SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

FOUNDATIONS OF COMPUTER SCIENCE
LECTURE 16: NPC problems

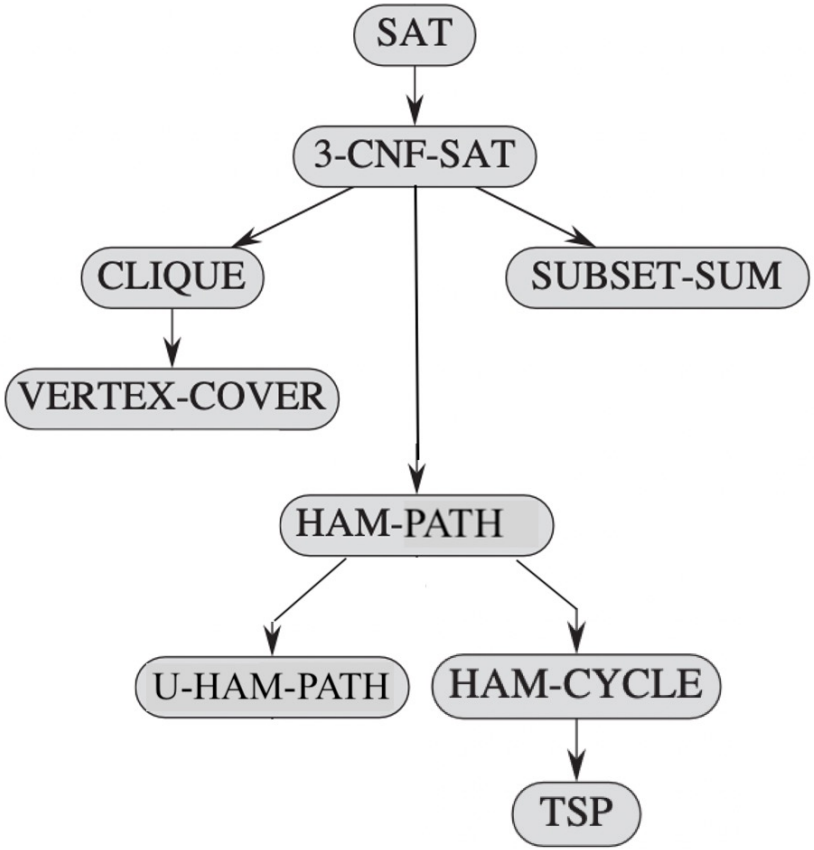
Prof. Daniele Gorla

NPC

For proving that a problem C is **NPC**, we have to

1. Prove that it belongs to **NP**; and
2. Find a $B \in$ **NPC** such that $B \leq_P C$.

This is the path we shall follow
(where $\longrightarrow$ graphically denotes $\leq_P$):



3-CNF-SAT (1)

- We can prove many problems **NP**-complete by reducing from *SAT*
- The reduction algorithm must handle any input formula

  → this requires handling a huge number of cases
- We often prefer to reduce from a restricted language of boolean formulas

  → we need to consider fewer cases
- We must not restrict the language so much that it becomes poly-time solvable

- A ***literal*** in a boolean formula is an occurrence of a variable or its negation.
- A boolean formula is in ***conjunctive normal form***, or ***CNF***, if it is expressed as an AND of ***clauses***, each of which is the OR of one or more literals.
- REMARK: an assignment satisfies a CNF formula if and only if each clause contains at least one literal that evaluates to 1.
- A boolean formula is in ***3-conjunctive normal form***, or ***3-CNF***, if each clause has exactly three distinct literals.

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

Def.: *3-CNF-SAT* = {⟨φ⟩ | φ is a satisfiable 3-CNF formula}

3-CNF-SAT (2)

Thm.: *3-CNF-SAT* ∈ **NP.**

Proof

Trivially, *3-CNF-SAT* ∈ **NP**: the certificate is the assignment; the verification consists in replacing every variable with its assigned value and computing the output.

We have to show that $SAT \leq_P$ *3-CNF-SAT*, i.e. show that every boolean formula φ can be turned into an equivalent (i.e., equi-satisfiable) 3-CNF formula.

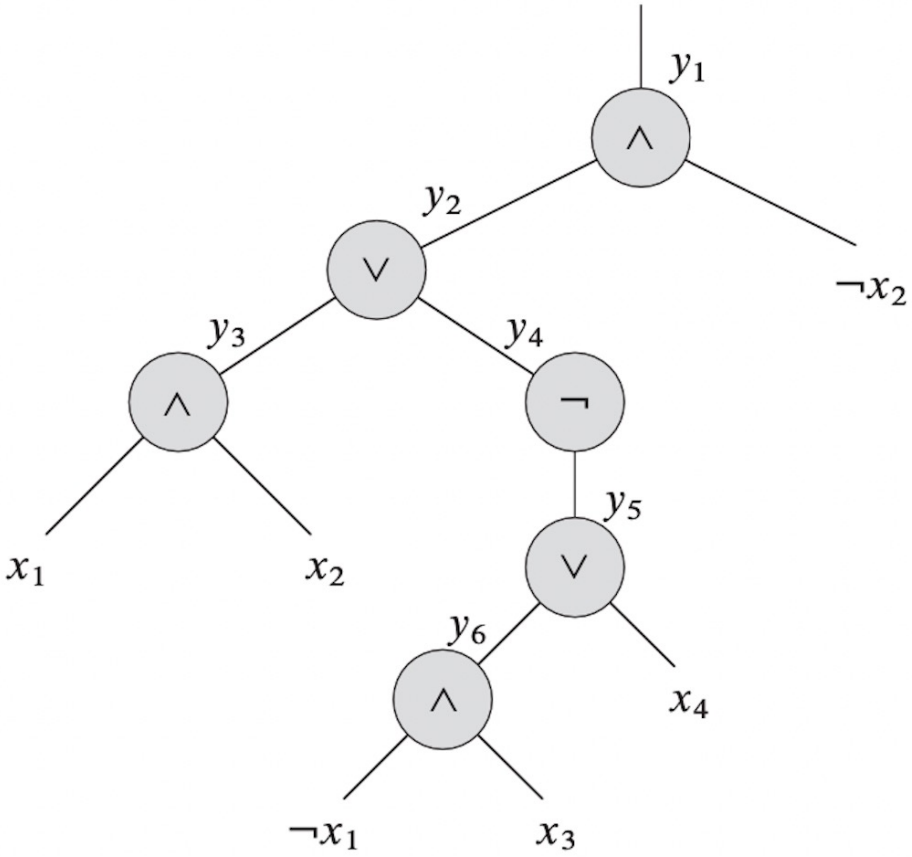
STEP 1:

- construct a binary parse tree for the input formula, with literals as leaves and connectives as internal nodes
- If the input formula contains a clause such as the OR of several literals, we use associativity to fully parenthesize the expression, so that every internal node in the resulting tree has 1 or 2 children.
- We introduce a (new) variable y_i for the output of each internal node.
- The new formula φ' is the AND of the root variable and a conjunction of subformulae describing the operation of each node.
- REMARK: each subformula has at most 3 literals.

3-CNF-SAT (3)

EXAMPLE: Consider the formula $((x_1 \wedge x_2) \vee \neg((\neg x_1 \wedge x_3) \vee x_4)) \wedge \neg x_2$

Its syntactic tree (with nodes annotated with variables) is:



The associated new formula from STEP 1 is:

$$
\begin{aligned}
& y_1 \wedge (y_1 \leftrightarrow (y_2 \wedge \neg x_2)) \\
& \wedge (y_2 \leftrightarrow (y_3 \vee y_4)) \\
& \wedge (y_3 \leftrightarrow (x_1 \wedge x_2)) \\
& \wedge (y_4 \leftrightarrow \neg y_5) \\
& \wedge (y_5 \leftrightarrow (y_6 \vee x_4)) \\
& \wedge (y_6 \leftrightarrow (\neg x_1 \wedge x_3))
\end{aligned}
$$

3-CNF-SAT (4)

STEP 2: convert φ' in CNF to obtain φ''

- Since the new formula is already a conjunction of subformulae, it suffices to convert each of them in CNF
- Since each subformula has at most 3 literals, we can built the truth table for each of them (8 rows at most) and apply the procedure for calculating a CNF from the truth table

EXAMPLE: Consider the subformula $(y_1 \leftrightarrow (y_2 \wedge \neg x_2))$

Its truth table is:

y_1	y_2	x_2	$(y_1 \leftrightarrow (y_2 \wedge \neg x_2))$
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	1

and so its CNF is

$$(\neg y_1 \vee \neg y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee x_2) \wedge (y_1 \vee \neg y_2 \vee x_2)$$

3-CNF-SAT (5)

STEP 3: transforms φ'' into φ''' whose clauses all have *exactly* 3 distinct literals:

- Consider two new (auxiliary) variables p and q
- For each clause C in φ''
  - If C has 3 literals, include C in φ'''
  - If $C = l_1 \vee l_2$ (where l denotes a literal), then include $(l_1 \vee l_2 \vee p) \wedge (l_1 \vee l_2 \vee \bar{p})$ in φ'''
    → one of the clauses is equivalent to $l_1 \vee l_2$, and the other evaluates to 1
  - If $C = l$, then include $(l \vee p \vee q) \wedge (l \vee p \vee \bar{q}) \wedge (l \vee \bar{p} \vee q) \wedge (l \vee \bar{p} \vee \bar{q})$ in φ'''
    → one of the four clauses is equivalent to l, and the others evaluate to 1

CONCLUSION:

It is easy to see that each of the 3 steps preserves satisfiability.

Each step can be done in polynomial time:

- Constructing φ' from φ introduces at most 1 variable and 1 clause for every connective in φ
- Constructing φ'' from φ' can introduce at most 8 clauses for each clause of φ'
- Constructing φ''' from φ'' introduces at most 4 clauses for each clause of φ''

Thus, the size of φ''' is polynomial in the length of φ.

Q.E.D.

CLIQUE (1)

Given an undirected graph $G = (V, E)$, a ***clique*** is a $V' \subseteq V$ such that every pair of vertices in V' are connected by an edge in E.

→ in other words, a clique is a complete subgraph of G

The ***size*** of a clique is the number of vertices it contains.

The ***clique problem*** is the optimization problem of finding a clique of maximum size.

This can be turned into a decision problem by asking whether a clique of a given size k exists.

Def.: $CLIQUE = \{\langle G, k\rangle \mid G$ is an undirected graph containing a clique of size $k\}$.

Thm.: $CLIQUE \in$ **NPC.**

Proof

Trivially, $CLIQUE \in$ **NP**: the certificate is the subset V'; the verification consists in checking that every $v \in V'$ belongs to V, that it has size k, and that, for every $u,v \in V'$, $\{u,v\}$ belongs to E.

→ this requires $O(|V|^2)$

We now show that $3\text{-}CNF\text{-}SAT \leq_P CLIQUE$, i.e., given a 3-CNF formula φ, we define a pair (G, k) such that φ is satisfiable if and only if $\exists\ V' \subseteq V$ s.t. V' is a clique of size k in G.

CLIQUE (2)

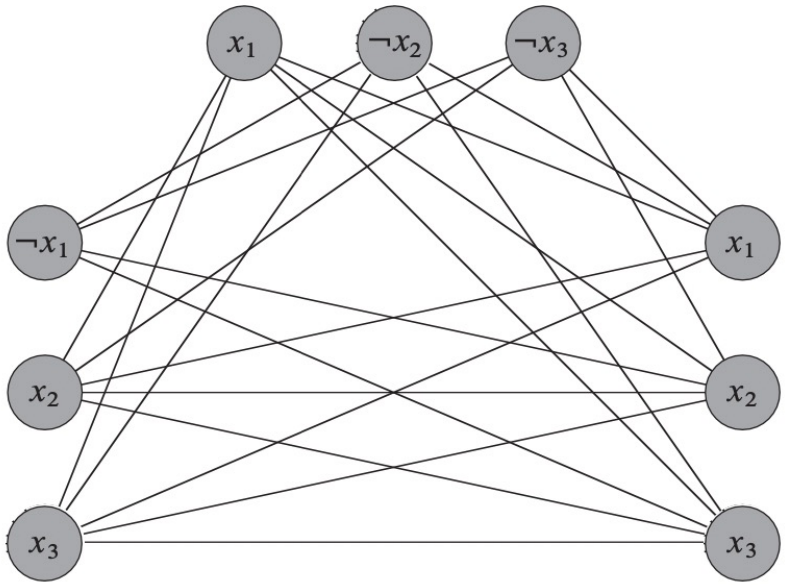
Let $\varphi = C_1 \wedge \ldots \wedge C_k$, where every $C_r = (l_1^r \vee l_2^r \vee l_3^r)$.

We define $G = (V, E)$ to be

- $V = \bigcup_{r=1,\ldots,k} \{v_1^r, v_2^r, v_3^r\}$
- $E = \{ \{v_i^r, v_j^s\} \mid r \neq s \text{ and } l_i^r \neq \neg l_j^s \}$

EXAMPLE: consider the formula $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$

Its associated graph is:



CLIQUE (3)

1. Suppose that φ has a satisfying assignment.

- Each clause C_r contains at least one literal l_i^r that is assigned 1
- Consider the set formed by picking exactly one such true literal from each clause
- This corresponds to a subset V' of V of size k; we claim that V' is a clique.
- Indeed, for any two different vertices v_i^r, $v_j^s \in V'$, by construction of V' we have that $r \neq s$ and that both l_i^r and l_j^s map to 1 by the given satisfying assignment (thus the literals cannot be complementary one of the other).

  → by construction of G, the edge $\{v_i^r, v_j^s\}$ belongs to E.

2. Suppose that G has a clique V' of size k.

- No edge in G connect vertices in the same triple

  → V' contains exactly one vertex per triple
- Let's assign 1 to each literal l_i^r such that $v_i^r \in V'$ (any variable that doesn't correspond to a vertex in the clique may be set arbitrarily)

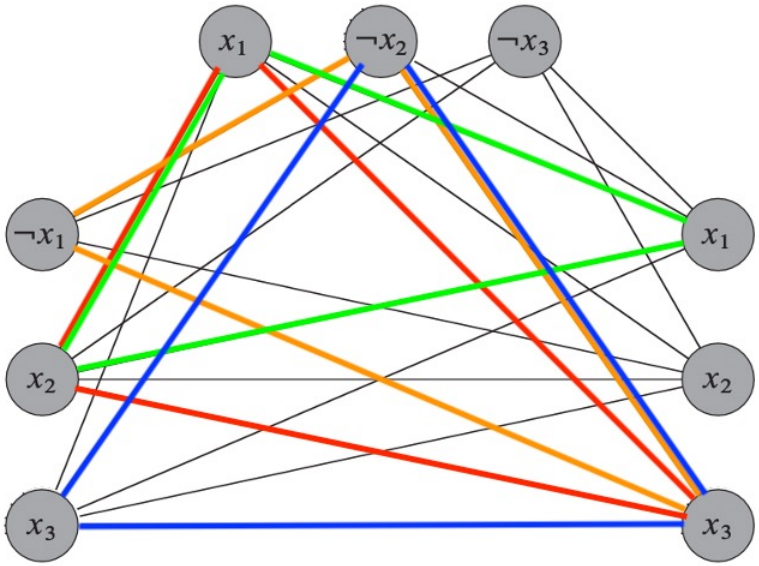
  → we'll never assigning 1 to both a literal and its complement, since G contains no edges between complementary literals
- Each clause is satisfied; so φ is satisfied. Q.E.D.

CLIQUE (4)

Coming back to the example with formula

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

and its associated graph:



Every clique of size 3 corresponds to a satisfying assignment, and vice versa. E.g.:

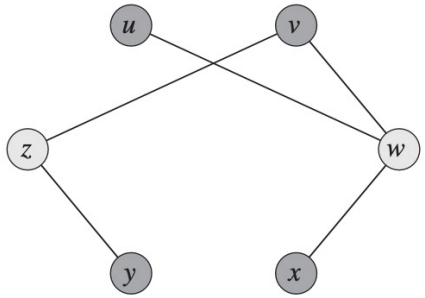
- The clique identified by the red arcs $\Leftrightarrow x_1 = x_2 = x_3 = 1$
- The clique identified by the green arcs $\Leftrightarrow x_1 = x_2 = 1$ (and x_3 can be everything)
- The clique identified by the orange arcs $\Leftrightarrow x_1 = x_2 = 0$ and $x_3 = 1$
- The clique identified by the blue arcs $\Leftrightarrow x_2 = 0$ and $x_3 = 1$ (and x_1 can be everything)
- …

VERTEX-COVER (1)

Given an undirected graph $G = (V, E)$, a ***vertex cover*** is any $V' \subseteq V$ such that, for all $\{u,v\} \in E$, either $u \in V'$ or $v \in V'$ (or both).

→ each vertex covers its incident edges, and a vertex cover is a set that covers all the edges.

EXAMPLE: in the graph



the set of vertices $\{w, z\}$ is a vertex-cover

The ***size*** of a vertex cover is the number of vertices in it.

The ***vertex-cover problem*** is to find a vertex cover of minimum size in a given graph.

This can be turned into a decision problem, by asking whether a graph has a vertex cover of a given size k.

Def.: *VERTEX-COVER* $= \{\langle G, k\rangle \mid G$ is an undirected graph with a vertex cover of size $k\}$.

VERTEX-COVER (2)

Thm.: *VERTEX-COVER* ∈ **NPC.**

Proof

Trivially, *VERTEX-COVER* ∈ **NP**: the certificate is the subset V'; the verification consists in checking that every $v \in V'$ belongs to V, that it has size k, and that, for every $\{u,v\} \in E$, either u or v belongs to V'.

→ this requires $O(|V|+|E|)$

We now show that *CLIQUE* $\leq_P$ *VERTEX-COVER*, i.e., given a pair (G, k), we define a new pair (G', k') such that G has a clique of size k if and only if G' has a vertex cover of size k'.

To this aim, we define ***complement*** of $G = (V, E)$ as $\bar{G} = (V, \bar{E})$, where $\bar{E} = \{ \{u,v\} \mid u,v \in V, u \neq v$ and $\{u,v\} \notin E\}$

Given (G, k) for *CLIQUE*, the pair we consider for *VERTEX-COVER* is $(\bar{G}, |V|-k)$

VERTEX-COVER (3)

1. Suppose that G has a clique V' of size k.

We claim that $V \setminus V'$ is a vertex cover for $\bar{G}$.

Let $\{u,v\}$ be any edge in $\bar{E}$. Then,

- By definition, $\{u,v\} \notin E$
- Being V' a clique, at least one of u or v does not belong to V', since every pair of vertices in V' is connected by an edge of E.
- Hence, at least one of u or v belongs to $V \setminus V'$, as desired.

Finally, $|V \setminus V'| = |V| - k$, being $|V'| = k$.

2. Suppose that $\bar{G}$ has a vertex cover V' of size $|V| - k$.

We claim that $V \setminus V'$ is a clique for G.

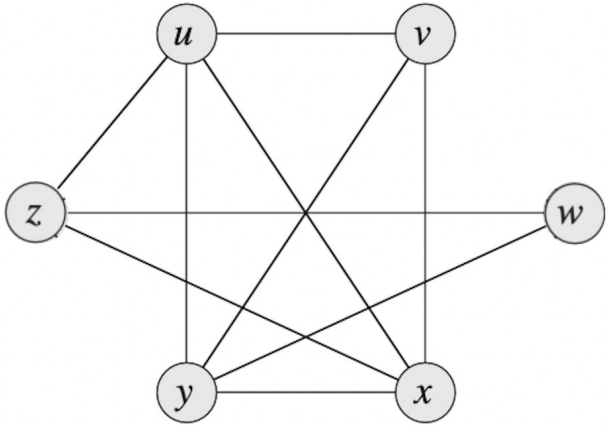
Let u,v be any pair of different vertices in $V \setminus V'$. Then,

- $\{u,v\}$ cannot belong to $\bar{E}$, otherwise it would not be covered by V' (that we assumed to be a vertex cover)
- Hence, by definition, $\{u,v\} \in E$, as desired.

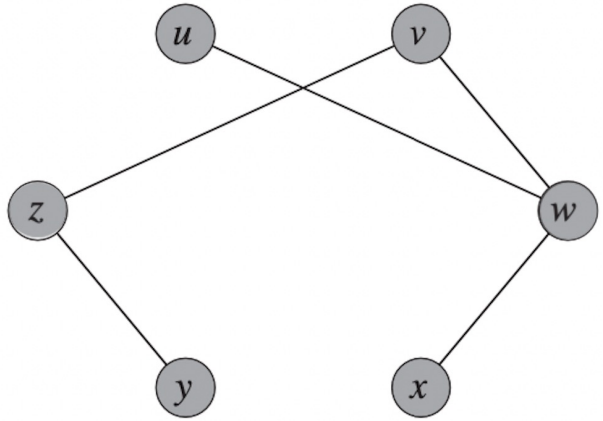
Finally, $|V \setminus V'| = k$, being $|V'| = |V| - k$. Q.E.D.

VERTEX-COVER (4)

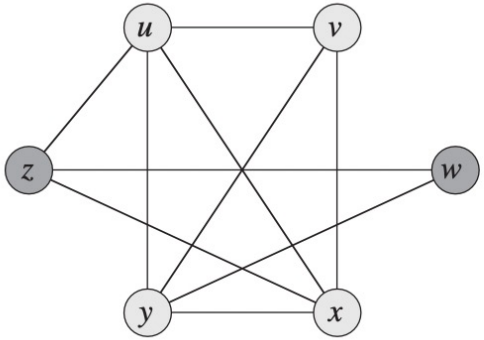
EXAMPLE: Consider the graph

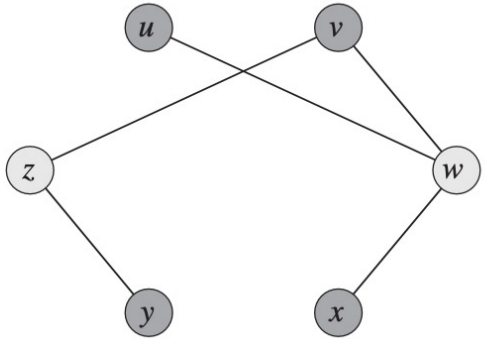


Its complement is the graph



A clique of size 4 exists in the left graph (i.e., $\{u,v,x,y\}$) and corresponds to a vertex cover of size 2 in the right one (i.e., $\{w,z\}$):





SUBSET-SUM (1)

Given a finite set S of naturals and a ***target*** $t > 0$, say whether there exists an $S' \subseteq S$ whose elements sum to t, i.e. $t = \sum_{s \in S'} s$.

For example, if $S = \{1,3,5,7\}$ and $t = 10$, then the answer is YES and the subset $\{3,7\}$ is a certificate. By constrast, if $t = 2$, the answer is NO.

Def.: *SUBSET-SUM* $= \{\langle S, t\rangle \mid \exists\, S' \subseteq S \text{ s.t. } t = \sum_{s \in S'} s\}$

Thm.: *SUBSET-SUM* $\in$ **NPC.**

Proof

Trivially, *SUBSET-SUM* $\in$ **NP**. Indeed, the certificate is the subset S' that sums to t:

- Check that every element of S' belongs to S; and
- Check that $t = \sum_{s \in S'} s$.

Both tasks can be performed in $O(|S| + \log_2 t)$ (the size of $\langle S, t\rangle$).

We now show that *3-CNF-SAT* $\leq_P$ *SUBSET-SUM*, i.e., given a 3-CNF formula φ, we define a pair (S, t) such that φ is satisfiable if and only if $\exists\, S' \subseteq S$ s.t. $t = \sum_{s \in S'} s$

SUBSET-SUM (2)

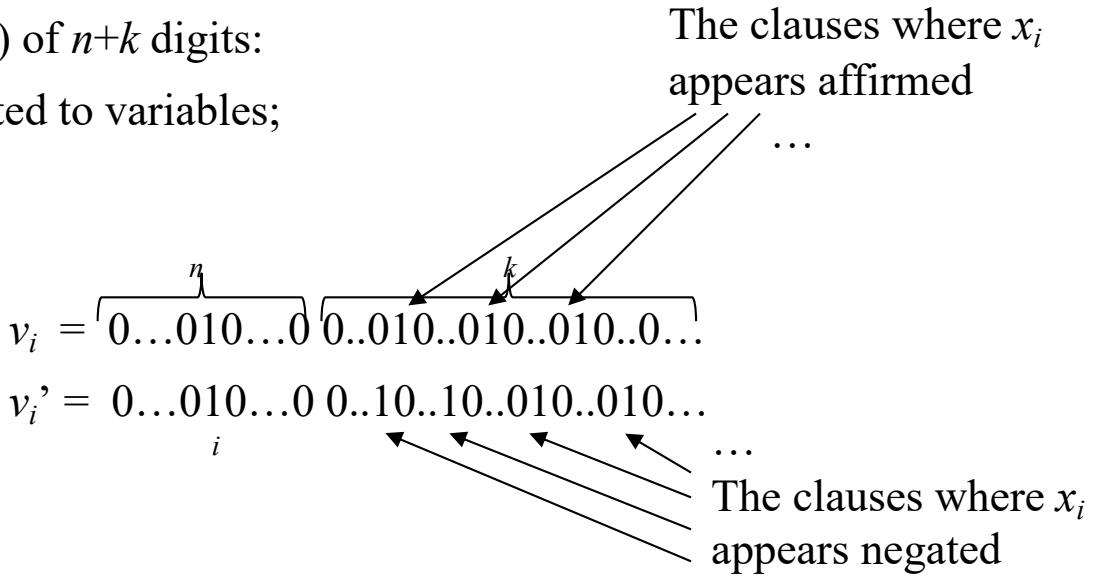
Let φ' be obtained by removing from φ every clause that contains both a variable and its negation

→ such clauses have no impact on the satisfiability of φ since they are always true

Let $x_1, x_2, \ldots, x_n$ be the variables occurring in φ' and $C_1, C_2, \ldots, C_k$ be its clauses.

The reduction considers naturals (in base 10) of $n+k$ digits:

- The most signifying n digits are associated to variables;
- The less signifying k ones to clauses.
- S contains
  - For every variable x_i, two naturals



  - For every clause C_j, two naturals

$$s_j = 0\ldots0\ \underset{j}{0\ldots010\ldots0}$$

$$s_j' = \underbrace{0\ldots0}_{n}\ \underbrace{0\ldots020\ldots0}_{k}$$

- $t = \underbrace{1\ldots1}_{n}\underbrace{4\ldots4}_{k}$

REMARK: all these naturals are pairwise distinct

SUBSET-SUM (3)

EXAMPLE:

$C_1 \wedge C_2 \wedge C_3 \wedge C_4$
where $C_1 = (x_1 \vee \neg x_2 \vee \neg x_3), C_2 = (\neg x_1 \vee \neg x_2 \vee \neg x_3), C_3 = (\neg x_1 \vee \neg x_2 \vee x_3)$ and $C_4 = (x_1 \vee x_2 \vee x_3)$

Then, set S is:

		x_1	x_2	x_3	C_1	C_2	C_3	C_4
v_1	=	1	0	0	1	0	0	1
v_1'	=	1	0	0	0	1	1	0
v_2	=	0	1	0	0	0	0	1
v_2'	=	0	1	0	1	1	1	0
v_3	=	0	0	1	0	0	1	1
v_3'	=	0	0	1	1	1	0	0
s_1	=	0	0	0	1	0	0	0
s_1'	=	0	0	0	2	0	0	0
s_2	=	0	0	0	0	1	0	0
s_2'	=	0	0	0	0	2	0	0
s_3	=	0	0	0	0	0	1	0
s_3'	=	0	0	0	0	0	2	0
s_4	=	0	0	0	0	0	0	1
s_4'	=	0	0	0	0	0	0	2

That is, S = {1001001, 1000110, 100001, 101110, 10011, 11100, 1000, 2000, 100, 200, 10, 20, 1, 2}

Number t is 1114444.

SUBSET-SUM (4)

REMARK: $\sum_{s \in S} s = 2\ldots26\ldots6$ and, in particular, no carry arises.

COMPLEXITY:

- S contains $2n + 2k$ values, each of which has $n + k$ digits;
- the time to produce each digit is polynomial in $n + k$;
- t has $n + k$ digits and it is produced in constant time;
- Hence, the reduction requires polynomial time.

1. If $\langle\varphi\rangle \in$ *3-CNF-SAT*, then $\langle S, t\rangle \in$ *SUBSET-SUM*:

Let us consider a satisfying assignment for φ and build S' as follows:

- For all $i = 1,\ldots,n$, if $x_i = 1$, then include v_i into S'; otherwise, include v_i' into S'

  → this ensures that the first n digits of $\Sigma_{s \in S'} s$ are all 1's, as needed to obtain t

  → Each of the last k digits of the sum is at least 1

  How can we ensure that each of the last k digits of the sum is 4?

- For all $j = 1,\ldots,k$

  - If C_j has 3 literals that evaluate to 1 under the given assignment, then include s_j into S'
  - If C_j has 2 literals that evaluate to 1 under the given assignment, then include s_j' into S'
  - If C_j has 1 literal that evaluate to 1 under the given assignment, then include s_j and s_j' into S'

SUBSET-SUM (5)

2. If $\langle S, t\rangle \in$ *SUBSET-SUM*, then $\langle \varphi \rangle \in$ *3-CNF-SAT*:

Let S' be a subset of S that sums to t.

For every $i = 1,\ldots,n$, S' must contain exactly one between v_i and v_i' (see the n m.s.d. of t).

Let us now consider the boolean assignment

$$x_i = \begin{cases} 1 & \text{if } v_i \in S' \\ 0 & \text{otherwise} \end{cases}$$

Every C_j is satisfied by this assignment. To prove this, consider the digit associated to it:

- it must sum to 4
- s_j and s_j' can give at most 3 in position j
- at least one v_i or one v_i' (with 1 in position j) must belong to S'
- if it is $v_i \in S$', then x_i is set to 1 by the assignment and x_i occurs affirmed in C_j

  → C_j is satisfied thanks to the 1 assigned to x_i
- If it is v_i' $\in S$', then x_i is set to 0 by the assignment and x_i occurs negated in C_j

  → C_j is satisfied thanks to the 1 assigned to $\bar{x}_i$

Q.E.D.

SUBSET-SUM (6)

Coming back to the previous example:

Formula $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$ is satisfied, e.g., by putting $x_1 = x_2 = 0$ and $x_3 = 1$.

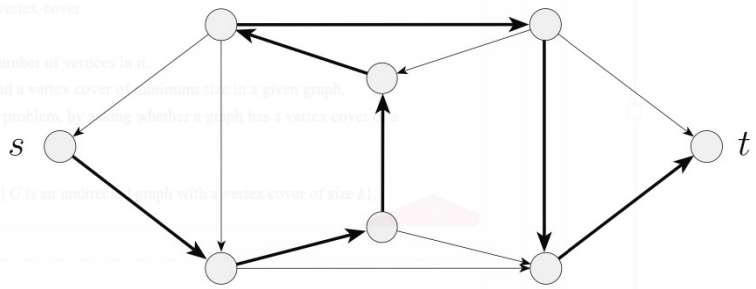
The subset of S that sums to the desired t and corresponds to this assignment is the one enlighted in the side table:

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

HAM-PATH (1)

A ***Hamiltonian path*** (from *s* to *t*) in a directed graph *G* is an *st*-path that goes through each node exactly once.

EXAMPLE:



Def.: *HAM-PATH* = {⟨*G*,*s*,*t*⟩ | *G* is a directed graph with a Hamiltonian path from *s* to *t*}.

Theor.: *HAM-PATH* ∈ **NPC**.

Proof

Easily, *HAM-PATH* ∈ **NP**: a certificate is any sequence of vertices, for which we have to check:

1. The first and last vertices are *s* and *t*, respectively;
2. Every adjacent pair of vertices is an edge of *G*; and
3. Every vertex is touched once and only once.

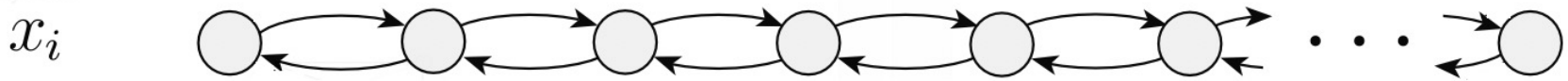
HAM-PATH (2)

We now show that *3-CNF-SAT* $\leq_P$ *HAM-PATH*.

Let $\varphi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \cdots \wedge (a_k \vee b_k \vee c_k)$, where

- each a, b, and c is a literal x_i or $\bar{x}_i$
- x_1 , ... , x_l are the variables of φ.

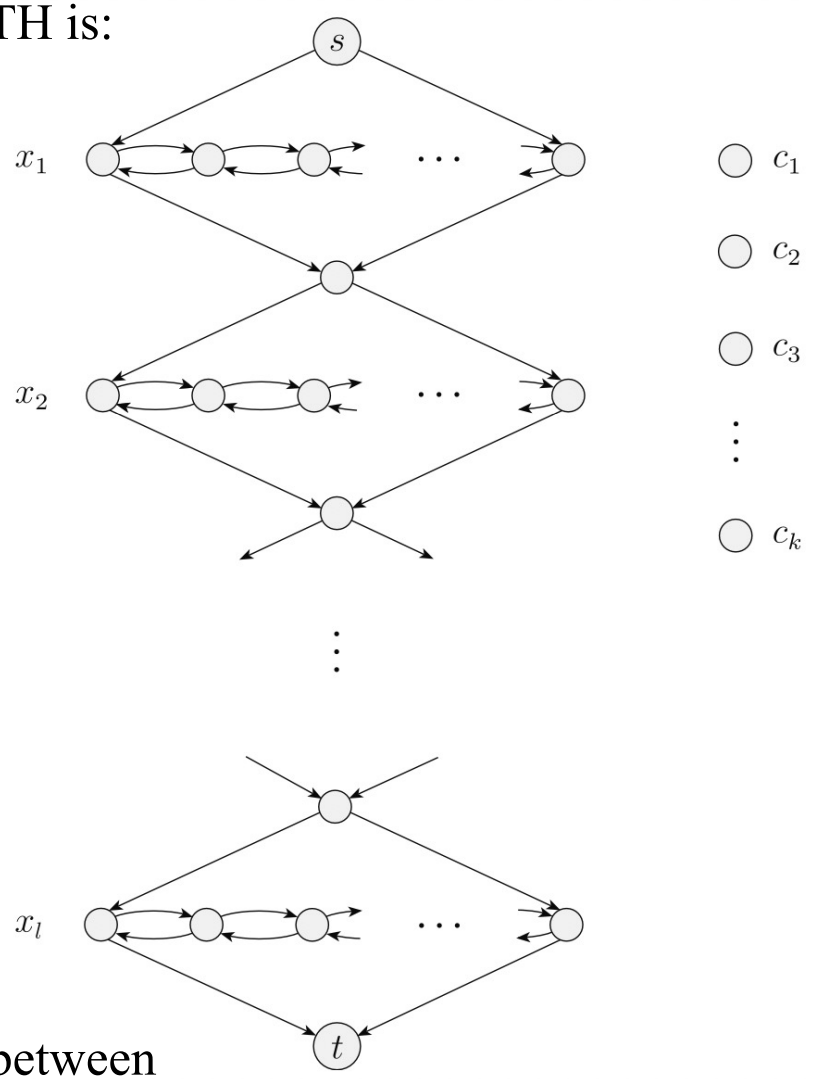
We represent each of the l variables x_i with a horizontal row of $3k + 3$ nodes:



Each of the k clauses is represented by a distinct node:

HAM-PATH (3)

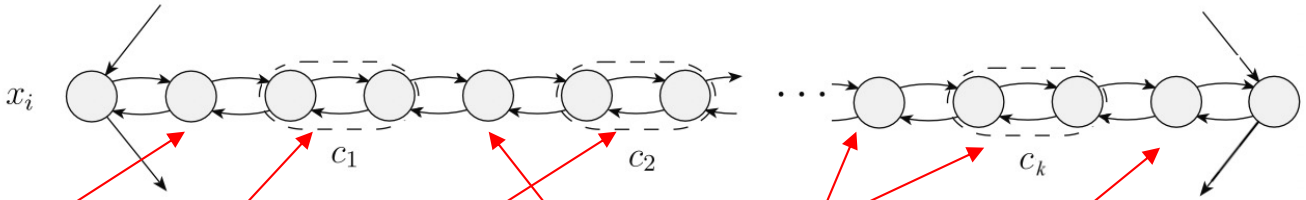
The overall layout of the graph for HAM-PATH is:



REMARK: we still need to add connections between the clause nodes and the variable nodes.

HAM-PATH (4)

We consider the nodes of the line for variable x_i grouped as follows:



IDEA:

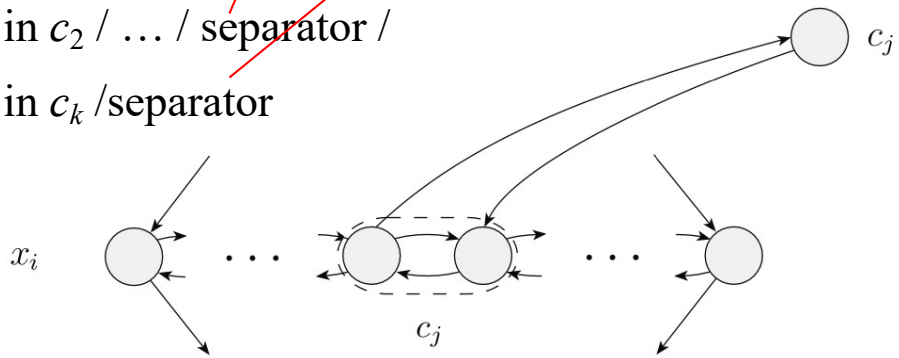
- Apart from the first and last node, we (logically) have a sequence:

  separator / two nodes for representing x_i in c_1 / separator /

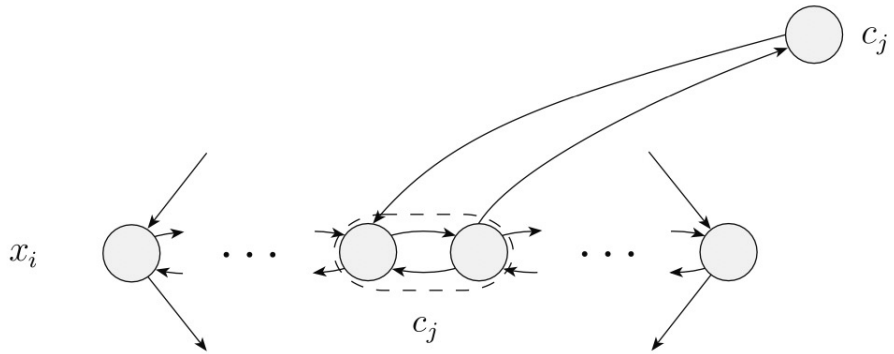
  two nodes for representing x_i in c_2 / … / separator /

  two nodes for representing x_i in c_k /separator

- If literal x_i appears in clause c_j, we add the two edges:



- If literal $\bar{x}_i$ appears in clause c_j, we add the two edges:



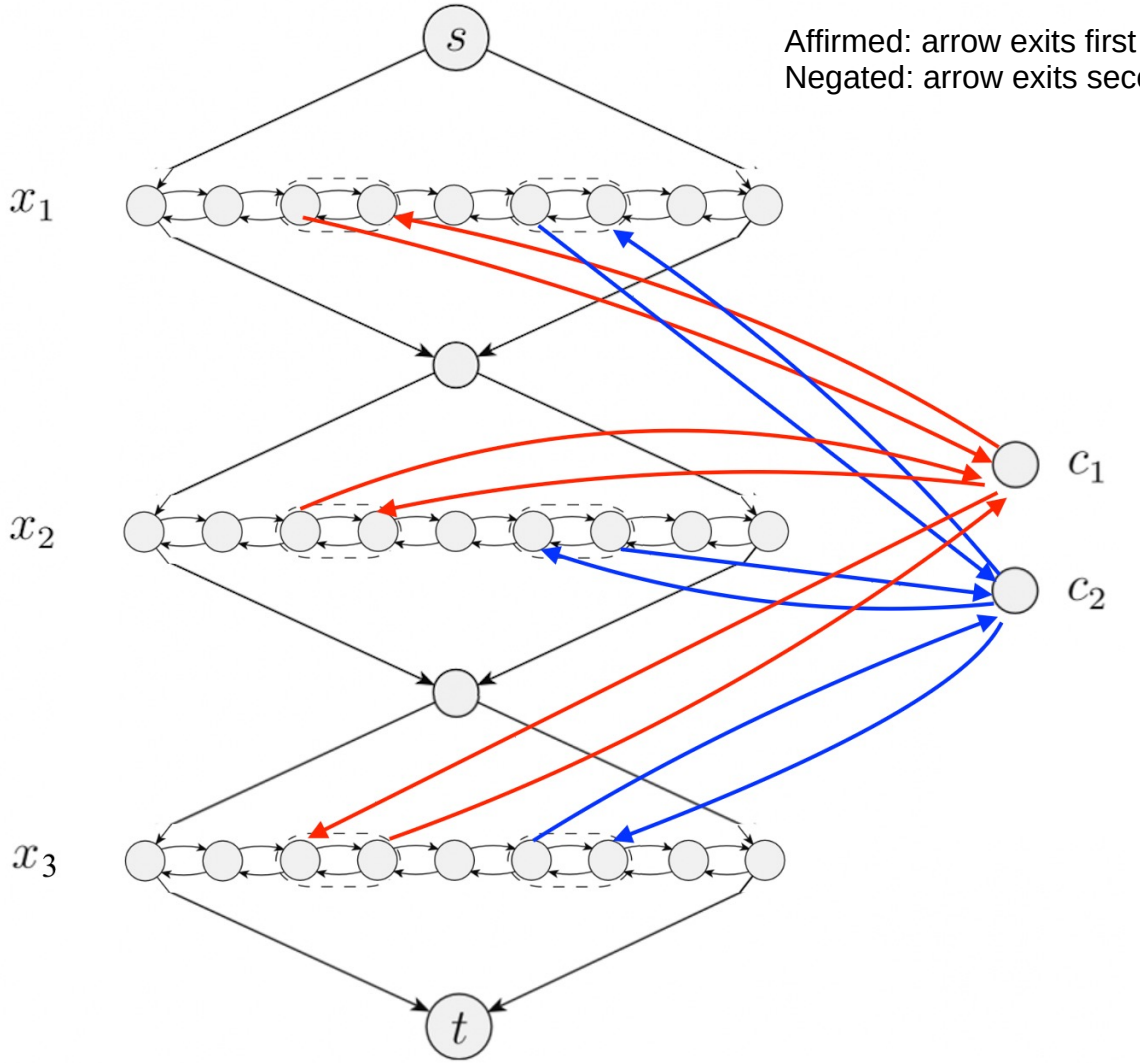
HAM-PATH (5)

EXAMPLE: consider the formula $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3)$

Its associated graph is:

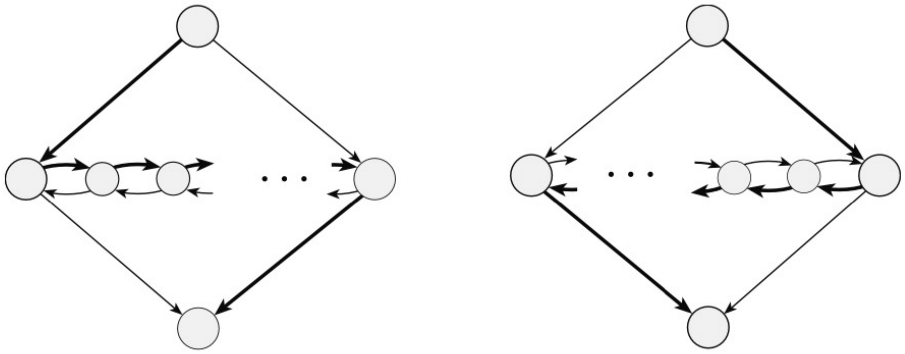
(for clarity, we draw in red the edges for the 1st clause and in blue the edges for the 2nd one)

there is no ham-path for unsatisfiable formulae

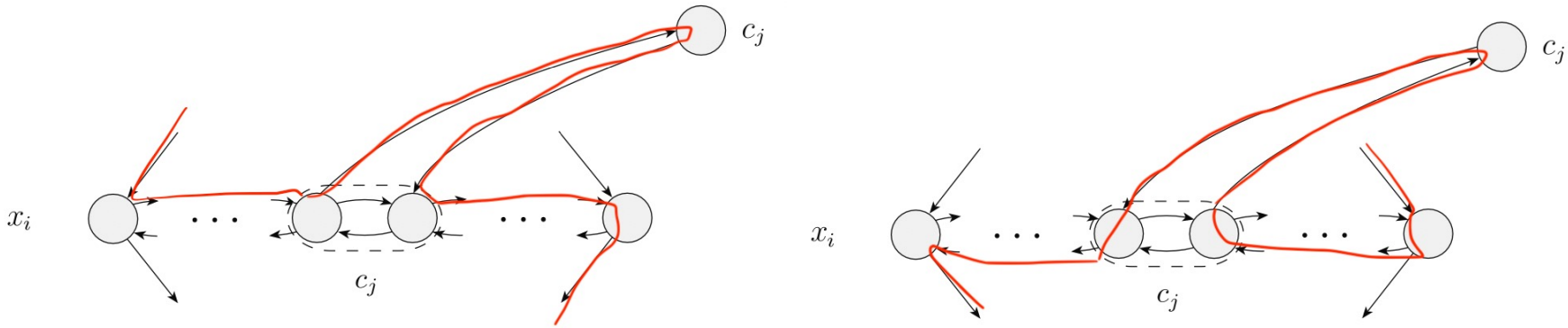


HAM-PATH (6)

Obs.1: every diamond (i.e., structure associated to a variable) can be hamiltonianly visited only in 2 possible ways:



OBS.2: to visit the node associated to a clause, we must do a detour while visiting the diamond corresponding to a variable that is contained in that clause, either during a zig-zag (if the variable occurs affirmed) or during a zag-zig (if the variable occurs negated):



HAM-PATH (7)

Suppose that φ is satisfiable (hence, there exists an assignment for the variables that satisfies every clause); let's show a Hamiltonian path from s to t in G:

We first ignore the clause nodes.

- The path begins at s, goes through each diamond in turn, and ends up at t.
- To visit the diamond associated to x_i, if x_i is assigned TRUE, the path zig-zags the diamond, otherwise the path zag-zigs it.

To visit the clause nodes:

- for each clause, select only one literal that holds TRUE
- For clause j:
  - if the literal that you selected is x_i, detour to c_j during the zig-zag of the i-th diamond
  - if the literal that you selected is $\bar{x}_i$, detour to c_j during the zag-zig of the i-th diamond
- Note that each true literal in a clause provides an *option* of a detour to touch the clause node (if several literals in a clause are true, only one detour must be taken).

HAM-PATH (8)

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

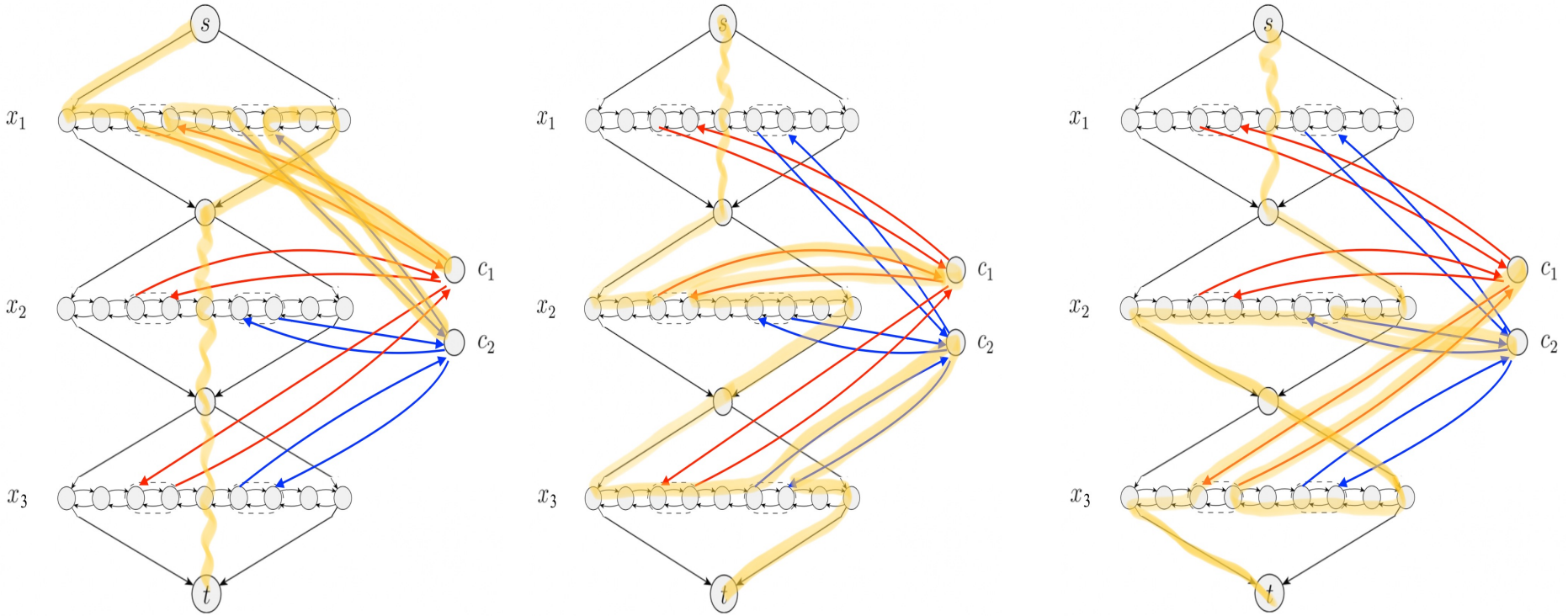
EXAMPLE: consider again the formula $(x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3)$

It has many satisfying assignments, e.g.:

$x_1 = 1$ $\qquad$ $x_2 = x_3 = 1$ $\qquad$ $x_2 = x_3 = 0$

These correspond to the following Hamiltonian paths in the associated graph:

I have to choose one and only one variable to go to clause node. if there are more than one, choose one



where the squeezed arrow crossing a diamond means that the diamond can be indifferently zig-zagged or zag-zigged.

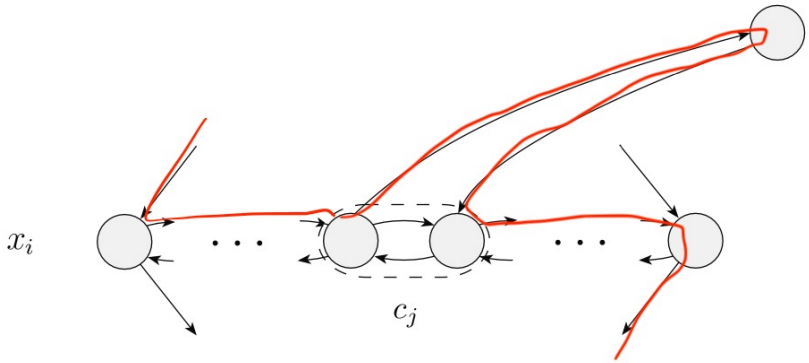
HAM-PATH (9)

Suppose that G has a Hamiltonian path from s to t; let's build a satisfying assignment for φ.

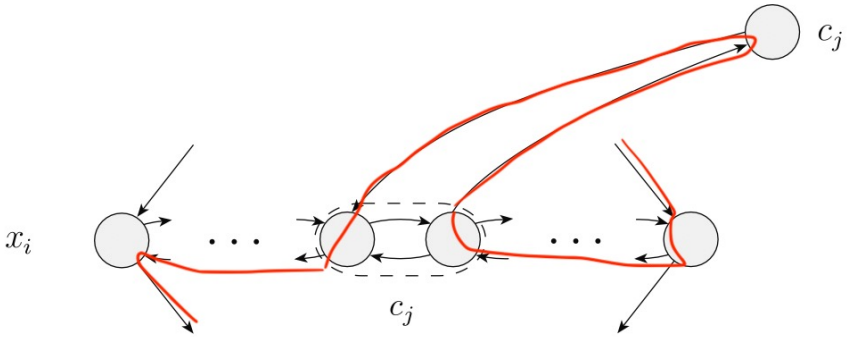
- If the Hamiltonian path is *normal*—that is, it goes through the diamonds in order from the top one to the bottom one, except for the detours to the clause nodes—we can easily obtain a satisfying assignment:

  If the path zig-zags through a diamond, we assign to the corresponding variable TRUE, otherwise FALSE.

- Because each clause node appears on the path, by observing how the detour to it is taken, we may conclude that the clause is TRUE:



x_i occurs affirmed in c_j
and it is assigned TRUE

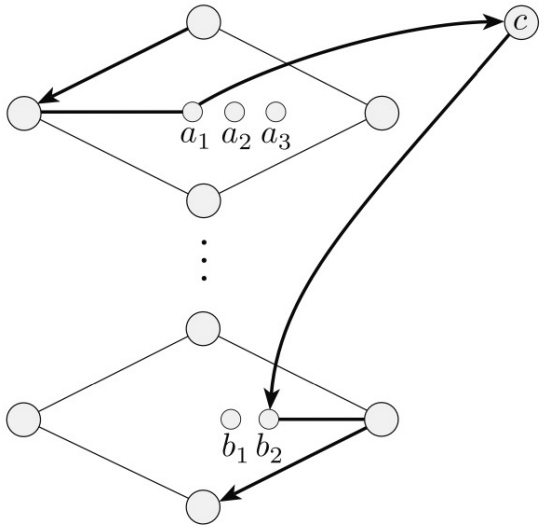


x_i occurs negated in c_j
and it is assigned FALSE

- All that remains to be shown is that every Hamiltonian path must be normal.

HAM-PATH (10)

Normality may fail only if the path enters a clause node from one diamond but returns to another:



OBS.1: a_1 cannot be a separator node because separators have not edges that go to clause nodes; so, either a_2 or a_3 must be a separator node (according to whether a_1 is the second or the first node of the pair for c in the diamond).

OBS.2: a_2 separator → the edges entering a_2 are from a_1 and a_3;
a_3 separator → the edges entering a_2 are from a_1, a_3, and c.

If the side situation happens, the path could not contain node a_2 (and so cannot be Hamiltonian):

- The path cannot enter a_2 from c or a_1 because the path goes elsewhere from these nodes
- If the path enters a_2 from a_3, then the path cannot exit from a_2:
  - a_3 has been used for entering
  - a_1 and c are already touched by the path.

REMARK: the construction is polynomial in the size of φ (a formula with l variables and k clauses): G has $l(3k+3) + (l+1) + k$ nodes.

Q.E.D.

REMARK: the separator nodes are needed only to simplify the proof that every hamiltonian path must be normal.

U-HAM-PATH (1)

U-HAM-PATH is the same problem as *HAM-PATH* but on *undirected* graphs.

Theor.: *U-HAM-PATH* ∈ **NP**.

Proof

Like for *HAM-PATH*, it is straithforward to show that *U-HAM-PATH* ∈ **NP**.

We now show that *HAM-PATH* $\leq_P$ *U-HAM-PATH*.

Given a directed graph G (with n vertices) and $s,t \in V_G$, we build an undirected graph G' (with $3n$ vertices) as follows:

1. For all $u \in V_G$, G' contains vertices u', u'', u''' and edges $\{u', u''\}$ and $\{u'', u'''\}$.
2. For all (u, v) belonging to E_G, then $\{u''', v'\}$ is an edge of G'.

Then we prove that

$$(G, s, t) \in \textit{HAM-PATH} \quad \text{iff} \quad (G', s', t''') \in \textit{U-HAM-PATH}$$

(→) Let $s = u_1, \ldots, u_n = t$ be a hamiltonian path in G. Then trivially $u_1', u_1'', u_1''', \ldots, u_n', u_n'', u_n'''$ is a hamiltonian path in G'.

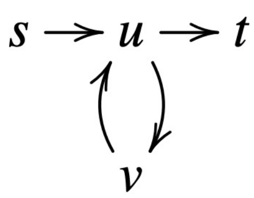
(←) Suppose that we have a hamiltonian path in G'. The only way to include any u'' is to use the edges $\{u', u''\}$ and $\{u'', u'''\}$. Then each u''' must have one single edge to some v'. By starting from s' until we arrive to t''', we can find a hamiltonian path from s to t in G.

Q.E.D.

U-HAM-PATH (2)

<u>REMARK:</u> in the previous proof, it would NOT be correct to have just 2 vertices in the undirected graph for every vertex of the directed one.

EXAMPLE: Consider the directed (non-hamiltonian) graph:



By replacing every node *x* with just a pair of nodes *x*' and *x*'', we would have:

$s' - s'' - u' - u'' - t' - t''$

$\qquad\qquad |\qquad |$

$v'' - v'$

that has the Hamiltonian path: $s', s'', u', v'', v', u'', t', t''$.

By contrast, replacing every vertex with 3 vertices solves this problem:

$s' - s'' - s''' - u' - u'' - u''' - t' - t'' - t'''$

$\qquad\qquad |\qquad\qquad |$

$v''' - v'' - v'$

Indeed, no hamiltonian path exists here, thanks to *u*''.

HAM-CYCLE

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Def.: *HAM-CYCLE* = {⟨*G*⟩ | *G* is a directed graph with a Hamiltonian path from some *u* to itself}.

Theor.: *HAM-CYCLE* ∈ **NPC**.

Proof

Easily, *HAM-CYCLE* ∈ **NP**.

We now prove that *HAM-PATH* $\leq_P$ *HAM-CYCLE*.

Given a directed graph *G* and $s,t \in V_G$, we build a new directed graph *G'* with:

- one new vertex *v*, and
- two new edges $v \longrightarrow s$ and $t \longrightarrow v$.

If $s = u_1, \ldots, u_n = t$ is an hamiltonian path from *s* to *t* in *G*, then $v, u_1, \ldots, u_n, v$ is an hamiltonian cycle in *G'*.

Conversely, if $u_1, \ldots, u_{n+2}$ is an hamiltonian cycle in *G'* (hence, $u_1 = u_{n+2}$), then it must touch *v*.

1. If $v = u_1 = u_{n+2}$, then $u_2 = s$ and $u_{n+1} = t$; so, $u_2, \ldots, u_{n+1}$ is an hamiltonian path from *s* to *t* in *G*.
2. Otherwise $v = u_i$, for some $i \in \{2,..,n+1\}$. Then $u_{i-1} = t$ and $u_{i+1} = s$; so, $u_{i+1}, \ldots, u_{n+2} = u_1, \ldots, u_{i-1}$ is an hamiltonian path from *s* to *t* in *G*.

Q.E.D.

TSP (1)

In the ***traveling-salesman problem***, a salesman must visit n cities with the lowest cost tour:

- The salesman wishes to make a tour, by visiting each city exactly once and finishing at the city he started from.
- The salesman incurs a nonnegative integer cost $c(i, j)$ to travel from city i to city j, and the salesman wishes to make the tour whose total cost is minimum (the total cost is the sum of the individual costs along the steps of the tour).

We model the problem as a complete (directed) graph with n vertices and edges weighted with naturals.

Def.: $\text{TSP} = \{ \langle G, c, k \rangle : G = (V, E)$ is a complete (undirected) graph, $c: V \times V \longrightarrow \mathbb{N}$, $k \in \mathbb{N}$, G has a traveling-salesman tour with cost at most $k \}$

Theor.: *TSP* $\in$ **NP**C.

Proof

To prove that *TSP* $\in$ **NP**, observe that a certificate is a sequence of vertices. The verification checks that: (1) this sequence contains each vertex exactly once, (2) sums up the edge costs, and (3) checks whether the sum is at most k.

TSP (2)

We now prove that *HAM-CYCLE* $\leq_P$ *TSP*.

Let $G = (V, E)$ be an instance of *HAM-CYCLE*.

We construct an instance of *TSP* to be $(G', c, 0)$, where

- $G' = (V, E')$ and $E' = \{(u,v) \mid u,v \in V \text{ and } u \neq v\}$
- $c(u,v) = \begin{cases} 0 & \text{if } (u,v) \in E \\ 1 & \text{otherwise} \end{cases}$

Since all (and only) the edges of G have cost 0 in G', it is very easy to see that G has a hamiltonian cycle if and only if graph G' has a tour of cost 0.

Q.E.D.