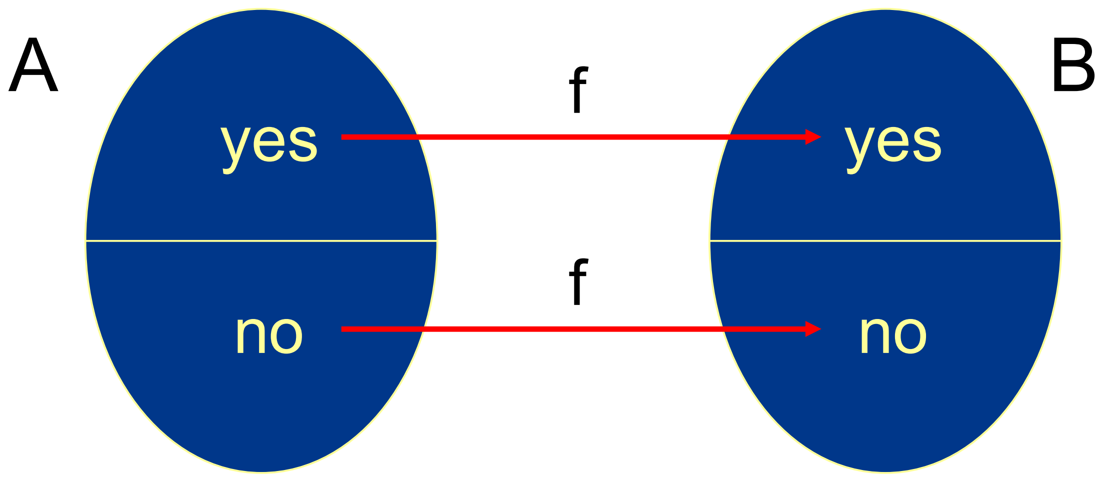
7. Time Complexity

- P and NP
  - Measuring complexity
  - The class P
  - The class NP
- NP-Completeness
  - Polynomial time reducibility
  - The definition of NP-Completeness
  - The Cook-Levin Theorem
- NP-Complete Problems

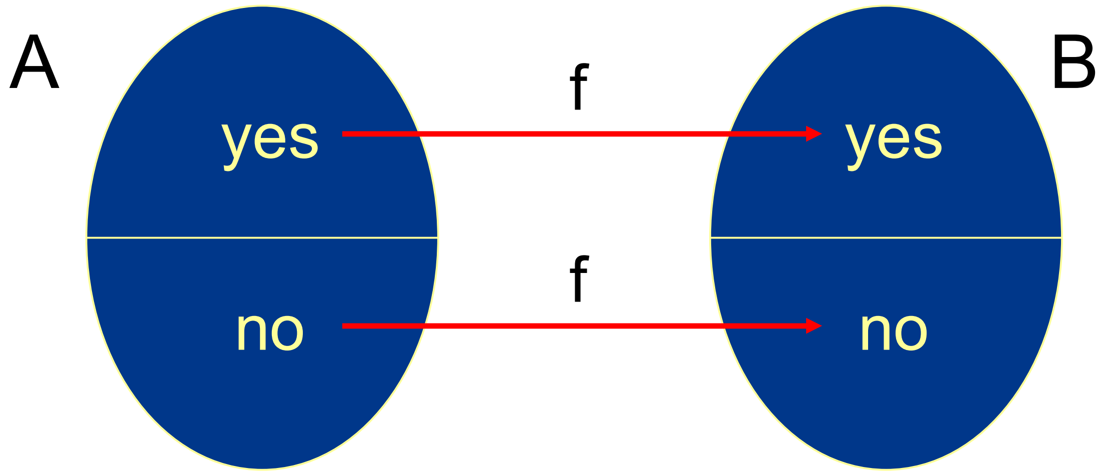
Poly-Time Reductions

- Type of reduction we will use:
  - “many-one” poly-time reduction (commonly)
  - “mapping” poly-time reduction (book)



reduction from language A to language B

Poly-Time Reductions



- function f should be poly-time computable

Definition: $f : \Sigma^* \to \Sigma^*$ is poly-time computable if for some $g(n) = n^{O(1)}$ there exists a $g(n)$-time TM M_f such that on every $w \in \Sigma^*$, M_f halts with $f(w)$ on its tape.

Poly-Time Reductions

Definition: $A \leq_P B$ ("A reduces to B") if there is a poly-time computable function f such that for all w

$$w \in A \Leftrightarrow f(w) \in B$$

- as before, condition equivalent to:
  - YES maps to YES *and* NO maps to NO
- as before, meaning is:
  - B is at least as "hard" (or expressive) as A

Poly-Time Reductions

Theorem: if $A \leq_P B$ and $B \in P$ then $A \in P$.

Proof:

– A poly-time algorithm for deciding A:

- on input w, compute f(w) in poly-time.
- run poly-time algorithm to decide if $f(w) \in B$
- if it says "yes", output "yes"
- if it says "no", output "no"

NP-Completeness

Definition: A language B is NP-complete if it satisfies two conditions:

1. B is in NP, and
2. Every A in NP is polynomial time reducible to B.

B is called NP-hard if we omit the first condition.

Theorem: If B is NP-complete and $B \in P$, then P=NP.

Theorem: If B is NP-complete and $B \leq_P C$ for C in NP, then C is NP-complete.

SAT

- A Boolean formula is satisfiable if some assignment of TRUE/FALSE to the variables makes the formula evaluate to TRUE.
- SAT = {<φ> | φ is a satisfiable Boolean formula}
  - E.g. $\Phi = (\neg x \wedge y) \vee (x \wedge \neg z)$

The Cook-Levin Theorem

- **Theorem:** SAT is NP-complete.

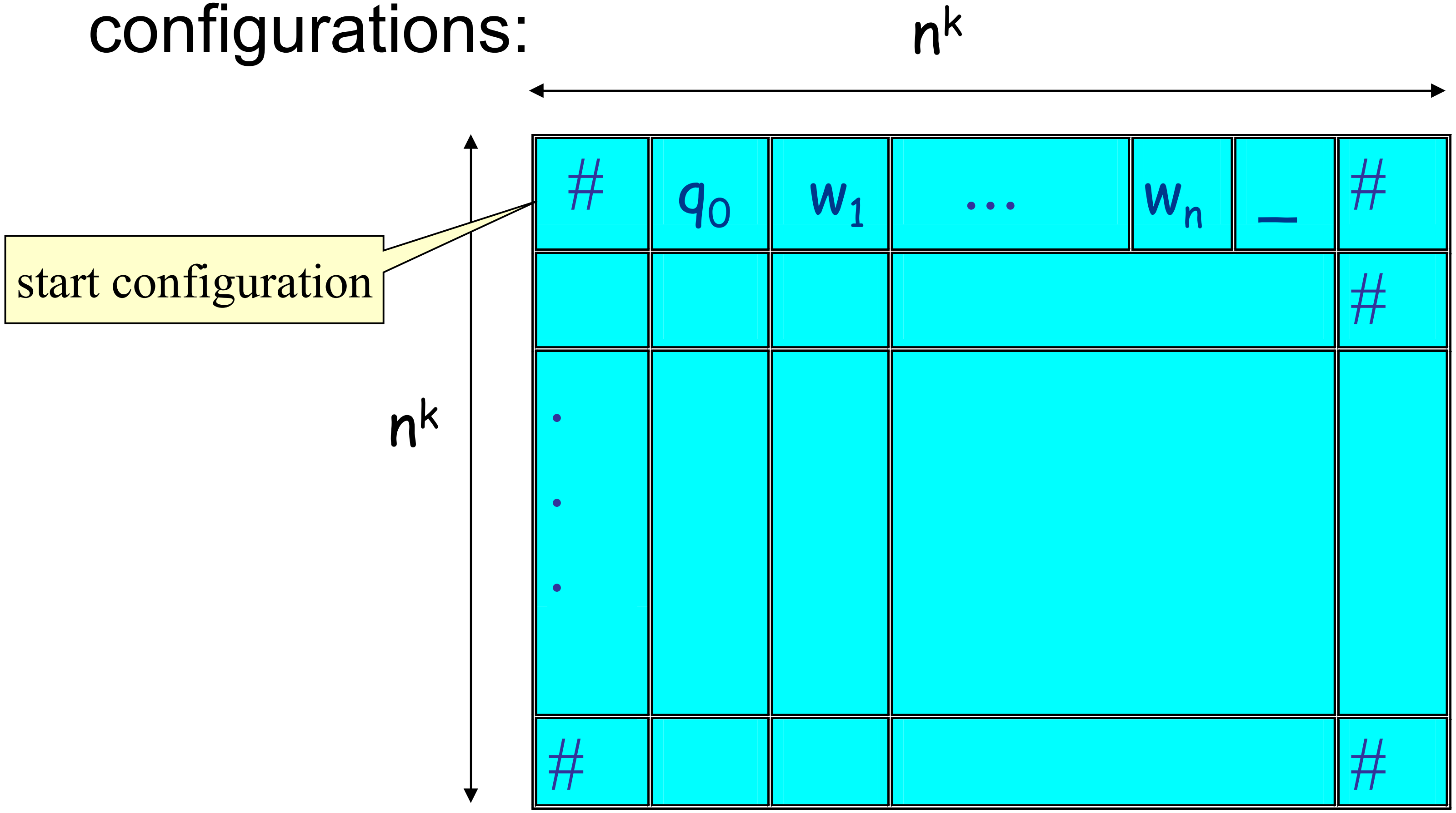
- **Proof**:
  - SAT is in NP
  - for any language A in NP, A is polynomial time reducible to SAT.

SAT is NP-Complete

- SAT $\in$ NP
  - guess an assignment to the variables, check the assignment
- $A \leq_P$ SAT (for any $A \in$ NP)
  - Proof idea: let M be a NTM that decides A in n^k time. For any input string w, we construct a Boolean formula $\varphi_{M,w}$ which is satisfiable iff M accepts w.

SAT is NP-Complete

- Represent a computation by a table of configurations:



SAT is NP-Complete

- The variables of the formula

$$x_{i,\, j,\, s}$$

- $i, j \in [1, n^k]$; $s \in Q \cup \Gamma \cup \{\#\}$
- It stands for "Is s the content of cell[i, j]?"
  - TRUE: s is the content of cell[i, j]
  - FALSE: s is not the content of cell[i, j]

SAT is NP-Complete

- The formula

$$\varphi_{M,w} = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}$$

cell content consistency

input consistency

transition legal

machine accepts

SAT is NP-Complete

- Requirement on each cell:

$$\varphi_{cell} = \bigwedge_{1\le i,j\le n^k}\left[\underbrace{\left(\bigvee_{s\in C} x_{i,j,s}\right)}_{\text{The (i,j) cell must contain some symbol}} \wedge \underbrace{\left(\bigwedge_{s\neq t\in C}\left(\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}}\right)\right)}_{\text{It shouldn't contain different symbols.}}\right]$$

The (i,j) cell must contain some symbol

It shouldn't contain different symbols.

SAT is NP-Complete

- Assuming the input string is $w_1 w_2 \ldots w_n$, we can explicitly state the start configuration in the first step.

$$\phi_{start} = x_{(1,1,\#)} \wedge x_{(1,2,q_0)} \wedge$$
$$x_{(1,3,w_1)} \wedge \cdots \wedge x_{(1,n+2,w_n)} \wedge$$
$$x_{(1,n+3,\_)} \wedge \cdots \wedge x_{(1,n^k-1,\_)} \wedge x_{(1,n^k,\#)}$$

- The accepting state is visited during the computation.

$$\varphi_{accept} = \bigvee_{1 \le i,j \le n^k} x_{i,j,q_{accept}}$$

SAT is NP-Complete

- Legal transitions: all 2x3 windows in the tableau are legal.

$$\varphi_{\text{move}} = \bigwedge_{1 \le i,j \le n^k} \bigvee_{a_1,\ldots,a_6} \left(x_{i-1,j,a_1} \wedge \ldots \wedge x_{i+1,j+1,a_6} \right)$$

for any $a_1,\ldots,a_6$ s.t. this is a legal window

a_1	a_2	a_3
a_4	a_5	a_6

SAP is NP-Complete

- Legal windows, e.g.

a	q_1	b
q_2	a	c

a	q_1	b
a	a	q_2

a	a	q_1
a	a	b

#	b	a
#	b	a

a	b	a
a	b	q_2

b	b	b
c	b	b

- Illegal windows, e.g.

a	b	a
a	a	a

a	q_1	b
q_1	a	a

b	q_1	b
q_2	b	q_2

SAP is NP-Complete

$$\varphi_{M,w} = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}$$

- $|\varphi_{M,w}| = |\phi_{cell}| + |\phi_{start}| + |\phi_{move}| + |\phi_{accept}|$

  $= O(n^{2k}) + O(n^k) + O(n^{2k}) + O(n^{2k})$

  $= O(n^{2k})$

- φ can be generated in polynomial time, and is satisfiable iff the TM accepts the input w.

3SAT

- x, $\neg x$ are literals; a clause is several literals connected with $\vee$s; a cnf-formula comprises several clauses connected with $\wedge$s; it is a 3cnf-formula if all the clauses have three literals.
  – E.g. $(x \vee y \vee \neg z) \wedge (\neg x \vee w \vee z)$
- 3SAT = $\{\langle\varphi\rangle \mid \varphi$ is a satisfiable 3cnf-formula$\}$

3SAT is NP-Complete

- 3SAT is in NP.
  - 3SAT is a special case of SAT, and is therefore clearly in NP.
- In order to show it's also NP-Complete, we alter the proof of SAT's NP-Completeness to generate a 3CNF formula.

3SAT is NP-Complete

Given a TM and an input we've produced a conjunction of:

$$\varphi_{cell} = \bigwedge_{1\le i,j\le n^k}\left[\left(\bigvee_{s\in C} x_{i,j,s}\right)\wedge\left(\bigwedge_{s\ne t\in C}(\overline{x_{i,j,s}}\vee\overline{x_{i,j,t}})\right)\right]$$

$$\varphi_{start} = x_{1,1,\#}\wedge x_{1,2,q_0}\wedge x_{1,3,w_1}\wedge\ldots\wedge x_{1,n+2,\_}\wedge\ldots\wedge x_{1,n^k-1,\_}\wedge x_{1,n^k,\#}$$

$$\varphi_{move} = \bigwedge_{1\le i,j\le n^k}\ \bigvee_{\text{legal } a_1,\ldots,a_6}\left(x_{i-1,j,a_1}\wedge\ldots\wedge x_{i+1,j+1,a_6}\right)$$

$$\varphi_{accept} = \bigvee_{1\le i,j\le n^k} x_{i,j,q_{accept}}$$

3SAT is NP-Complete

- $\rightarrow$ CNF
  - All the sub-formulas, but ϕ_{move}, form a CNF formula. Using the distributive law, we can transform ϕ_{move} into a conjunction of clauses.
  - The size of the formula is increased only by a constant factor.
- CNF $\rightarrow$ 3CNF

$$(x \vee y) \wedge (x_1 \vee x_2 \vee \ldots \vee x_t) \wedge \ldots$$

clauses with 1 or 2 literals

replication

clauses with more than 3 literals

split

$$(x \vee y \vee x) \qquad (x_1 \vee x_2 \vee c_1) \wedge (\neg c_1 \vee x_3 \vee c_2) \wedge \ldots \wedge (\neg c_{t-3} \vee x_{t-1} \vee x_t)$$

Search vs. Decision

- Definition: given a graph $G = (V, E)$, an **independent set** in G is a subset $V' \subseteq V$ such that for all $u,w \in V'$, $(u,w) \notin E$
- A problem:

  given G, find the largest independent set
- This is called a search problem
  - searching for *optimal* object of some type
  - comes up frequently

Search vs. Decision

- We want to talk about languages (or decision problems)
- Most search problems have a natural, related decision problem by adding a bound "k"; for example:
  - search problem: given G, find the *largest* independent set
  - decision problem: given (G, k), is there an independent set of size *at least* k

Ind. Set is NP-Complete

Theorem: the following language is NP-complete:

IS = {<G, k> | G has an IS of size $\geq$ k}

- Proof:
  - Part 1: IS $\in$ NP. Proof?
  - Part 2: IS is NP-hard.
    - reduce from 3-SAT

Ind. Set is NP-Complete

- We are reducing from the language:

  3SAT = { <φ> | φ is a 3-CNF formula that has a satisfying assignment }

  to the language:

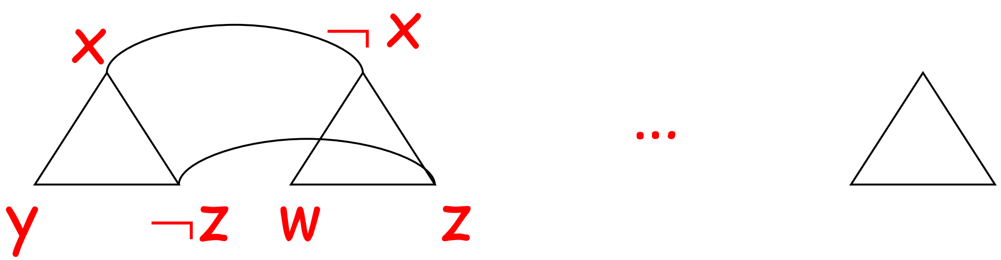
  IS = {<G, k> | G has an IS of size ≥ k}

Ind. Set is NP-Complete

The reduction f: given

$$\varphi = (x \vee y \vee \neg z) \wedge (\neg x \vee w \vee z) \wedge \ldots \wedge (\ldots)$$

we produce graph G_φ:

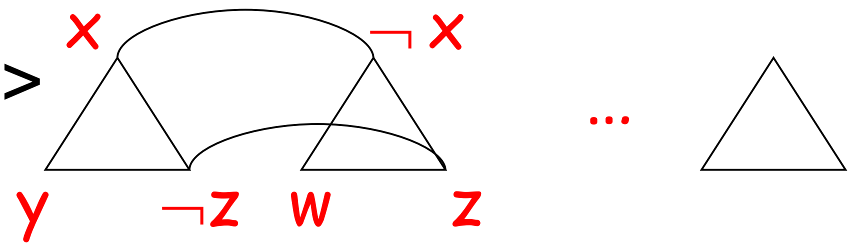


- one triangle for each of m clauses
- edge between every pair of contradictory literals
- set k = m

Ind. Set is NP-Complete

$$\varphi = (x \vee y \vee \neg z) \wedge (\neg x \vee w \vee z) \wedge \ldots \wedge (\ldots)$$

$f(\langle\varphi\rangle) = \langle G_\varphi, \text{\# clauses}\rangle$

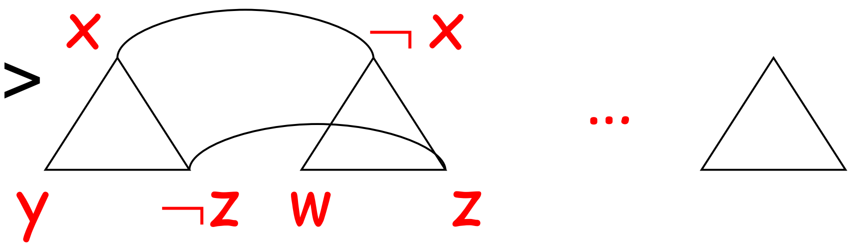


- Is f poly-time computable?
- YES maps to YES?
  - 1 true literal per clause is satisfying assignment A
  - choose corresponding vertices (1 per triangle)
  - IS, since no contradictory literals in A

Ind. Set is NP-Complete

$$\varphi = (x \vee y \vee \neg z) \wedge (\neg x \vee w \vee z) \wedge \ldots \wedge (\ldots)$$

$f(<\varphi>)=<G_\varphi, \#\text{ clauses}>$



- NO maps to NO?
  - IS can have at most 1 vertex per triangle
  - since IS, no contradictory vertices
  - can produce satisfying assignment by setting these literals to true

Vertex Cover

- Definition: given a graph $G = (V, E)$, a vertex cover in G is a subset $V' \subseteq V$ such that for all $(u,w) \in E$, $u \in V'$ or $w \in V'$
- A search problem:

  given G, find the smallest vertex cover

- corresponding language (decision problem):

  VC = {<G, k> | G has a VC of size ≤ k}.

Vertex Cover is NP-Complete

Theorem: the following language is NP-complete:

$VC = \{\langle G, k\rangle \mid G \text{ has a VC of size} \leq k\}$

- Proof:
  - Part 1: VC $\in$ NP. Proof?
  - Part 2: VC is NP-hard.
    - reduce from?

Vertex Cover is NP-Complete

- We are reducing from the language:

  IS = {<G, k> | G has an IS of size $\geq$ k}

  to the language:

  VC = {<G, k> | G has a VC of size $\leq$ k}

Vertex Cover is NP-Complete

- How are IS, VC related?

- Given a graph $G = (V, E)$ with n nodes
  - if $V' \subseteq V$ is an independent set of size k
  - then $V-V'$ is a vertex cover of size n-k
- Proof:
  - suppose not. Then there is some edge with neither endpoint in $V-V'$. But then both endpoints are in V'. contradiction.

Vertex Cover is NP-Complete

- How are IS, VC related?

- Given a graph G = (V, E) with n nodes
  - if V' $\subseteq$ V is a vertex cover of size k
  - then V-V' is an independent set of size n-k
- Proof:
  - suppose not. Then there is some edge with both endpoints in V-V'. But then neither endpoint is in V'. contradiction.

Vertex Cover is NP-Complete

The reduction:

- given an instance of IS: <G, k>, f produces the pair <G, n-k>

- f poly-time computable?
- YES maps to YES?
  - IS of size $\geq$ k in G $\Rightarrow$ VC of size $\leq$ n-k in G
- NO maps to NO?
  - VC of size $\leq$ n-k in G $\Rightarrow$ IS of size $\geq$ k in G

Clique

- Definition: given a graph $G = (V, E)$, a clique in G is a subset $V' \subseteq V$ such that for all $u, v \in V'$, $(u, v) \in E$
- A search problem:

  given G, find the largest clique

- corresponding language (decision problem):

  CLIQUE = {<G, k> | G has a clique of size $\geq$ k}.

Clique is NP-Complete

Theorem: the following language is NP-complete:

CLIQUE = {<G, k> | G has a clique of size $\geq$ k}

- Proof:
  - Part 1: CLIQUE $\in$ NP. Proof?
  - Part 2: CLIQUE is NP-hard.
    - reduce from?

Clique is NP-Complete

- We are reducing from the language:

$$IS = \{\langle G, k\rangle \mid G \text{ has an IS of size} \geq k\}$$

to the language:

$$CLIQUE = \{\langle G, k\rangle \mid G \text{ has a CLIQUE of size} \geq k\}.$$

Clique is NP-Complete

- How are IS, CLIQUE related?
- Given a graph $G = (V, E)$, define its complement G' $= (V, E' = \{(u,v) \mid (u,v) \notin E\})$
  - if $V' \subseteq V$ is an independent set in G of size k
  - then V' is a clique in G' of size k
- Proof:
  - suppose not. Then there are $u,v \in V'$ with $(u,v) \notin E'$ which implies $(u,v) \in E$. But then both endpoints of edge (u,v) in G are in V'. contradiction.

Clique is NP-Complete

- How are IS, CLIQUE related?
- Given a graph $G = (V, E)$, define its complement G' $= (V, E' = \{(u,v) \mid (u,v) \notin E\})$
  - if $V' \subseteq V$ is a clique in G' of size k
  - then V' is an independent set in G of size k
- Proof:
  - suppose not. Then there are $u,v \in V'$ with $(u,v) \in E$ which implies $(u,v) \notin E'$. But then there is no edge between u and v in G'. contradiction.

Clique is NP-Complete

The reduction:

– given an instance of IS: <G, k>, f produces the pair <G', k>

- f poly-time computable?
- YES maps to YES?
  – IS of size $\geq$ k in G $\Rightarrow$ CLIQUE of size $\geq$ k in G'
- NO maps to NO?
  – CLIQUE of size $\geq$ k in G' $\Rightarrow$ IS of size $\geq$ k in G

Hamilton Path

- Definition: given a directed graph G = (V, E), a Hamilton path in G is a directed path that touches every node exactly once.

- A language (decision problem):

  HAMPATH = {<G, s, t> | G has a Hamilton path from s to t}

HAMPATH is NP-Complete

Theorem: the following language is NP-complete:

HAMPATH = {<G, s, t> | G has a Hamilton path from s to t}

- Proof:
  - Part 1: HAMPATH $\in$ NP. Proof?
  - Part 2: HAMPATH is NP-hard.
    - reduce from?

HAMPATH is NP-Complete

- We are reducing from the language:

3SAT = { <φ> | φ is a 3-CNF formula that has a satisfying assignment }

to the language:

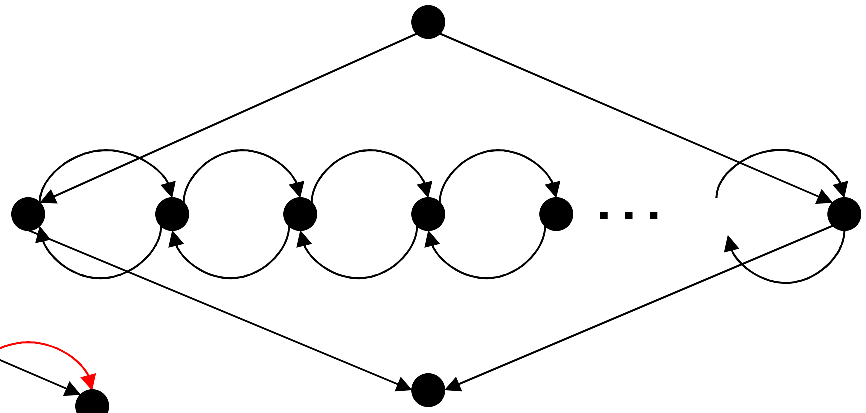
HAMPATH = {<G, s, t> | G has a Hamilton path from s to t}

HAMPATH is NP-Complete

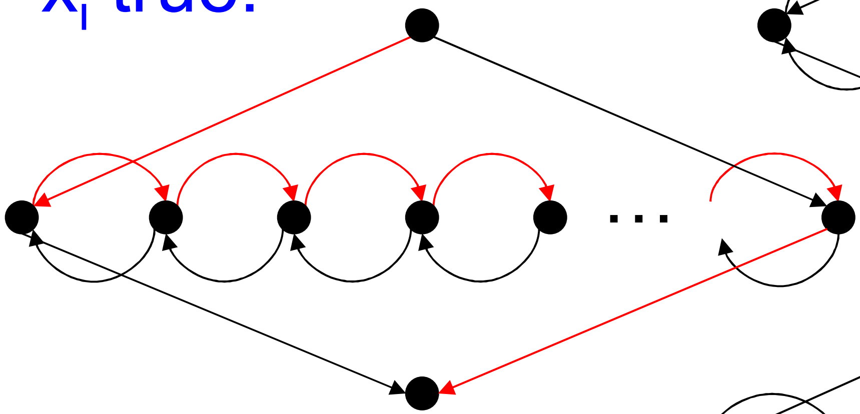
- We want to construct a graph from φ with the following properties:
  - a satisfying assignment to φ translates into a Hamilton Path from s to t
  - a Hamilton Path from s to t can be translated into a satisfying assignment for φ
- We will build the graph up from pieces called gadgets that "simulate" the clauses and variables of φ.

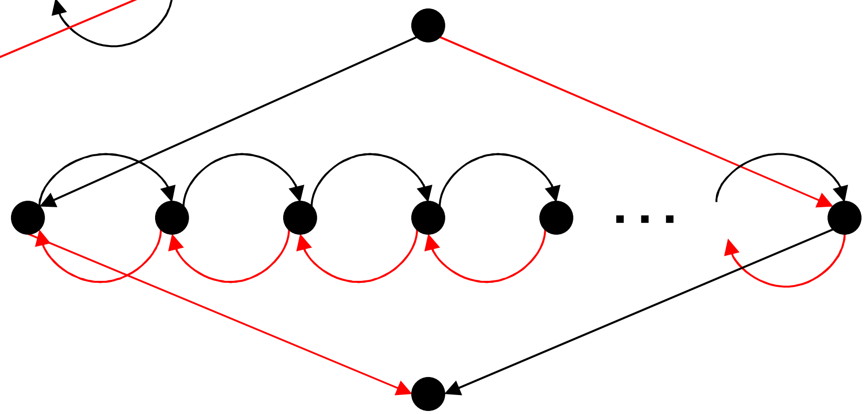
HAMPATH is NP-Complete

- The variable gadget (one for each x_i):



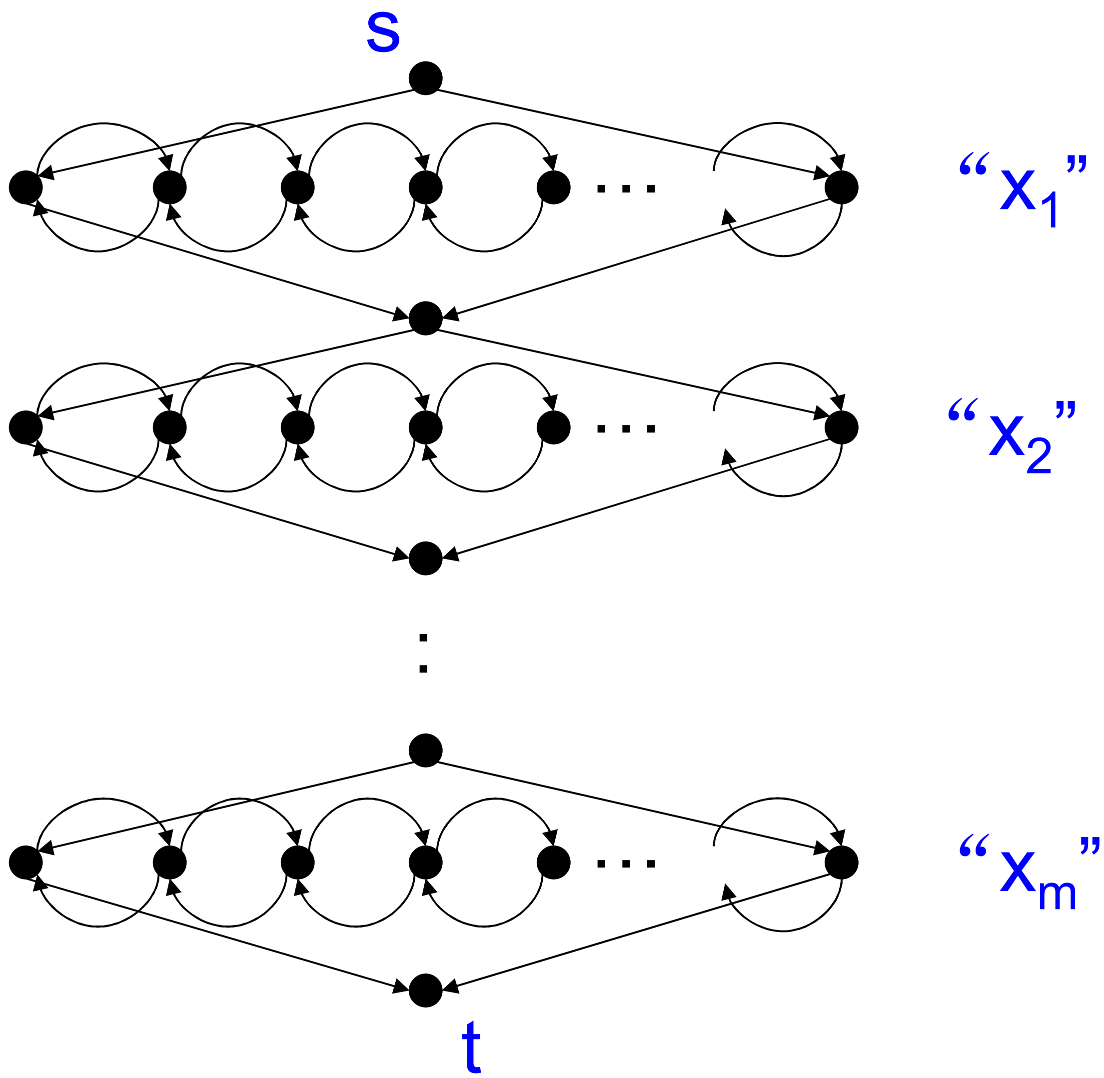
x_i true:





x_i false:

HAMPATH is NP-Complete



- path from s to t translates into a truth assignment to $x_1 \ldots x_m$
- why must the path be of this form?

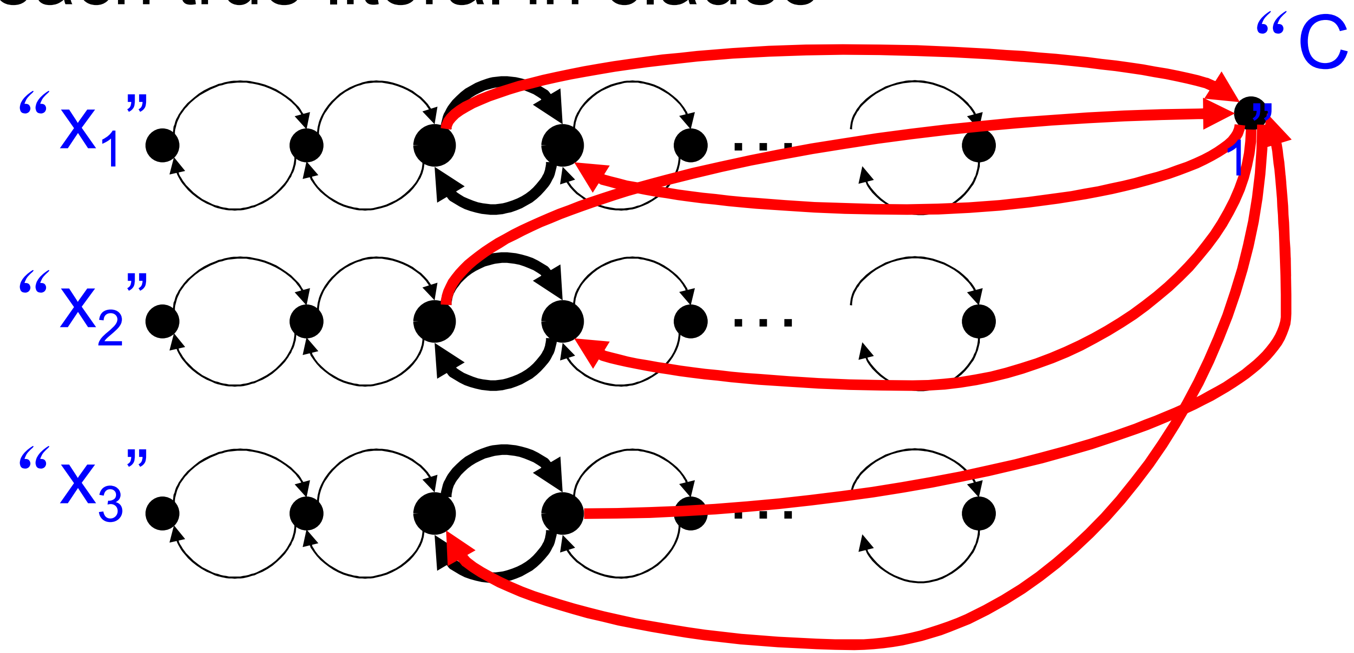
HAMPATH is NP-Complete

$\varphi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_4 \vee x_3) \wedge \ldots \wedge (\ldots)$

- How to ensure that all k clauses are satisfied?
- need to add nodes for "clauses"
  - can be visited in path if the clause is satisfied
  - if visited in path, implies the clause is satisfied by the assignment given by path through variable gadgets

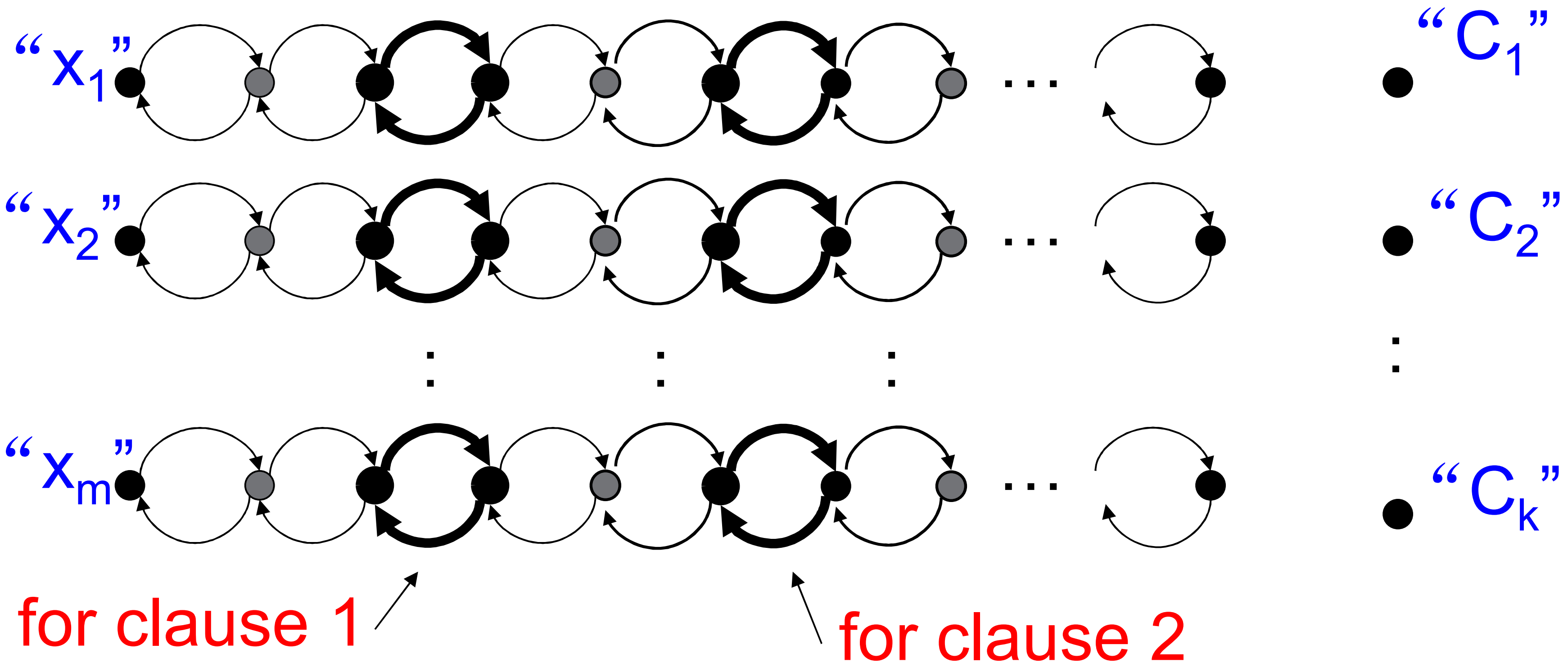
HAMPATH is NP-Complete

- $\varphi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_4 \vee x_3) \wedge \ldots \wedge (\ldots)$
- Clause gadget allows "detour" from "assignment path" for each true literal in clause



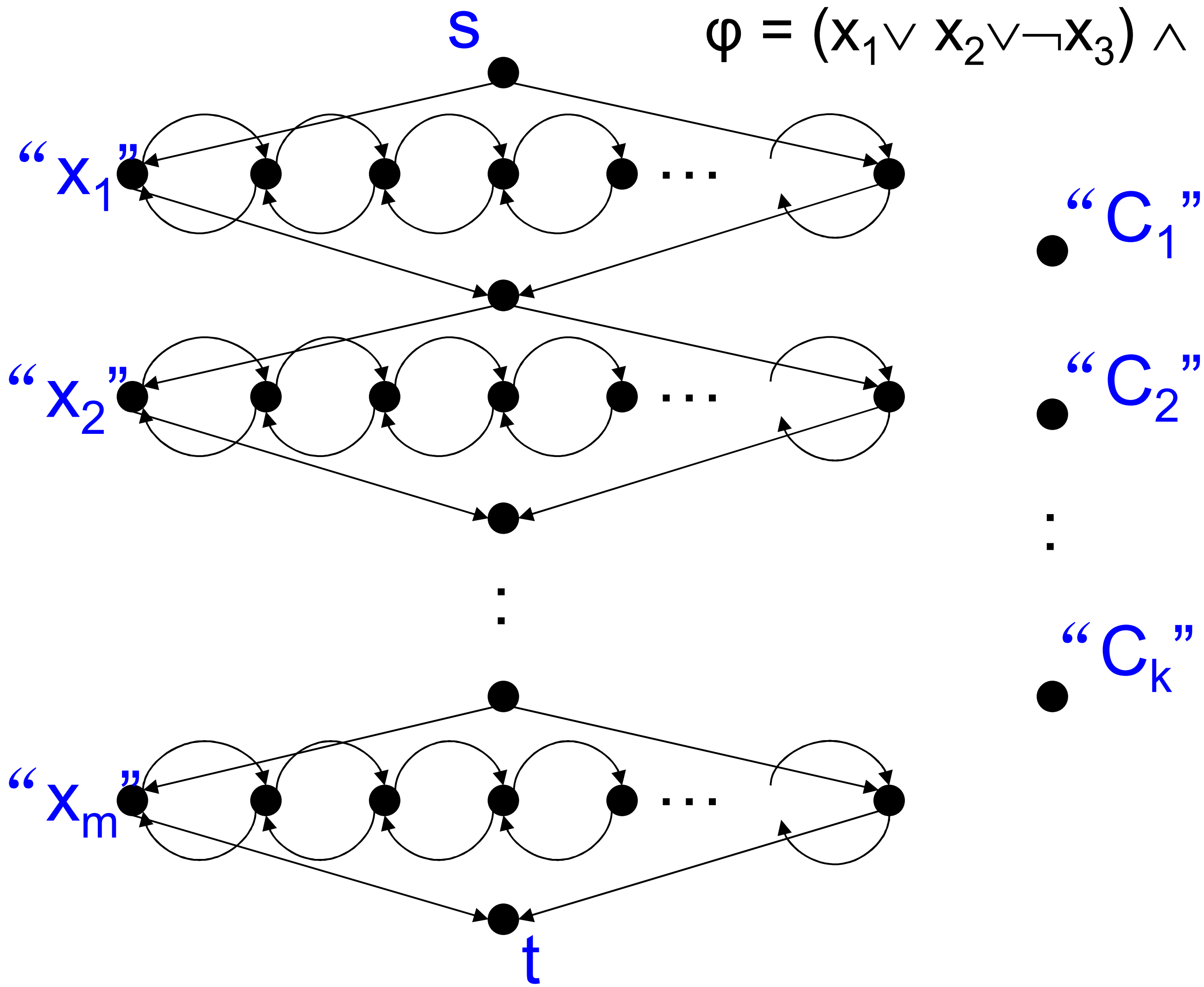
HAMPATH is NP-Complete

- One clause gadget for each of k clauses:



HAMPATH is NP-Complete

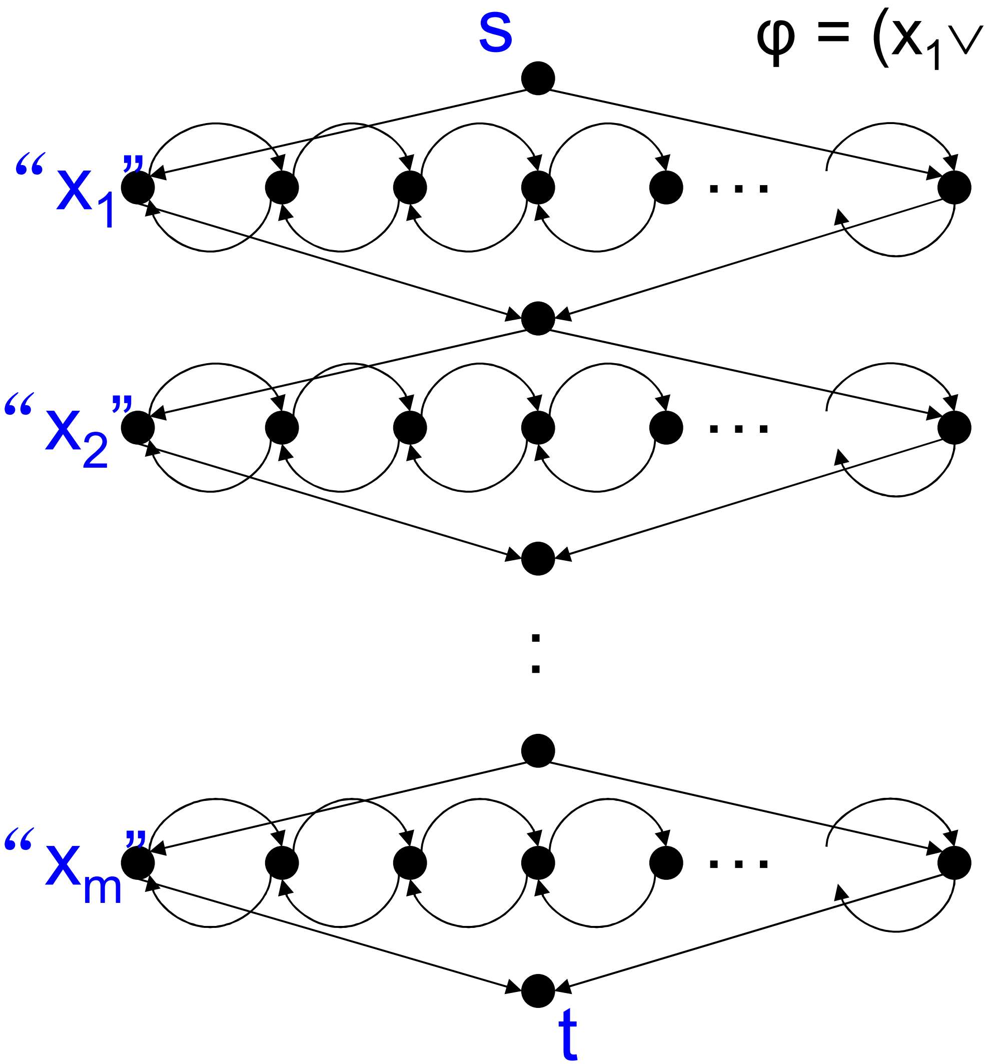
$\varphi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_4 \vee x_3) \wedge \ldots$

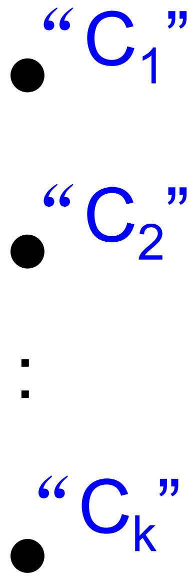


- $f(\varphi)$ is this graph (edges to/from clause nodes not pictured)
- f poly-time computable?
  - # nodes = O(km)

HAMPATH is NP-Complete

$\varphi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_4 \vee x_3) \wedge \ldots$





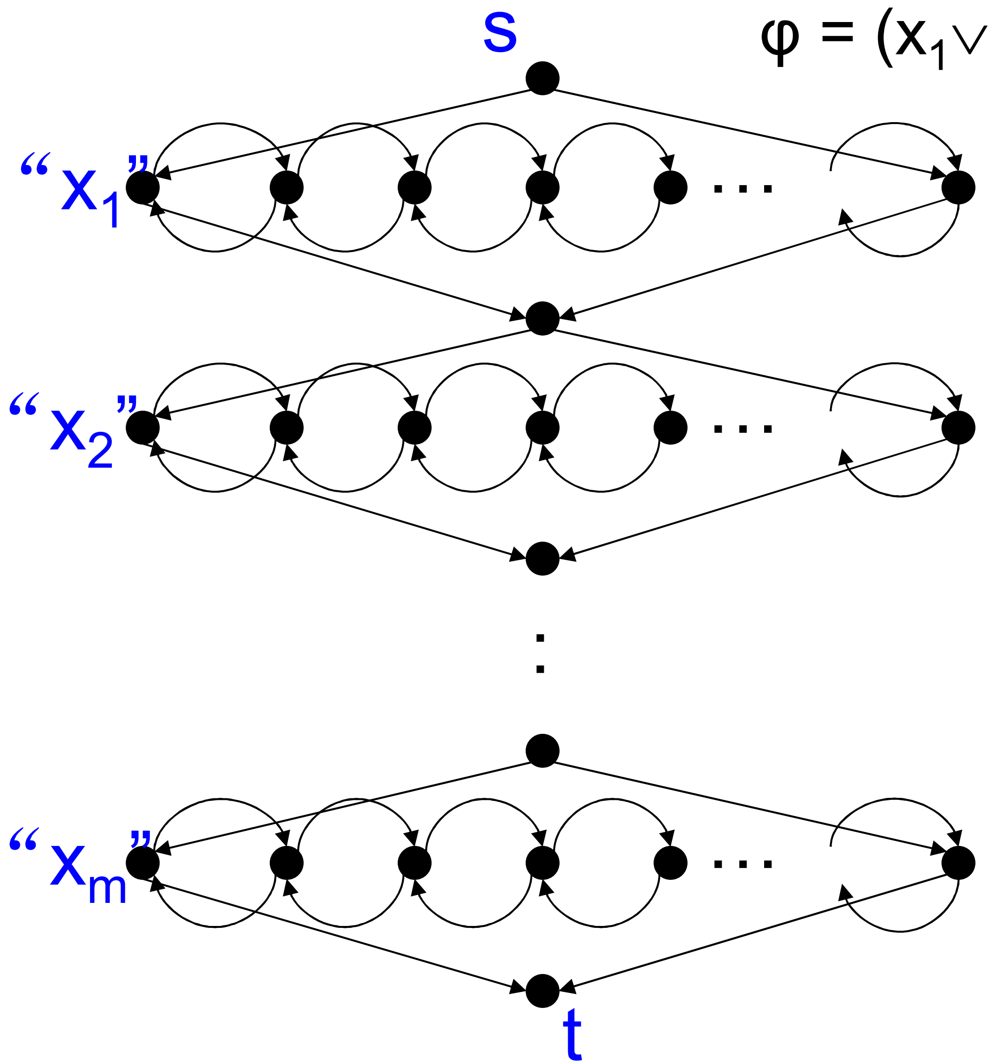
- YES maps to YES?

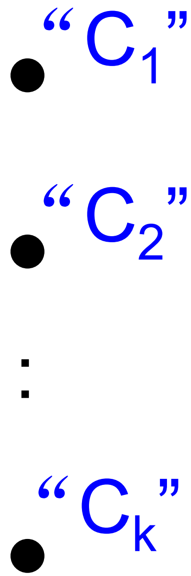
- first form path from satisfying assign.

- pick true literal in each clause and add detour

HAMPATH is NP-Complete

$\varphi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_4 \vee x_3) \wedge \ldots$



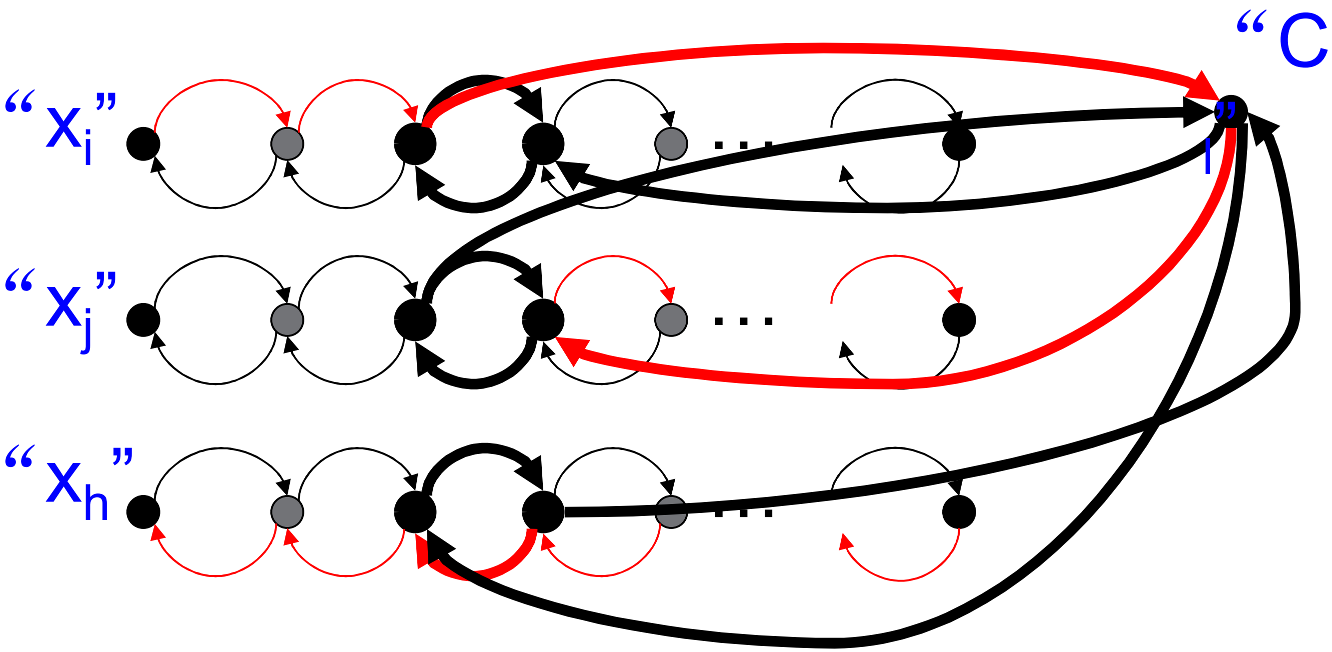


- NO maps to NO?
- try to translate path into satisfying assignment
- if path has "intended" form, OK.

HAMPATH is NP-Complete

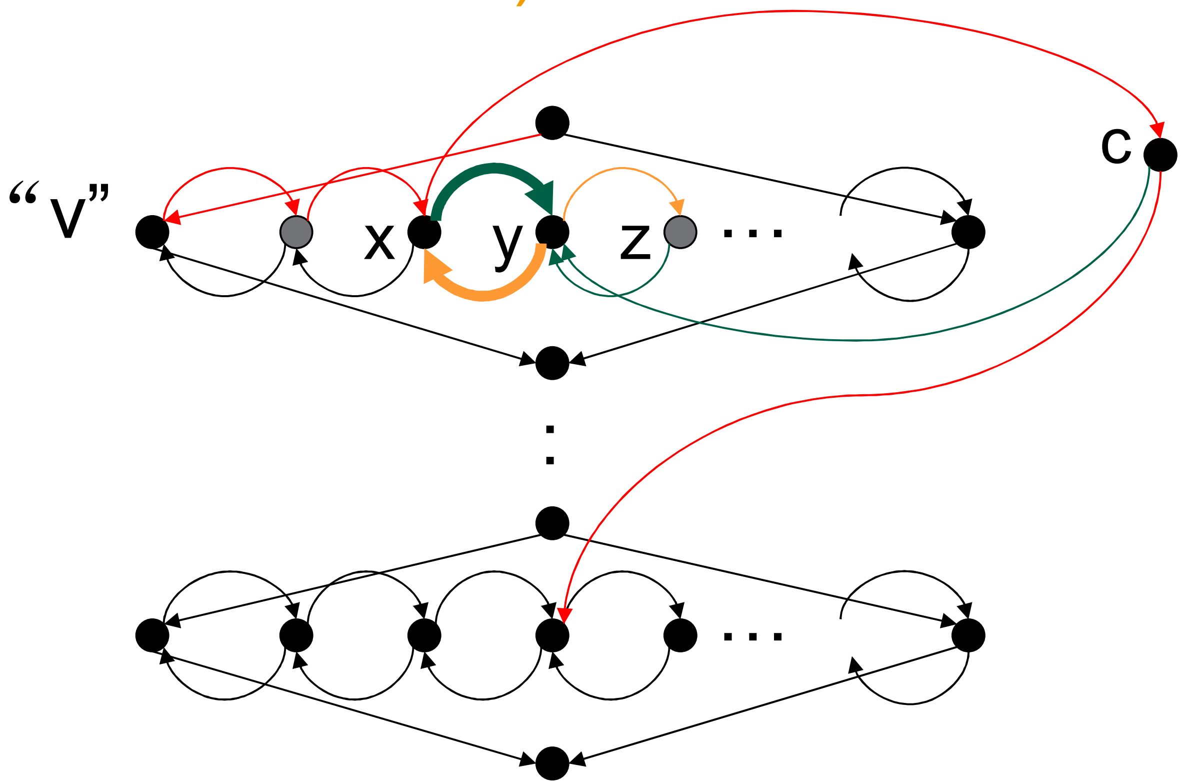
- What can go wrong?
  - path has “intended form” unless return from clause gadget to different variable gadget

we will argue that this cannot happen:



HAMPATH is NP-Complete

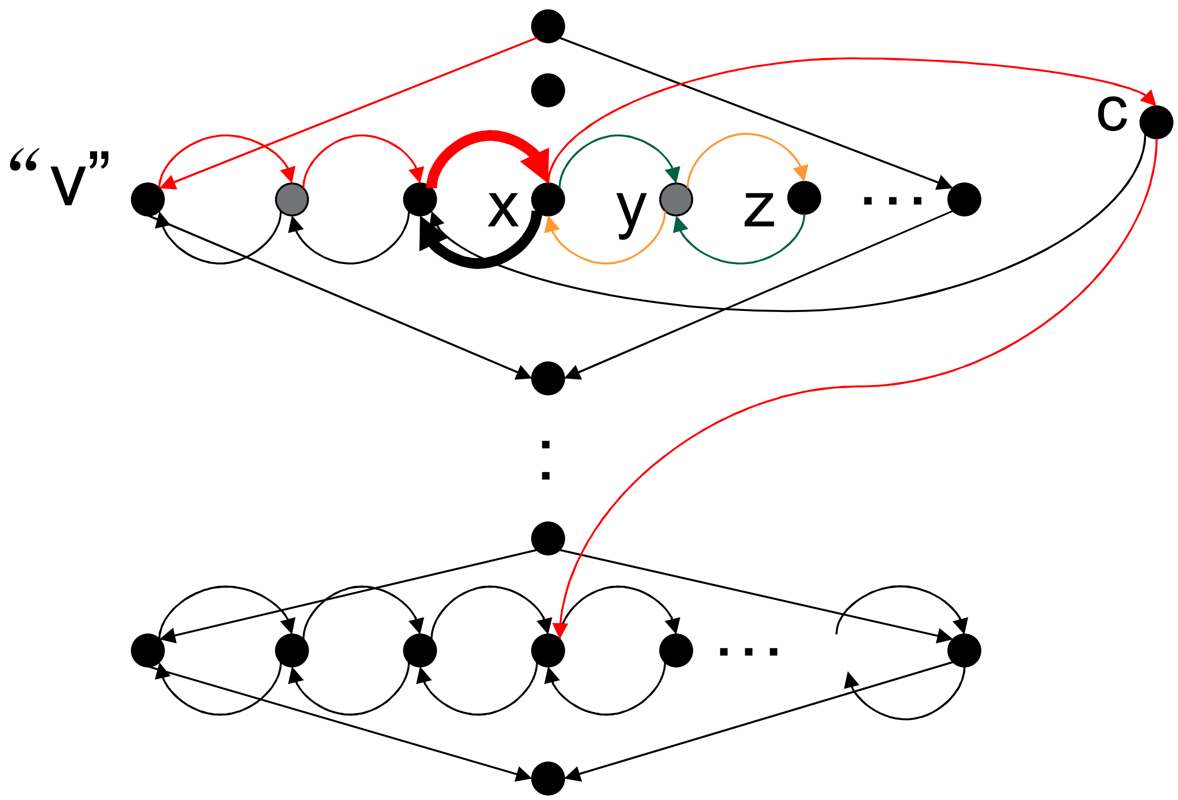
Case 1 (positive occurrence of v in clause):



- path must visit y
- must enter from x, z, or c
- must exit to z (x is taken)
- x, c are taken. can't happen

HAMPATH is NP-Complete

Case 2 (negative occurrence of v in clause):



- path must visit y
- must enter from x or z
- must exit to z (x is taken)
- x is taken. can’t happen

Undirected Hamilton Path

- HAMPATH refers to a directed graph.
- Is it easier on an undirected graph?

- A language (decision problem):

UHAMPATH = {<G, s, t> | undirected G has a Hamilton path from s to t}

UHAMPATH is NP-Complete

Theorem: the following language is NP-complete:

UHAMPATH = {<G, s, t> | undirected graph G has a Hamilton path from s to t}

- Proof:
  - Part 1: UHAMPATH $\in$ NP. Proof?
  - Part 2: UHAMPATH is NP-hard.
    - reduce from?

UHAMPATH is NP-Complete

- We are reducing from the language:

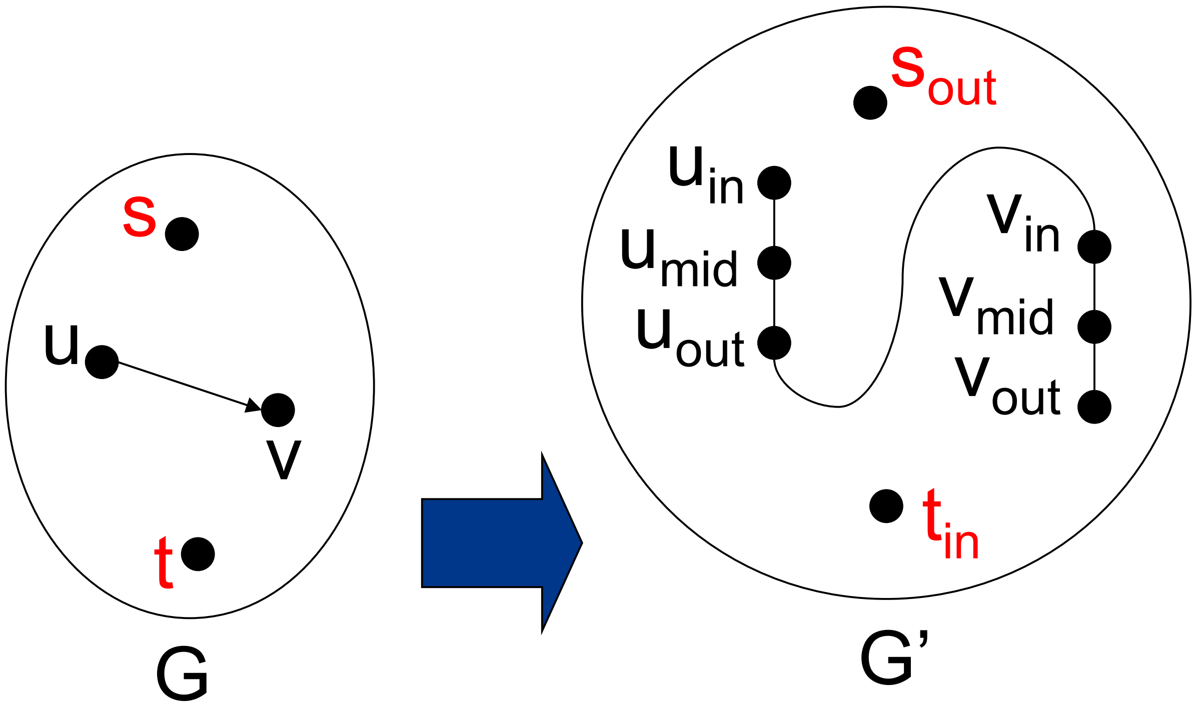
  HAMPATH = {<G, s, t> | directed graph G has a Hamilton path from s to t}

  to the language:

  UHAMPATH = {<G, s, t> | undirected graph G has a Hamilton path from s to t}

UHAMPATH is NP-Complete

- The reduction:



- replace each node with three (except s, t)
- (u_{in}, u_{mid})
- (u_{mid}, u_{out})
- (u_{out}, v_{in}) iff G has (u,v)

UHAMPATH is NP-Complete

- Does the reduction run in poly-time?

- YES maps to YES?
  - Hamilton path in G: $s, u_1, u_2, u_3, \ldots, u_k, t$
  - Hamilton path in G':

$$s_{out}, (u_1)_{in}, (u_1)_{mid}, (u_1)_{out}, (u_2)_{in}, (u_2)_{mid}, (u_2)_{out}, \ldots$$
$$(u_k)_{in}, (u_k)_{mid}, (u_k)_{out}, t_{in}$$

UHAMPATH is NP-Complete

- NO maps to NO?
  - Hamilton path in G':

$$s_{out}, v_1, v_2, v_3, v_4, v_5, v_6, \ldots, v_{k-2}, v_{k-1}, v_k, t_{in}$$

  - $v_1 = (u_{i1})_{in}$ for some i_1 (only edges to ins)
  - $v_2 = (u_{i1})_{mid}$ for some i_1 (only way to enter mid)
  - $v_3 = (u_{i1})_{out}$ for some i_1 (only way to exit mid)
  - $v_4 = (u_{i2})_{in}$ for some i_2 (only edges to ins)
  - ...
  - Hamilton path in G: $s, u_{i1}, u_{i2}, u_{i3}, \ldots, u_{ik}, t$

Undirected Hamilton Cycle

- Definition: given a undirected graph G = (V, E), a Hamilton cycle in G is a cycle in G that touches every node exactly once.
- Is finding one easier than finding a Hamilton path?
- A language (decision problem):

  UHAMCYCLE = {<G> | G has a Hamilton cycle}

UHAMCYCLE is NP-Complete

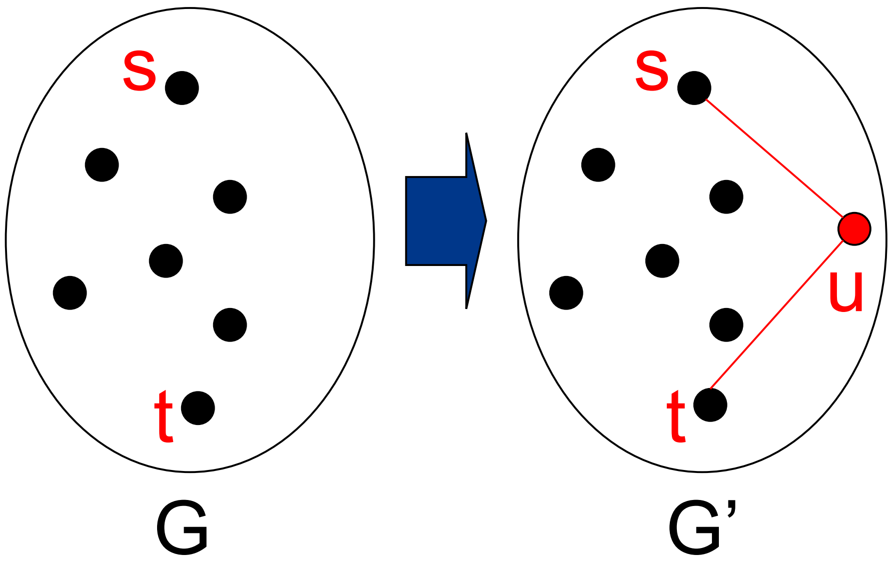
Theorem: the following language is NP-complete:

UHAMCYCLE = {<G> | G has a Hamilton cycle}

- Proof:
  - Part 1: UHAMCYCLE $\in$ NP. Proof?
  - Part 2: UHAMCYCLE is NP-hard.
    - reduce from?

UHAMCYCLE is NP-Complete

- The reduction (from UHAMPATH):



- H. path from s to t implies H. cycle in G'
- H. cycle in G' must visit u via red edges
- removing red edges gives H. path from s to t in G

Traveling Salesperson Problem

- Definition: given n cities $v_1, v_2, \ldots, v_n$ and inter-city distances d_{ij}, a TSP tour in G is a permutation π of {1…n}. The tour's length is $\Sigma_{i=1\ldots n}\, d_{\pi(i)\pi(i+1)}$ (where n+1 means 1).
- A search problem:

  given the $\{d_{ij}\}$, find the shortest TSP tour
- corresponding language (decision problem):

  TSP = $\{\langle\{d_{ij} : 1 \le i < j \le n\}, k\rangle \mid$ these cities have a TSP tour of length $\le k\}$

TSP is NP-Complete

Theorem: the following language is NP-complete:

$TSP = \{<\{d_{ij} \mid 1 \le i < j \le n\}, k> \mid$ these cities have a TSP tour of length $\le k\}$

- Proof:
  - Part 1: TSP $\in$ NP. Proof?
  - Part 2: TSP is NP-hard.
    - reduce from?

TSP is NP-Complete

- We are reducing from the language:

  UHAMCYCLE = {<G> | G has a Hamilton cycle}

  to the language:

  TSP = $\{\langle\{d_{ij} : 1 \le i < j \le n\}, k\rangle$ | these cities have a TSP tour of length $\le k\}$

TSP is NP-Complete

- The reduction:
  - given $G = (V, E)$ with n nodes

  produce:
  - n cities corresponding to the n nodes
  - $d_{uv} = 1$ if $(u, v) \in E$
  - $d_{uv} = 2$ if $(u, v) \notin E$
  - set $k = n$

TSP is NP-Complete

- YES maps to YES?
  - if G has a Hamilton cycle, then visiting cities in that order gives TSP tour of length n
- NO maps to NO?
  - if TSP tour of length ≤ n, it must have length exactly n.
  - all distances in tour are 1. Must be edges between every successive pair of cities in tour.

Subset Sum

- A language (decision problem):

  SUBSET-SUM = {<S = {a_1, a_2, a_3, …, a_k}, B> | there is a subset of S that sums to B}

- example:

  S = {1, 7, 28, 3, 2, 5, 9, 32, 41, 11, 8}, B = 30

  30 = 7 + 3 + 9 + 11. yes.

SUBSET-SUM is NP-Complete

Theorem: the following language is NP-complete:

SUBSET-SUM = {<S = {a_1, a_2, a_3, …, a_k}, B> | there is a subset of S that sums to B}

- Proof:
  - Part 1: SUBSET-SUM $\in$ NP. Proof?
  - Part 2: SUBSET-SUM is NP-hard.
    - reduce from?

SUBSET-SUM is NP-Complete

- We are reducing from the language:

  3SAT = { <φ> | φ is a 3-CNF formula that has a satisfying assignment }

  to the language:

  SUBSET-SUM = {<S = {a_1, a_2, a_3, …, a_k}, B> | there is a subset of S that sums to B}

SUBSET-SUM is NP-Complete

- $\varphi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_4 \vee x_3) \wedge \ldots \wedge (\ldots)$

- Need integers to play the role of truth assignments
- For each variable x_i include two integers in our set S:
  - x_i^{TRUE} and x_i^{FALSE}
- set B so that exactly one must be in sum

SUBSET-SUM is NP-Complete

$x_1^{TRUE} = 1\ 0\ 0\ 0\ \ldots\ 0$

$x_1^{FALSE} = 1\ 0\ 0\ 0\ \ldots\ 0$

$x_2^{TRUE} = 0\ 1\ 0\ 0\ \ldots\ 0$

$x_2^{FALSE} = 0\ 1\ 0\ 0\ \ldots\ 0$

…

$x_m^{TRUE} = 0\ 0\ 0\ 0\ \ldots\ 1$

$x_m^{FALSE} = 0\ 0\ 0\ 0\ \ldots 1$

$B = 1\ 1\ 1\ 1\ \ldots\ 1$

- every choice of one from each $(x_i^{TRUE}, x_i^{FALSE})$ pair sums to B
- every subset that sums to B must choose one from each $(x_i^{TRUE}, x_i^{FALSE})$ pair

SUBSET-SUM is NP-Complete

- $\varphi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_4 \vee x_3) \wedge \ldots \wedge (\ldots)$

- Need to force subset to "choose" at least one true literal from each clause
- Idea:
  - add more digits
  - one digit for each clause
  - set B to force each clause to be satisfied.

SUBSET-SUM is NP-Complete

– $\varphi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_4 \vee x_3) \wedge \ldots \wedge (\ldots)$

		clause 1	clause 2	clause 3	…	clause k
x_1^{TRUE}	= 1 0 0 0 … 0	1	0		…	
x_1^{FALSE}	= 1 0 0 0 … 0	0	1			
x_2^{TRUE}	= 0 1 0 0 … 0	1	0			
x_2^{FALSE}	= 0 1 0 0 … 0	0	0			
x_3^{TRUE}	= 0 0 1 0 … 0	0	1			
x_3^{FALSE}	= 0 0 1 0 … 0	1	0		…	
…		:				
B	= 1 1 1 1 … 1	?	?	?		?

SUBSET-SUM is NP-Complete

– B = 1 1 1 1 … 1 ? ? ? … ?

– if clause i is satisfied, sum might be 1, 2, or 3 in corresponding column.

– want ? to "mean" $\geq$ 1

– solution: set ? = 3

– add two "filler" elements for each clause i:

– FILL1_i = 0 0 0 0 ... 0 0 ... 0 1 0 ... 0

– FILL2_i = 0 0 0 0 ... 0 0 ... 0 1 0 ... 0

column for clause i

SUBSET-SUM is NP-Complete

- Reduction: m variables, k clauses
  - for each variable x_i:
    - x_i^{TRUE} has ones in positions i and {m+j | clause j includes literal x_i}
    - x_i^{FALSE} has ones in positions i and {m+j | clause j includes literal $\neg x_i$}
  - for each clause j:
    - $FILL1_j$ and $FILL2_j$ have one in position m+j
  - bound B has 1 in positions 1…m, and 3 in positions m+1…m+k

SUBSET-SUM is NP-Complete

- Reduction computable in poly-time?
- YES maps to YES?
  - choose one from each (x_i^{TRUE}, x_i^{FALSE}) pair corresponding to a satisfying assignment
  - choose 0, 1, or 2 of filler elements for each clause i depending on whether it has 3, 2, or 1 true literals
  - first m digits add to 1; last k digits add to 3

SUBSET-SUM is NP-Complete

- NO maps to NO?
  - first m digits of B force subset to choose exactly one from each (x_i^{TRUE}, x_i^{FALSE}) pair
  - last k digits of B require at least one true literal per clause, since can only sum to 2 using filler elements
  - resulting assignment must satisfy φ

A Scheduling Problem

- each of n jobs has
  - processing time t_i
  - deadline d_i
  - profit p_i
- objective: schedule jobs to maximize profit
- schedule: $s_1, s_2, s_3, \ldots, s_n$
  - no overlaps: $[s_i, s_i + t_i]$ disjoint from $[s_j, s_j + t_j]$ $\forall i \neq j$
- profit: sum of p_i for all i such that $s_i + t_i \leq d_i$

A Scheduling Problem

Theorem: the following language is NP-complete:

SCHEDULE = {<$(t_1, d_1, p_1), (t_2, d_2, p_2), \ldots, (t_n, d_n, p_n), k$ | there is a schedule for these jobs with profit $\geq$ k}

- Proof:
  - Part 1: SCHEDULE $\in$ NP. Proof?
  - Part 2: SCHEDULE is NP-hard.
    - reduce from?

SCHEDLE is NP-Complete

- We are reducing from the language:

$$\text{SUBSET-SUM} = \{\langle S = \{a_1, a_2, a_3, \ldots, a_n\}, B\rangle \mid \text{there is a subset of } S \text{ that sums to } B\}$$

to the language:

$$\text{SCHEDULE} = \{\langle (t_1, d_1, p_1), (t_2, d_2, p_2), \ldots, (t_n, d_n, p_n), k\rangle \mid \text{there is a schedule for these jobs with profit} \geq k\}$$

SCHEDULE is NP-Complete

- Given instance

$$S = \{a_1, a_2, a_3, \ldots, a_n\}, B$$

- produce the instance

$(t_1, d_1, p_1) = (a_1, B, a_1)$

$(t_2, d_2, p_2) = (a_2, B, a_2)$

…

$(t_n, d_n, p_n) = (a_n, B, a_n)$, $k = B$

SCHEDULE is NP-Complete

- Does the reduction run in polynomial time?

$<(t_1=a_1, d_1=B, p_1=a_1),$

$(t_2=a_2, d_2=B, p_2=a_2), \ldots,$

$(t_n=a_n, d_n=B, p_n=a_n),$
$k=B>$

- YES maps to YES
  - $a_{i_1}+a_{i_2}+a_{i_3}+\ldots+a_{i_m}=B$
  - schedule:

    $s_{i_1}=0,\ s_{i_2}=s_{i_1}+a_{i_1},\ s_{i_3}=s_{i_2}+a_{i_2},\ \ldots,\ s_{i_m}=s_{i_{m-1}}+a_{i_{m-1}}$

    (rest don't matter)
  - profit = $a_{i_1}+a_{i_2}+a_{i_3}+\ldots+a_{i_m}= B = k$

SCHEDULE is NP-Complete

- NO maps to NO
  - schedule:

    $s_1, s_2, s_3, \ldots, s_n$

    with profit ≥ k

    $<(t_1=a_1,\ d_1=B,\ p_1=a_1),$

    $(t_2=a_2,\ d_2=B,\ p_2=a_2),\ \ldots,$

    $(t_n=a_n,\ d_n=B,\ p_n=a_n),$

    $k=B>$
  - profit: sum of p_i for all i such that $s_i + t_i \leq d_i$
  - sum of a_i for all i such that $s_i + a_i \leq B$
  - profit must be exactly B