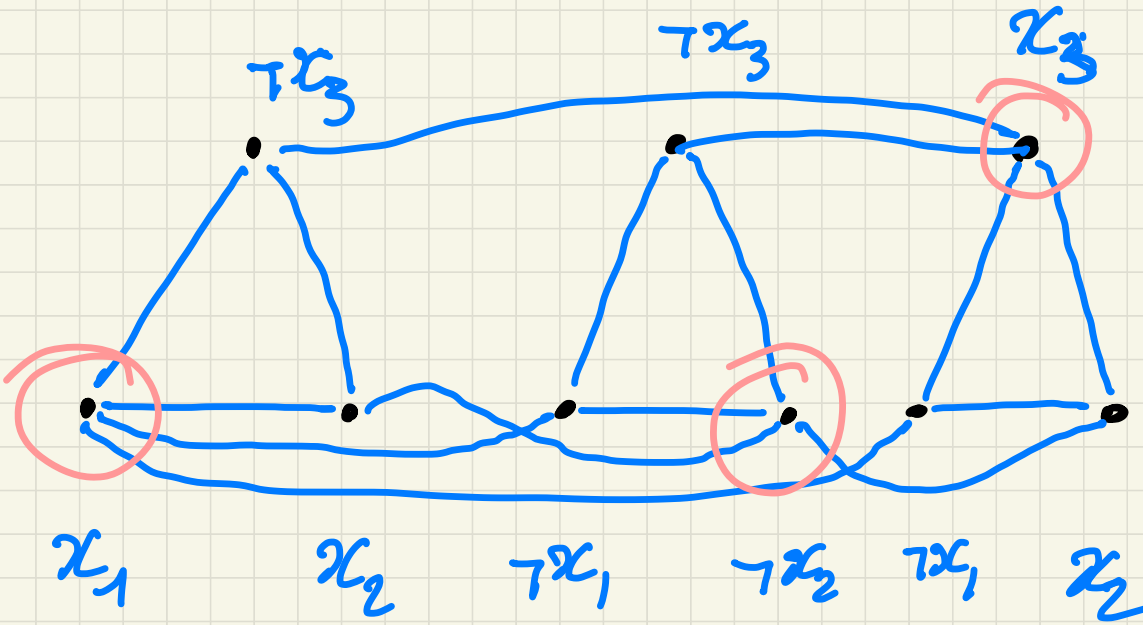
$C = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$

Proof: To do !

Example: assume

$C = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$

Proof: To do !



$J = 3$

- Step 1 (Polynomial)

$|V| = 3k$ done !

$|E|$?

$|E| \geq 3k$

YES: Construction

- Remarks

- $c_{ji} \in V' \Rightarrow c_{jh} \notin V'$ for $h \neq i$

- $c_{ji} \in V' \Rightarrow$ all c_{hu} s.t. $(c_{ji}, c_{hu}) \in E$ not in V'

Remark

I.S. search $V' \subseteq V$

$|V'| \geq J = k$

In the case of graph we built:

$k(J)$ is the maximal size for an Independent Set

by contradiction:

Assume we have

an I.S. V' of size at least $k+1$ then clearly V' can contain one vertex per triangle $\Rightarrow$ k vertices then the $k+1^{th}$ must be a vertex of one Δ where one vertex is already in $V' \Rightarrow$ not I.S

then $J = k$ is the maximal size of a I.S :

and $|V'| \leq J$ on this graph

$|V'| \geq J$ (I.S. Pb)

$\Rightarrow |V'| = J$

Step 2

2.1. Assume THE instance of I.S. is positive

Then $\exists V' \subseteq V$, $|V| \geq J$

s.t. $\forall u \neq v \in V'$, $(u,v) \notin E$

Indeed we must have

$|V'| = J$

from V' build one truth assignment making all clauses True

We set all literals corresponding to vertices in "V" to True

- is it truth assignment?

$f : U \to \{True, False\}$?

YES: $x_i \in U$ can not be set to T and F at the same time.

It could be if a vertex {corresponding labeled with x_i} is a element of V' and a vertex labeled with $\neg x_i$ too.

Not possible because they share an edge

- Are all clauses True?

As noticed, V' contains exactly

one vertex per triangle thus at least one literal is True in each clause

2.2. Assuming the instance of 3-SAT is positive then there is a truth assignment which makes

all clauses True

Build V' by:

- picking one True literal inside each clause
- put the corresponding vertex to V'

Claim: V' is an I.S. of size $\geq J$

Proof:

- About $|V'| \geq J$:
  indeed $|V'| = J$
  because we picked
  exactly 1 literal
  per clause $\Rightarrow k = J$
  vertices
  in V'

- $\forall u \in V', v \in V', (u,v) \notin E$
  - $\forall v \in V'$, $v \in V'$
    v and $\bar{v}$ can not
    lie inside a
    triangle

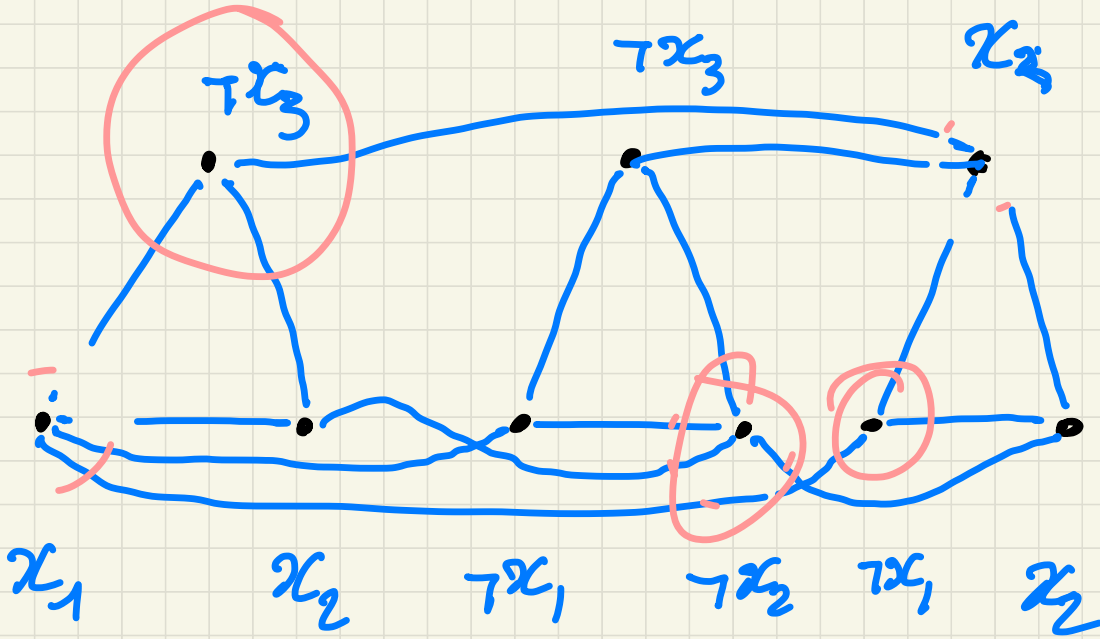
because we picked only 1 vertex per triangle

- as all literals we have chosed are set to True the corresponding vertices do not share an edge

Example: assume

$C = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$

Proof: To do !



$J = 3$

3. I.S. $\in$ NP

certificate ?

$V' \subseteq V$

! - check $|V'| \geq J$

$O(|V'|) = O(|V|)$

- $\forall u \in V', v \in V', (u,v) \notin E$

$O(|V'|^2) = O(|V|^2)$

$\binom{|V|}{2}$

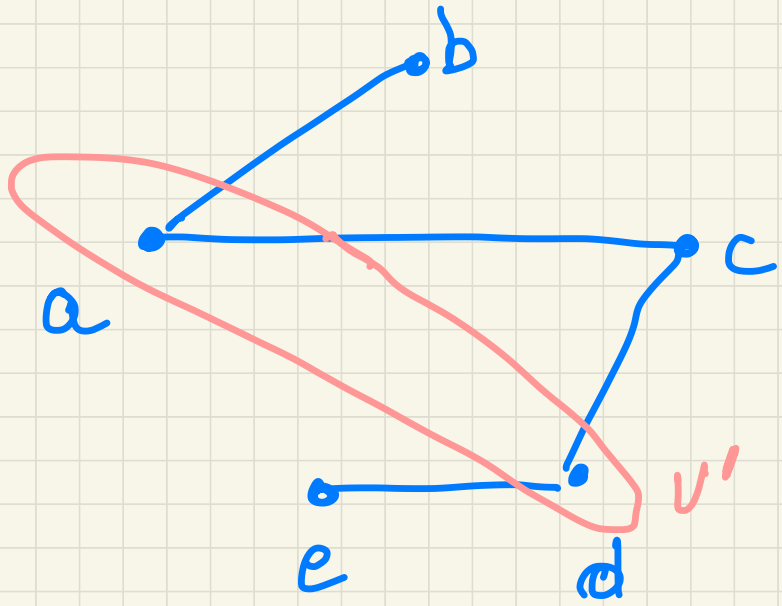
$O(|V|^2)$

(Cover by vertex)

VERTEX COVER
INSTANCE : a graph $G = (V, E)$ and an integer $J \leq |V|$
QUESTION : is there a subset $V' \subseteq V$ with $|V'| \leq J$ and such that $\forall (u, v) \in E$ we have $u \in V'$ or $v \in V'$

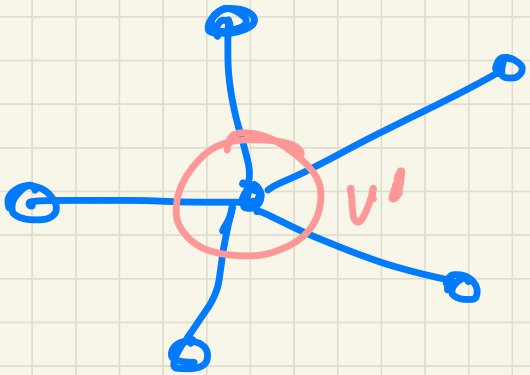
From...
3-SAT
INSTANCE : a set C of k clauses (denoted $C_j, j \in \{1, \ldots, k\}$) each of which is a disjunction of 3 literals (denoted c_{j1}, c_{j2}, c_{j3}), a set U of l boolean variables denoted x_i. Each literal can be either a boolean variable x_i or its negation $\neg x_i$ (sometimes written $\bar{x}_i$).
QUESTION : is there a truth assignment for U, that is a function $f : U \mapsto \{True, False\}$ such that the k clauses of C are True.

3SAT $\leq_P$ VC



$J = 2$

$J = 1$



Example: assume
$C = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$
Proof: To do !

VERTEX COVER

Given an instance of **3-SAT** (C the set of clauses), instance of **VERTEX COVER** ($G = (V, E)$ and J) is defined by:

1. V is composed of:
   - **1.1** two vertices x_i and $\neg x_i$ for all $x_i \in U$;
   - **1.2** three vertices labelled with c_{j1}, c_{j2}, c_{j3} corresponding to c_{j1}, c_{j2}, c_{j3} for every clause C_j;
2. E is composed of:
   - **2.1** an edge $(x_i, \neg x_i)$ for every couple $x_i, \neg x_i$;
   - **2.2** three edges (c_{j1}, c_{j2}),(c_{j1}, c_{j3}),(c_{j2}, c_{j3}) for every triplet c_{j1}, c_{j2}, c_{j3} ;
   - **2.3** an edge between vertex c_{ji} and vertex x_r (resp. $\neg x_r$) if literal C_{ji} is x_r (resp. $\neg x_r$) ;
3. $J = l + 2k$ (recall $l = |U|$).

l edges

3k

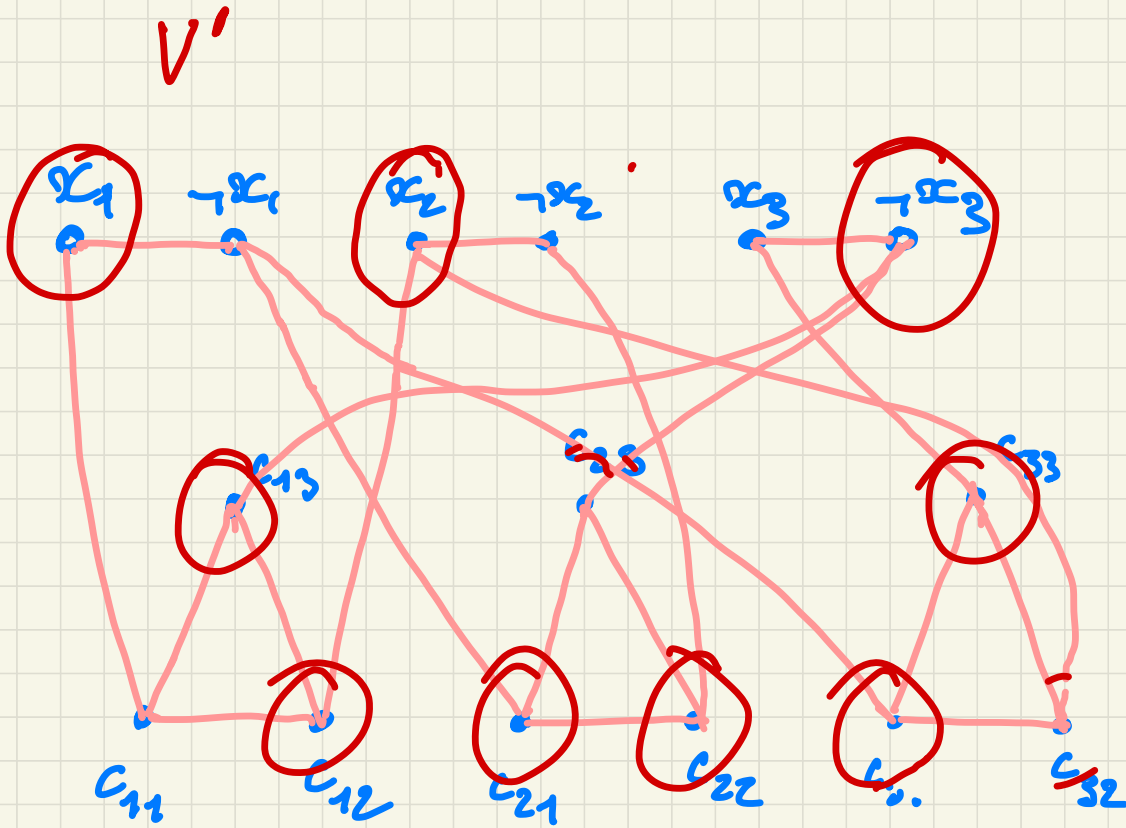
3k

Example: assume $C = (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee \neg x_4)$

Example: assume

$C = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$

Proof: To do !



$J = 3 + 2.3$ clauses

variables

$= 9$

VERTEX COVER

Step 1: reduction is polynomial

Recall $l = |U|$ and $k = |C|$

$|V| = 2l + 3k$ (done !)

Note : $|E| \leq (2l + 3k).(2l + 3k - 1)/2$

More precisely, here: $|E| = l + 3k + 3k$

Computation of J is also $O(l + k)$

$|E| = O(|V|^2)$

Step 2.1: Assume the instance of **3-SAT** is positive

VERTEX COVER

There is a truth assignment such that the k clauses of C are True. We define a Vertex Cover V' of G by choosing the following vertices:

1. if variable x_i (resp. $\neg x_i$) is True, then choose vertex x_i (resp. $\neg x_i$) of G ;
2. among the three vertices c_{j1}, c_{j2}, c_{j3}, choose two of them, s.t. the one **which is not chosen** shares an edge with a vertex chosen at the first step.

VERTEX COVER

Is it a Vertex Cover ?

1. every edge $(x_i, \neg x_i)$ has exactly one vertex in V' ;
2. every one of the edges (c_{j1}, c_{j2}),(c_{j1}, c_{j3}),(c_{j2}, c_{j3}) have at least [redacted] in V' by the second step of the picking procedure 2 vertices
3. concerning edges between c_{ji} and x_r, two possibilities:
   - 3.1 $c_{ji} \in V'$;
   - 3.2 if not, then it shares an edge with a vertex $x_r \in V'$

Finally, note that we have picked exactly $l + 2k$ vertices in V' (i.e. J vertices)

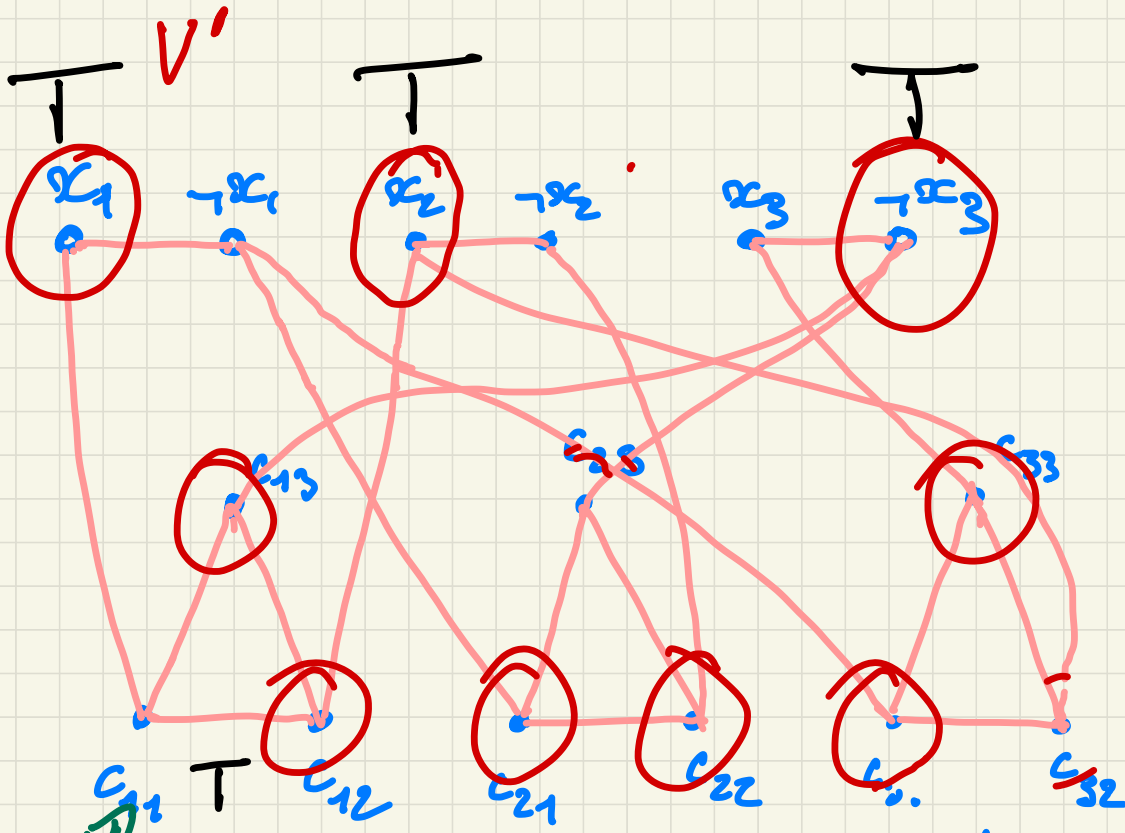
$\Rightarrow$ Positive instance of VC

Step 2.2: Assume the instance of **VERTEX COVER** is positive

Example: assume

$C = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$

Proof: To do !



$J = 3 + 2.3$ clauses

variables

$= 9$

VERTEX COVER

There is a set $V' \subseteq V$ with $|V'| \leq J = I + 2k$
First note that $I + 2k$ is the minimal size for a **VERTEX COVER**
thus $|V'| = I + 2k$ (structure ?)
If vertex $x_i \in V'$ (resp. $\neg x_i \in V'$) set variable x_i (resp. $\neg x_i$) to value True (OK because $x_i \in V' \Rightarrow \neg x_i \notin V'$ and converse)
Now for every triangle (c_{j1}, c_{j2}, c_{j3}), consider THE vertex (say c_{jh}) which is not in V'
Then consider edge (c_{jh}, x_i) (resp. $(c_{jh}, \neg x_i)$). As $c_{jh} \notin V'$ we must have $x_i \in V'$ (resp. $\neg x_i \in V'$)
As variable x_i (resp. $\neg x_i$) has then a value True, clause C_j is True
Instance of **3-SAT**

$J = l + 2k \Rightarrow V'$ must contain:

- 1 vertex per $(x_i, \neg x_i)$
- 2 vertices per triangle

Minimal size to cover all edges:

- at least one vertex per $(x_i, \neg x_i)$
  $\Rightarrow$ l vertices
- at least 2 vertices per triangle
  $\Rightarrow$ $2k$ vertices

! Assume we use 3 vertices to cover edges of one Δ

then remains:

$$2k-3$$

to cover $k-1$ Δ

although I need $2(k-1) = 2k-2$

VERTEX COVER

— Steps 1 & 2 ⇒ VC NP-hard

Step 3: **VERTEX COVER** is in NP ?
certificate ? V' !
checking that $\forall (u, v) \in E$ we have $u \in V'$ or $v \in V'$ is $O(|V|^2)$

$|E| \leq |V|^2$

— VC ∈ NP

⇒ VC NP-complete

DOMINATING SET

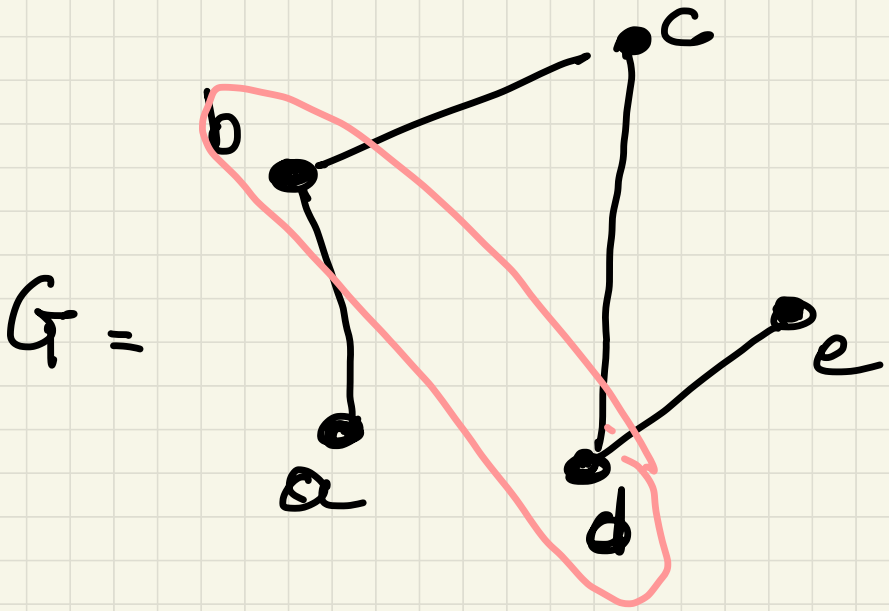
Incidentally

DOMINATING SET

INSTANCE : a graph $G = (V, E)$ and an integer $J \leq |V|$

QUESTION : is there a subset $V' \subseteq V$ with $|V'| \leq J$ and such that $\forall u \in V - V'$ ($u \in V$ but $u \notin V'$) there is $v \in V'$ with $(u, v) \in E$

Application: Broadcast in Mobile Ad'hoc NETwork, social networks...



$J = 2$ YES, positive

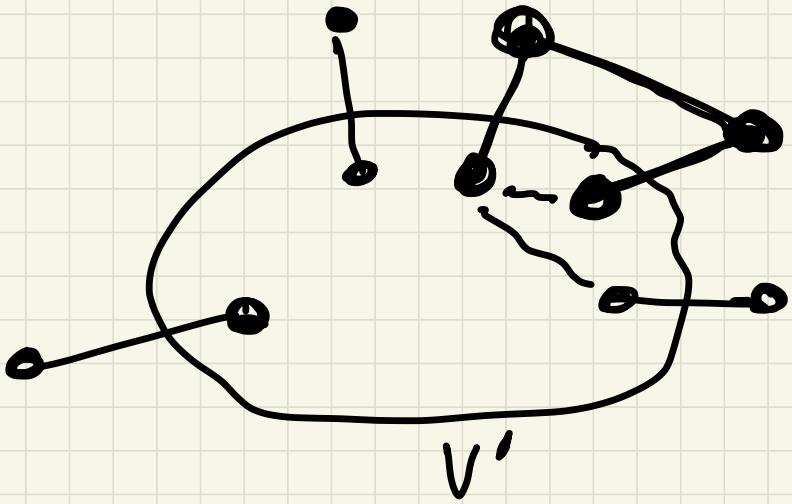
$J = 1$ NO

Choose $V' = \{b, d\}$

- $|V'| = 2 \leq$
- $a \notin V'$ but $(a, b) \in E$ and $b \in V'$

- $c \notin V'$ but $(c,b) \in E$ & $b \in V'$

- $e \notin V'$ but $(e,d) \in E$ & $d \in V'$



For dec. 3rd

GRAPH k-COLORABILITY
INSTANCE : a graph $G = (V, E)$ and an integer $k \leq |V|$
QUESTION : is G k-colorable, that is, does there exist a function $f : V \mapsto \{1, 2, \ldots, k\}$ s.t. $f(u) \neq f(v)$ whenever $(u, v) \in E$.

Example: Scheduling, Bandwidth allocation...
Reduction from **3-SAT**

GRAPH K-COLORABILITY

Rem : # U ≤ 3#C

read

NOT-ALL-EQUAL-(3)SAT (NAESAT)

INSTANCE : a set C of k clauses (denoted $C_j, j \in \{1, \ldots, k\}$) each of which is a disjunction of 3 literals (denoted c_{j1}, c_{j2}, c_{j3}), a set U of l boolean variables denoted x_i. Each literal can be either a boolean variable x_i or its negation $\neg x_i$ (sometimes written $\bar{x_i}$).

QUESTION : is there a truth assignment for U, that is a function $f : U \mapsto \{True, False\}$ such that the k clauses of C are True and every clause has at least one false literal

From **3-SAT**:

replace clause $C_j = c_{1j} \vee c_{2j} \vee c_{3j}$ by

3SAT ≤p NAESAT

$$(c_{1j} \vee c_{2j} \vee x_j') \wedge (\neg x_j' \vee c_{3j} \vee y_j') \wedge (x_j' \vee y_j' \vee \alpha)$$

(adding 2 variables x_j and y_j per initial clause C_j, plus one global variable α)

Example: assume

$C = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$

Proof: To do !

$f(x_1) = T$ F

$f(x_2) = F$ T

$f(x_3) = T$ F

$C_1 : (x_1 \vee x_2 \vee x_1') \wedge (\neg x_1' \vee \neg x_3 \vee y_1') \wedge (x_1' \vee y_1' \vee \alpha')$

$C_2 : (\neg x_1 \vee \neg x_2 \vee x_2') \wedge (\neg x_2' \vee \neg x_3 \vee y_2') \wedge (x_2' \vee y_2' \vee \alpha')$

C_3 : ?

3SAT		NAESAT
$\#U = l$	U	$\#U' = l + 2k + 1$
$\#C = k$	C	$\#C' = 3\#C = 3k$

$$\#V' = \underbrace{\ell}_{\substack{\text{init.} \\ \text{variables} \\ \text{from } V}} + \underbrace{2k}_{\substack{x'_j, y'_j \\ \forall j \in \{1,\dots,k\}}} + \underbrace{1}_{\alpha}$$

$$\#C' = 3k$$

1. Construction is polynomial

Step 2.1

read

2.1 - If instance of **3-SAT** is positive
set α to True and, for all 7 values of the c_{ij}, set x_j and y_j to ... =>

2.2 - If instance of **NAESAT** is positive

- if α has a value True
  then x_j or y_j must have value False
  - if x_j has value False
    then set c_{1j} or c_{2j} to True
  - if not, then y_j has value False then set c_{3j} to True
- if α has value False
  then negate all variables !

<=

$(c_{1j} \vee c_{2j} \vee x_j') \wedge (\neg x_j' \vee c_{3j} \vee y_j') \wedge (x_j' \vee y_j' \vee \alpha)$

F F T F T F T F T

c_{1j}	c_{2j}	c_{3j}	x_j'	y_j'
T	T	T	F	F
T	T	F	F	X
T	F	T	X	F
T	F	F	F	T
F	T	T	X	F
F	T	F	F	T
F	F	T	T	F

for any truth assignment on the instance of 3-SAT

(variables in U)
making all clauses True
we can build
a truth assignment
for the instance
of NAESAT s.t.
- all clauses are T
- and every clause
has at least
1 literal set
to F

Note: Proof starts with assumption $\alpha \leftarrow T$

in the end: Inst. of NAESAT is positive.

What if we start with assumption $\alpha \leftarrow F$?

Proof OK with $\alpha \leftarrow T$

if $\alpha \leftarrow F$, switch ALL boolean values for <u>all</u> variables (incl. α)

Step 2.2

$$(c_{1j} \vee c_{2j} \vee x_j') \wedge (\neg x_j' \vee c_{3j} \vee y_j') \wedge (x_j' \vee y_j' \vee \alpha)$$

T X F T X T F T T

X T

- $\alpha \leftarrow T$

$\Rightarrow x_j' \leftarrow F$

set c_{1j} or c_{2j} to True $\Rightarrow$ positive instance of 3-SAT

OR $x_j' \leftarrow T$

$(c_{1j} \vee c_{2j} \vee x_j') \wedge (\neg x_j' \vee c_{3j} \vee y_j') \wedge (x_j' \vee y_j' \vee \alpha)$

X T F F T F T F T

T X

$\alpha \leftarrow T$

negating of all var.

or $x_j' \leftarrow T$ then $c_{3j} \leftarrow T$

$\alpha \leftarrow F$

We use: if a truth assignment satisfies all

the constructs then negating all the values also works.

POSITIVE INST of NAESAT

$(\ell_{1,1}$ $\ell_{1,2})$ $\wedge$ $(\ell_{2,1}$ $\ell_{2,2})$ $\wedge$ $(\ell_{3,1}$ $\ell_{3,2})$

T F F T T F F T T F F T

$f(c_{1,1}) = T$ take $f(c_{1,1})$ $= \neg f(c_{1,1})$

Step 1

Step 2.1 + 2.2

3-SAT $\leq_p$ NAESAT

Step 3:

NAESAT $\in$ NP?

certificate?

a clauses

YES: certificate

a truth assignment

then check in

every clause at least

one T literal and

one f literal

Example: assume

$C = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$

T F F T F T

Proof: To do !

certificate $f(x_1) = T$

$f(x_2) = F$

$f(x_3) = T$

operations (to check) is $O(k)$

NAESAT $\in$ NP

Steps 1 to 3: NAESAT is NP-complete

Why this formula

NOT-ALL-EQUAL-(3)SAT (NAESAT)

INSTANCE : a set C of k clauses (denoted $C_j, j \in \{1, \ldots, k\}$) each of which is a disjunction of 3 literals (denoted c_{j1}, c_{j2}, c_{j3}), a set U of l boolean variables denoted x_i. Each literal can be either a boolean variable x_i or its negation $\neg x_i$ (sometimes written $\bar{x}_i$).

QUESTION : is there a truth assignment for U, that is a function $f : U \mapsto \{True, False\}$ such that the k clauses of C are True and every clause has at least one false literal

From **3-SAT**:

replace clause $C_j = c_{1j} \vee c_{2j} \vee c_{3j}$ by

$$(c_{1j} \vee c_{2j} \vee x_j) \wedge (\neg x_j \vee c_{3j} \vee y_j) \wedge (x_j \vee y_j \vee \alpha)$$

(adding 2 variables x_j and y_j per initial clause C_j, plus one global variable α)

NOT-ALL-EQUAL-(3)SAT (NAESAT)
INSTANCE : a set C of k clauses (denoted $C_j, j \in \{1, \ldots, k\}$) each of which is a disjunction of 3 literals (denoted c_{j1}, c_{j2}, c_{j3}), a set U of l boolean variables denoted x_i. Each literal can be either a boolean variable x_i or its negation $\neg x_i$ (sometimes written $\bar{x_i}$).
QUESTION : is there a truth assignment for U, that is a function $f : U \mapsto \{True, False\}$ such that the k clauses of C are True and every clause has at least one false literal
From **3-SAT**:
replace clause $C_j = c_{1j} \vee c_{2j} \vee c_{3j}$ by

$$(c_{1j} \vee c_{2j} \vee x_j) \wedge (\neg x_j \vee c_{3j} \vee y_j) \wedge (x_j \vee y_j \vee \alpha)$$

(adding 2 variables x_j and y_j per initial clause C_j, plus one global variable α)

GRAPH K-COLORABILITY

NAESAT $\leq_p$ GRAPH 3-COLOR

Given an instance of **NAESAT** (C the set of clauses, 3 literals/clause), instance of **GRAPH 3-COLORABILITY** is defined by:

↑ number of colors

1. V is composed of:
   - **1.1** two vertices x_i et $\neg x_i$ for all $x_i \in U$;
   - **1.2** three vertices c_{j1}, c_{j2}, c_{j3} corresponding to C_{j1}, C_{j2}, C_{j3} for every clause C_j;
   - **1.3** one vertex p
2. E is composed of:
   - **2.1** an edge $(x_i, \neg x_i)$ for every couple $x_i, \neg x_i$;
   - **2.2** three edges (c_{j1}, c_{j2}),(c_{j1}, c_{j3}),(c_{j2}, c_{j3}) for every triplet c_{j1}, c_{j2}, c_{j3} ;
   - **2.3** an edge between vertex c_{ji} and vertex x_r (resp. $\neg x_r$) if literal C_{ji} is x_r (resp. $\neg x_r$) ;
   - **2.4** two edges (x_i, p) and $(\neg x_i, p)$ for all i;

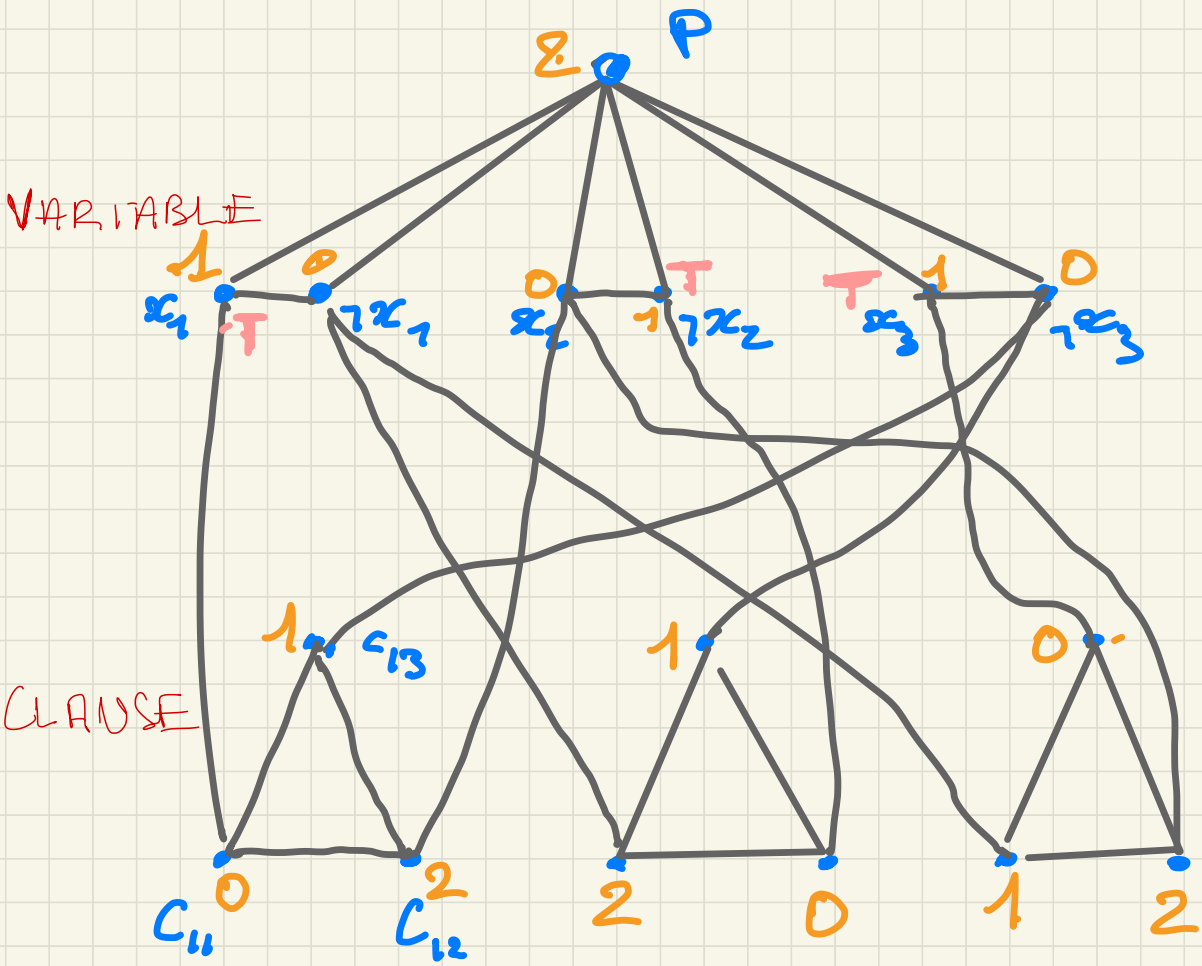
Assuming **NAESAT** is NP-complete, show that **GRAPH 3-COLORABILITY** is NP-complete

NAESAT

Example: assume

$C = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$

Proof: To do !



Step 1 ?

NAESAT : k clauses with #ℓ var.

$$\#V = \underbrace{2\ell}_{x_i, \neg x_i} + \underbrace{3k}_{c_{1j} c_{2j} c_{3j}} + \underbrace{1}_{P}$$

$$\#E = \underbrace{\ell}_{2.1} + \underbrace{3k}_{2.2} + \underbrace{3k}_{2.3} + \underbrace{2\ell}_{2.4}$$

Step 2.1

Assume instance of NAESAT is positive

there is a truth assignment f

s.t. - every clauses True

- at least one literal False in every clause

Coloring (using colors 0,1,2)

- regarding vertices $(x_i, \neg x_i)$, if x_i (resp. $\neg x_i$) is True, color of vertex x_i (resp $\neg x_i$)

set to one
the other vertex of
the pair : set to 0

- for vertex p : set to 2
  - upper triangles are correctly colored (3 ≠ colors)
- for vertices in the "clauses" triangles (down)
  - one literal, say x_i, set to True
    ⇒ corresponding

vertex has a color 1
thus: set the color
of the vertex in
triangle to 0

- one literal set to
False
corresponding vertex
(layer VARIABLE)
has a color 0
thus: set the color
(layer CLAUSE)
to 1

- last vertex: color 2

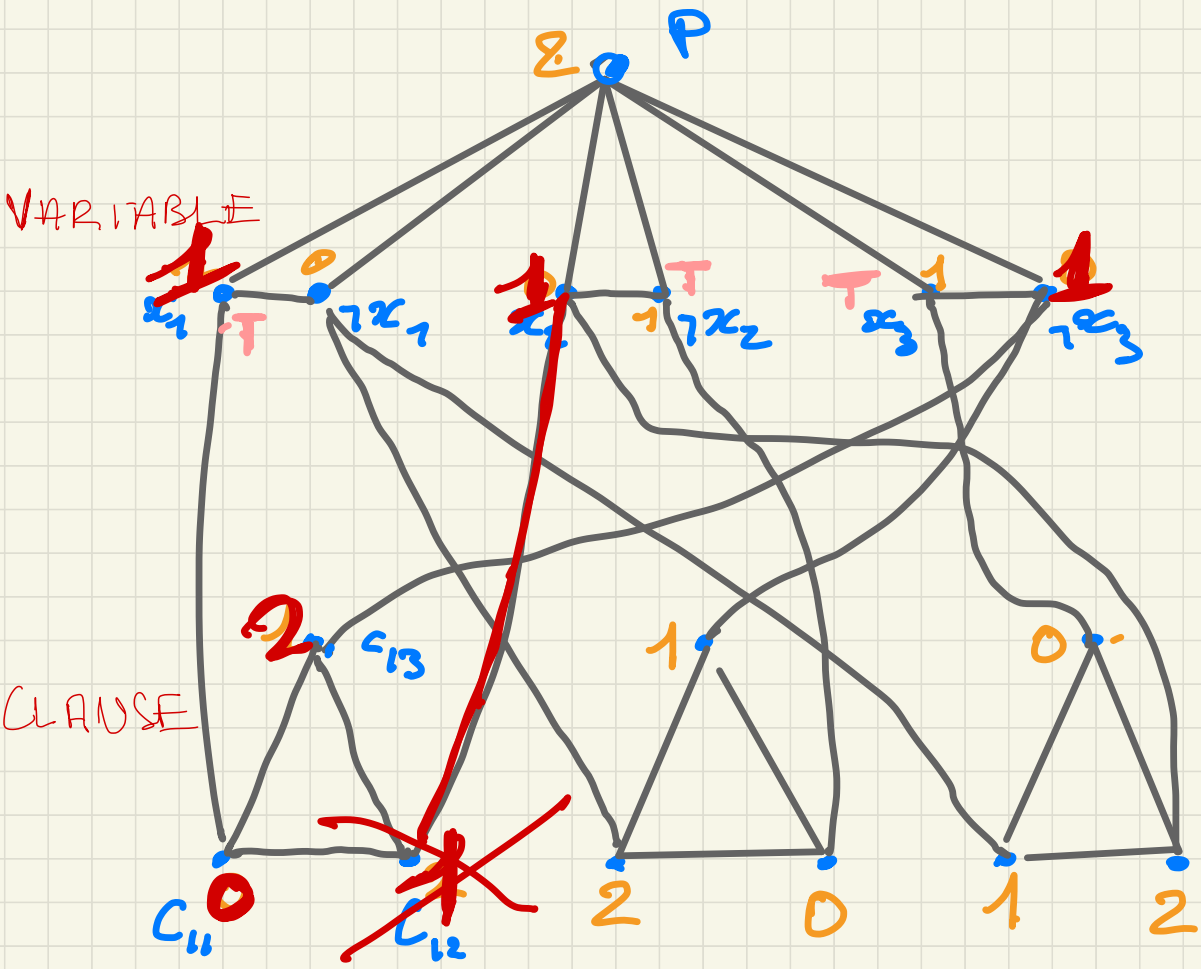
In the end: instance of GRAPH 3-COLORABILITY is positive

NAESAT

Example: assume

$C = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$

Proof: To do !



FROM 3SAT: May be, all literals are True

Step 2.2

Assume positive instance of GRAPH 3-COLORABILITY

thus $\exists\, c : V \to \{0,1,2\}$

s.t. $\#(v,o) \in E$, $c(v) \neq c(v)$

renumber / permute colors s.t. $c(p) = 2$

thus for pairs $(x_i, \neg x_i)$

we must have

$$c(x_i) = 1 \wedge c(\neg x_i) = 0$$

OR

$$c(x_i) = 0 \wedge c(\neg x_i) = 1$$

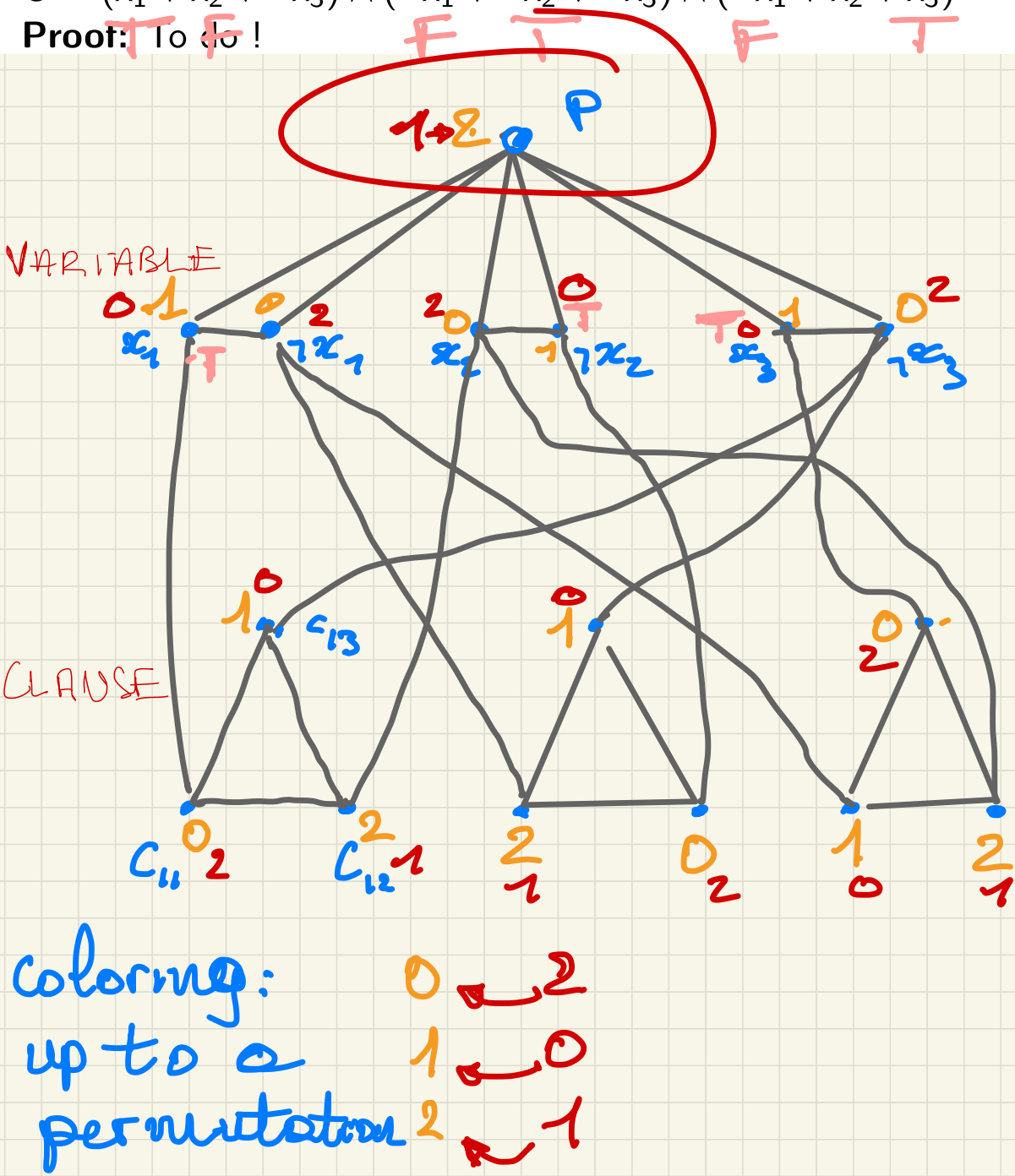
(upper triangles)

NAESAT

Example: assume

$C = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$

Proof: To do !



coloring: up to a permutation

0 ↩ 2

1 ↩ 0

2 ↩ 1

if $c(x_i)=1$ set x_i to True
else set x_i to False.

in the layer CLAUSE:
We must have the
3 colors (0, 1, 2) for the
3 vertices (triangles)

thus:
at least one vertex
has color 1
thus shares an edge
with vertex in
VARIABLE layer

colored with 0
thus the literal
is false

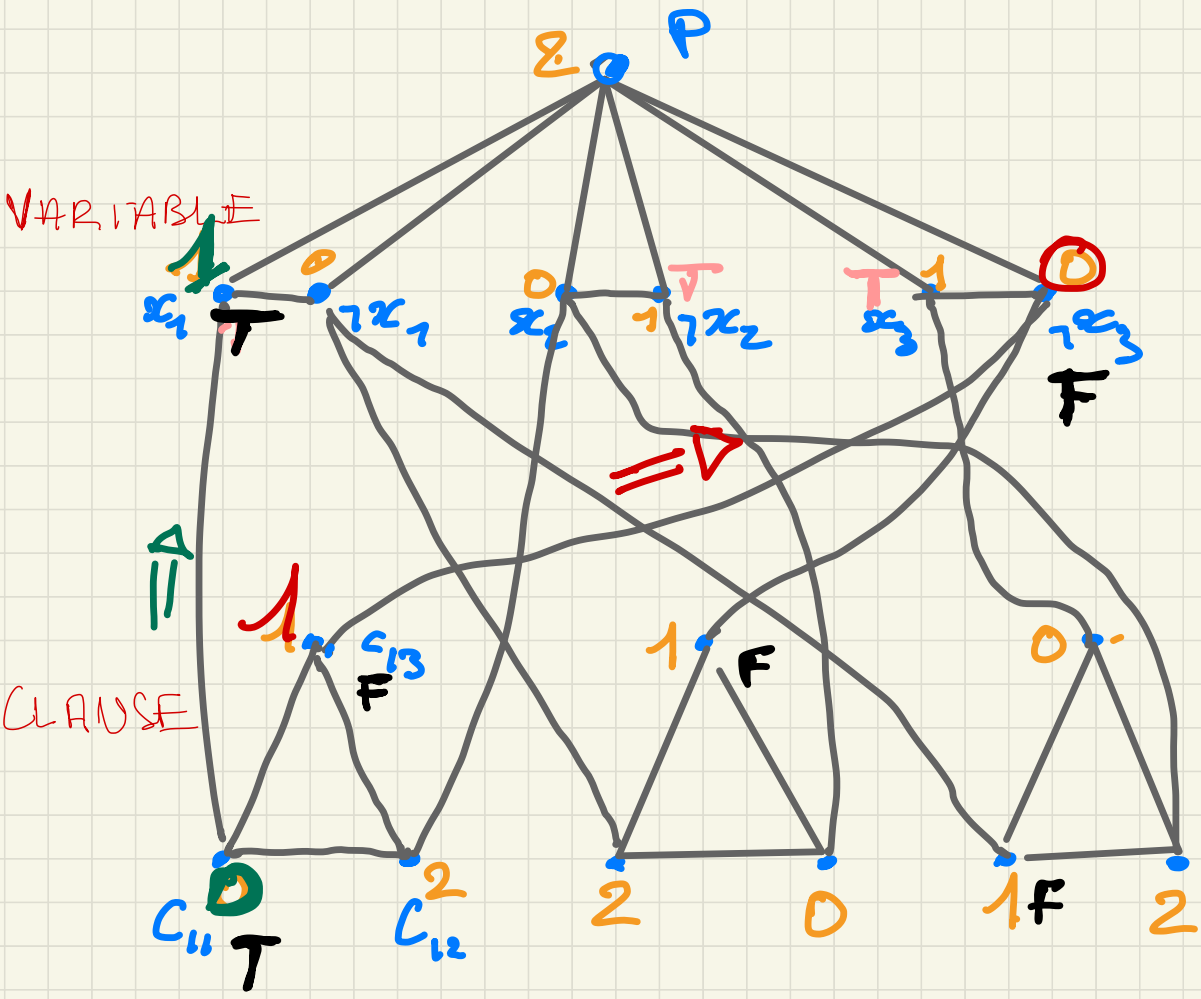
- One other vertex
has color 0
thus share an edge
in VARIABLE layer
colored with 1
thus literal is true

NAESAT

Example: assume

$C = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$

Proof: To do !



Steps 1 and 2

NAESAT $\leq_p$ GRAPH 3-COLORABILITY

Step 3: GRAPH 3-COLORABILITY $\in$ NP

certificate: coloring

$c : V \to \{0, 1, 2\}$ $3^{|V|}$

check:

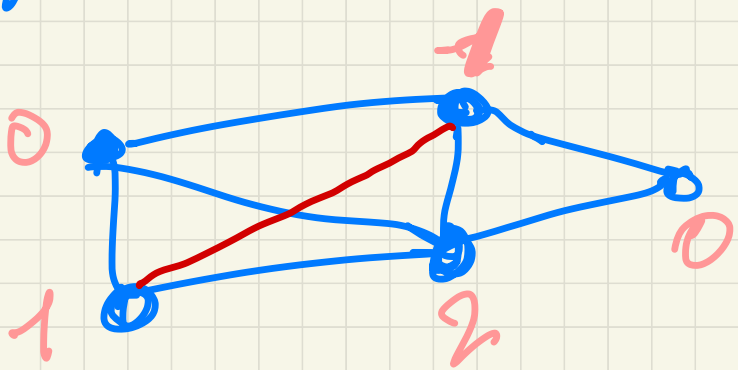
$\forall (u, v) \in E \quad c(u) \neq c(v)$

for all vertices check $c(u) \neq c(v)$ $O(|E|)$

GRAPH 3-COLORABILITY

IS NP-complete

! In step 3, this is for any instance of GRAPH 3-COLOR



Theorem 1

SAT *is NP-complete*

Proof: we must show that $\forall A \in NP, A \leq_P$ **SAT**
Let w a positive instance of A. As $A \in NP$ there is a Non-Deterministic Turing Machine accepts w in polynomial time
Encode this computation into a propositional formula True if and only if w is positive
Denote $w = w_1 \dots w_n$
Let $M = (Q, \Sigma, \Gamma, E, q_0, F, \#)$ the Non-Deterministic Turing Machine
p a polynomial such that $t_M(n) \leq p(n)$.
An accepting computation of w is composed of at most $p(n) + 1$ configurations
Configuration = state + tape's content + head's position

COOK's THEOREM

Store information in:

1. table R of dimension $(p(n)+1) \times (p(n)+1)$ for symbols of the tape. $R(i,j) =$ symbol in cell j at step i
2. a vector Q of dimension $p(n)+1$. $Q(i) =$ state of M at step i
3. a vector P of dimension $p(n)+1$. $P(i) =$ position of the head at step i

Must be encoded in a propositional formula with boolean variables:

1. $r_{ij\alpha}$ with $0 \leq i,j \leq p(n)$ et $\alpha \in \Gamma$;
2. $q_{i\kappa}$ with $0 \leq i \leq p(n)$ et $\kappa \in Q$;
3. p_{ij} with $0 \leq i,j \leq p(n)$.

Number of variables: $O(p^2(n))$

COOK's THEOREM

1. $r_{ij\alpha}$ is True if $R(i,j)$ contains symbol α ;
2. $q_{i\kappa}$ is True if $Q(i)$ is κ ;
3. p_{ij} is True if $P(i)$ is j.

COOK's THEOREM

Computation = set of logical constraints on these variables, which must all be True
e.g. only one α such that $r_{ij\alpha}$ is True, only one κ such that $q_{i\kappa}$ is True...

$$\bigwedge_{0\leq i,j\leq p(n)} \left[\left(\bigvee_{\alpha\in\Sigma} r_{ij\alpha} \right) \wedge \bigwedge_{\alpha'\neq\alpha\in\Sigma} \left(\neg r_{ij\alpha} \vee \neg r_{ij\alpha'} \right) \right] \tag{3}$$

Example: Set $\Sigma = \{a, b, c\}$.
Formula is

$$(r_{ija} \vee r_{ijb} \vee r_{ijc}) \wedge ((\neg r_{ija} \vee \neg r_{ijb}) \wedge (\neg r_{ija} \vee \neg r_{ijc}) \wedge (\neg r_{ijb} \vee \neg r_{ijc}))$$

Verify that only one $r_{ij\alpha}$ must be True
Length: $O(p(n)^2)$

Same kind of formula for $q_{i\kappa}$ (To Do)
For the position of the head:

$$\bigwedge_{0\leq i\leq p(n)} \left[\left(\bigvee_{0\leq j\leq p(n)} p_{ij} \right) \wedge \bigwedge_{0\leq j\neq j'\leq p(n)} (\neg p_{ij} \vee \neg p_{ij'}) \right] \qquad (4)$$

Length: $O(p(n)^3)$

COOK's THEOREM

Formulas encoding that M accepts w:

1. first configuration is an proper initial configuration (length $O(p(n))$)

$$\bigwedge_{\substack{0\leq i,j\leq p(n)\\ \alpha\in\Gamma}} \left[\neg r_{ij\alpha} \vee p_{ij} \vee r_{(i+1)j\alpha}\right] \tag{5}$$

2. transitions are in accord with transition function of M (length $O(p(n)^2)$)

$$\bigwedge_{\substack{0\leq i,j\leq p(n)\\ \alpha\in\Gamma}} \left[(r_{ij\alpha} \wedge \neg p_{ij}) \rightarrow r_{(i+1)j\alpha}\right] \tag{6}$$

or equivalently

$$\bigwedge_{\substack{0\leq i,j\leq p(n)\\ \alpha\in\Gamma}} \left[\neg r_{ij\alpha} \vee p_{ij} \vee r_{(i+1)j\alpha}\right] \tag{7}$$

3. in configuration i, state κ, head on cell j, symbol on tape α, transitions are such: $\sigma(\kappa,\alpha)=\alpha'$,$\delta(\kappa,\alpha)=\kappa'$, $\Delta(\kappa,\alpha)=d$ (with $d\in\{+1,-1\}$). we encode:

$$\bigwedge_{\substack{0\leq i,j\leq p(n)\\ \alpha\in\Gamma}}\left[\begin{array}{c}((q_{i\kappa}\wedge p_{ij}\wedge r_{ij\alpha})\rightarrow r_{(i+1)j\alpha'})\wedge\\ ((q_{i\kappa}\wedge p_{ij}\wedge r_{ij\alpha})\rightarrow q_{(i+1)\kappa'})\wedge\\ ((q_{i\kappa}\wedge p_{ij}\wedge r_{ij\alpha})\rightarrow p_{(i+1)(j+d)})\end{array}\right] \tag{8}$$

which rewrites (length $O(p(n)^2)$):

$$\bigwedge_{\substack{0\leq i,j\leq p(n)\\ \alpha\in\Gamma}}\left[\begin{array}{c}(\neg q_{i\kappa}\vee\neg p_{ij}\vee\neg r_{ij\alpha}\vee r_{(i+1)j\alpha'})\wedge\\ (\neg q_{i\kappa}\vee\neg p_{ij}\vee\neg r_{ij\alpha}\vee q_{(i+1)\kappa'}\wedge\\ (\neg q_{i\kappa}\vee\neg p_{ij}\vee\neg r_{ij\alpha}\vee p_{(i+1)(j+d)})\end{array}\right] \tag{9}$$

4. finale state is reached in at most $p(n)$ steps (length $O(p(n))$)

$$\bigvee_{\substack{0\leq i\leq p(n)\\ \kappa\in F}}[q_{i\kappa}] \tag{10}$$

COOK's THEOREM

Length of formulas: $O(p(n)^3)$
All formulas True if and only if M accepts w in at most $p(n)$ steps

SAT to 3-SAT

$SAT \leq_p 3\text{-}SAT$

1. clauses with one literal (x_1):

   $$(x_1 \vee y_1 \vee y_2) \wedge (x_1 \vee y_1 \vee \neg y_2) \wedge (x_1 \vee \neg y_1 \vee y_2) \wedge (x_1 \vee \neg y_1 \vee \neg y_2)$$

2. clauses with two literals $(x_1 \vee x_2)$:

   $$(x_1 \vee x_2 \vee y) \wedge (x_1 \vee x_2 \vee \neg y)$$

3. clauses with strictly more than three literals:

   $$(x_1 \vee x_2 \vee \ldots \vee x_i \vee \ldots \vee x_l) \tag{11}$$

   rewrites:

   $$\begin{aligned} (x_1 \vee x_2 \vee y_1) \quad & \wedge (\neg y_1 \vee x_3 \vee y_2) \wedge (\neg y_2 \vee x_4 \vee y_3) \wedge \ldots \\ & \wedge (\neg y_{i-2} \vee x_i \vee y_{i-1}) \wedge \ldots \\ & \wedge (\neg y_{l-4} \vee x_{l-2} \vee y_{l-3}) \qquad (12) \\ & \wedge (\neg y_{l-3} \vee x_{l-1} \vee x_l) \qquad (13) \end{aligned}$$

SAT to 3-SAT

SAT $\leq_p$ 3-SAT

1. clauses with one literal (x_1):

$$(x_1 \vee y_1 \vee y_2) \wedge (x_1 \vee y_1 \vee \neg y_2) \wedge (x_1 \vee \neg y_1 \vee y_2) \wedge (x_1 \vee \neg y_1 \vee \neg y_2)$$

T T T T

→ 3-SAT is positive ...

SAT to 3-SAT

SAT $\leq_p$ 3-SAT

1. clauses with one literal (x_1):

$$(x_1 \vee y_1 \vee y_2) \wedge (x_1 \vee y_1 \vee \neg y_2) \wedge (x_1 \vee \neg y_1 \vee y_2) \wedge (x_1 \vee \neg y_1 \vee \neg y_2)$$

2. clauses

3-SAT is positive ...

by contradiction: if $x_1 = F$ consider all values for y_1, y_2 and show always 1 clauses (among the 4) which is F

2. clauses with two literals $(x_1 \vee x_2)$:

$$(x_1 \vee x_2 \vee y) \wedge (x_1 \vee x_2 \vee \neg y)$$

F F T F F F

F F F T

3SAT ≥ 0 $\Rightarrow$ SAT ≥ 0

by contradiction : Assume $\neg$(SAT ≥ 0)

$\Rightarrow$ $x_1 \leftarrow F$

$x_2 \leftarrow F$

whatever value for y

one clause (in 3SAT) is F

contradiction

$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$

$\hookrightarrow (x_1 \vee x_2 \vee y_1) \wedge (\neg y_1 \vee x_3 \vee y_2) \wedge (\neg y_2 \vee x_4 \vee x_5)$

T F T F F

$(x_1 \vee x_2 \vee y_1) \wedge (\neg y_1 \vee x_3 \vee y_2) \wedge (\neg y_2 \vee x_4 \vee y_3) \wedge \ldots$

$\wedge (\neg y_{i-2} \vee x_i \vee y_{i-1}) \wedge \ldots$

$\wedge (\neg y_{l-4} \vee x_{l-2} \vee y_{l-3})$ (12)

$\wedge (\neg y_{l-3} \vee x_{l-1} \vee x_l)$ (13)

SAT to 3-SAT

- If clause (11) is True (say because of x_i
  then in (12) set all y_j to True in clauses before the one of x_i
  and set all y_j to False in clauses after the one of x_i

$$
\begin{aligned}
&(x_1 \vee x_2 \vee y_1) \\
&\wedge(\neg y_1 \vee x_3 \vee y_2) \wedge (\neg y_2 \vee x_4 \vee y_3) \wedge \dots \\
&\wedge(\neg y_{i-2} \vee x_i \vee y_{i-1}) \wedge \dots \\
&\wedge(\neg y_{l-4} \vee x_{l-2} \vee y_{l-3}) \\
&\wedge(\neg y_{l-3} \vee x_{l-1} \vee x_l)
\end{aligned}
\qquad (12)
$$

SAT to 3-SAT

- If clause (??) is True (say because of x_i then in (??) set all y_j to True in clauses before the one of x_i and set all y_j to False in clauses after the one of x_i
- Conversely (clause (M) is False)
  All x_i are False
  Thus y_1 must be True, therefore also y_2,..., therefore also y_{l-3}, and last clause is False

3SAT inst ≥ 0 $\Rightarrow$ SAT inst ≥ 0

P Q

$\neg Q \Rightarrow \neg P$

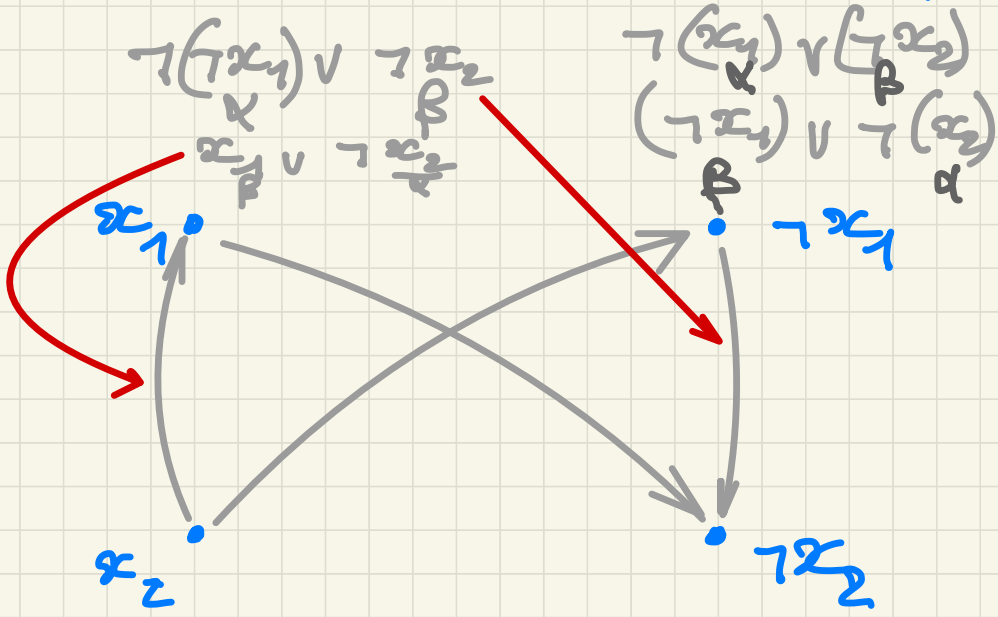
SAT inst < 0 $\Rightarrow$ 3SAT inst < 0

$$(x_1 \vee x_2 \vee \ldots \vee x_i \vee \ldots \vee x_l) \tag{11}$$

rewrites:

$$(x_1 \vee x_2 \vee y_1) \wedge (\neg y_1 \vee x_3 \vee y_2) \wedge (\neg y_2 \vee x_4 \vee y_3) \wedge \ldots$$
$$\wedge (\neg y_{i-2} \vee x_i \vee y_{i-1}) \wedge \ldots$$
$$\wedge (\neg y_{l-4} \vee x_{l-2} \vee y_{l-3}) \tag{12}$$
$$\wedge (\neg y_{l-3} \vee x_{l-1} \vee x_l) \tag{13}$$

$\phi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_2)$



Arc (α, β) if $\neg\alpha \vee \beta$ is clause
or $\beta \vee \neg\alpha$ " "

ϕ NOT satisfiable i.f.f.

$\exists$ variable x_i
s.t path x_i to $\neg x_i$
and $\neg x_i$ to x_i

About **2SAT**

2SAT $\in P$

Let ϕ be a instance of **2SAT** with $U = \{x_1, x_2, \ldots, x_n\}$. We build the directed graph $G(\phi) = (V, E)$:

$$V = \{x_1, x_2, \ldots, x_n, \neg x_1, \neg x_2, \ldots, \neg x_n\},$$

and arcs:

$$E = \{(\alpha, \beta) | (\neg\alpha \vee \beta) \text{ ou } (\beta \vee \neg\alpha) \text{ is a clause of } \phi\}$$

$G(\phi)$ is called the *implication graph*. To understand that, consider the clause $(x_1 \vee x_2)$: if x_1 is not True, then x_2 must be True, or in other words: $\neg x_1 \implies x_2$. Evidently, we have $\neg x_2 \implies x_1$, thus the two arcs in the graph. Applying this principle step by step, note that a path from a vertex α to a vertex β corresponds to $\neg\alpha \implies \beta$. Additionally, this logically implies $\neg\beta \implies \neg\alpha$, thus, a path from $\neg\beta$ to $\neg\alpha$.
Principle 1: path α to β implies path $\neg\beta$ to $\neg\alpha$.

Question 1 *Build the graph* $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_2)$ *What are the modifications if the last clause is deleted from* ϕ*?*

Lemme 1 *Let* ϕ *be an instance of* **2SAT** ϕ *is not satisfiable if and only there exists a variable* $x_i \in U$ *such that there is a path in* $G(\phi)$ *(taking into account the directions) from* x_i *to* $\neg x_i$ *and a path from* $\neg x_i$ *to* x_i*.*

Proof:

- Let us assume a path $x_i \in V$ to $\neg x_i \in V$ and a path from $\neg x_i$ to x_i and assume that ϕ is satisfiable: there is a truth assignment f such that all clauses are True. Because there are paths in both directions we can assume without loss of generality that $f(x_i) = True$ (if not, we consider $\neg x_i$ instead of x_i). In the path from x_i to $\neg x_i$, there is a arc (α, β) such that α has a value True et β a value False. Thus $(\neg\alpha \vee \beta)$ or $(\beta \vee \neg\alpha)$ is not True and ϕ is not, a contradiction;

- reciprocally, assume that there is no variables $x_i \in V$ such that there is a path from x_j to $\neg x_i \in V$ and a path from $\neg x_i$ to x_i. We define a truth assignment f with algorithm 1:

Algorithm 1 Truth assignment from the graph

while there exists $x_i \in U$ s.t. $f(x_i)$ not defined **do**
  $f(x_i) \leftarrow True$
  for all arc (α, β) **do**
    $f(\beta) \leftarrow True$
  end for
end while

Notice first that this algorithm also consider the case of vertices with no arc leaving: indeed these vertices correspond to literal which impose no constraint on any other literal. Second, by setting α to True, all clauses are satisfied. Still, it remains to be proved that the algorithm will not set a variable to True and False. This would be the case if there is an arc from a literal α to the literal β, and an arc from α to $\neg\beta$. But by **Principle 1**, there is also a arc from $\neg\beta$ to $\neg\alpha$, what is not true by hypothesis.

Assumption about the instances used in reduction

e.g. 3SAT

Assume all clauses are composed of 3 distinct literals.

$(x_1 \vee x_1 \vee \neg x_2)$

$x_1 \vee \neg x_1 \vee \neg x_2$

Preprocessing

Any instance $\rightsquigarrow$ Instance + Assumption

e.g.

instance of 3SAT not verifying assump

because $\phi_1 = (x_i \vee x_i \vee x_l)$

$\phi_2 = (x_i \vee x_l \vee y) \wedge (x_i \vee x_l \vee \neg y)$

introducing y

ϕ_1 SAT iff ϕ_2 SAT