Pre-lecture brain teaser

A boolean expression is in disjunctive normal form if it consists of the union of clauses where each clause is composed of the intersection of literals. For example:

$$(\overline{x_1} \wedge x_3 \wedge x_4) \vee (x_2 \wedge \overline{x_3} \wedge x_4) \tag{1}$$

Imagine we have a problem: DNF-SAT, where given a DNF formula, we want to know if there is a satisfying assignment. We know two things:

- Finding a satisfying assignment for a DNF formula takes polynomial time.
- We can rewrite any CNF formula as a DNF formula.

Hence I do the smart thing and say since CNF-SAT $\leq_P$ DNF-SAT, then CNF-SAT $\in$ NP.

Am I correct?

CS/ECE-374: Lecture 26 - NP-Complete reductions

Lecturer: Nickvash Kani
Chat moderator: Samir Khan

April 27, 2021

University of Illinois at Urbana-Champaign

Pre-lecture brain teaser

SAT: CNF: $(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3) \wedge \dots$

A boolean expression is in disjunctive normal form if it consists of the union of clauses where each clause is composed of the intersection of literals. For example:

satisfying $x_1 = 0$ $x_3 = 1$ $x_2 = X$ $x_4 = 1$

$$(\overline{x_1} \wedge x_3 \wedge x_4) \vee (x_2 \wedge \overline{x_3} \wedge x_4) \quad (2)$$

Imagine we have a problem: DNF-SAT, where given a DNF formula, we want to know if there is a satisfying assignment. We know two things:

- Finding a satisfying assignment for a DNF formula takes polynomial time.
- We can rewrite any CNF formula as a DNF formula.

Construction of reduction table $O(2^n)$

error

Hence I do the smart thing and say since CNF-SAT $\leq_P$ DNF-SAT, then CNF-SAT $\in$ NP.

CNF $f(x) = (x_1 \vee x_2)$

DNF $f(x) = x_1 x_2 + \overline{x_1} x_2 + x_1 \overline{x_2}$

Am I correct?

Today

NP-Completeness of two problems:

- Hamiltonian Cycle
- 3-Color

Important: understanding the problems and that they are hard.

$X \in NP$

$X \le 3SAT \le P$

Proofs and reductions will be sketchy and mainly to give a flavor

NP-Complete → NP-hard (harder than all NP problems)

$\in NP$ → Write poly time certifier

$C(s, t)$

P instance

solution

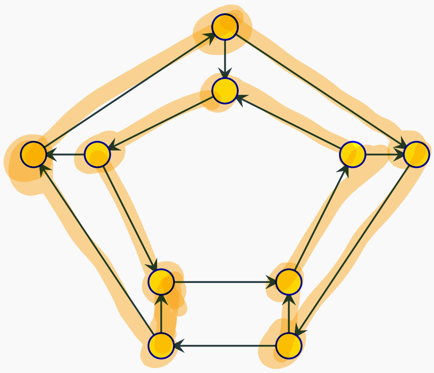
Reduction from 3SAT to Hamiltonian Cycle

Directed Hamiltonian Cycle

Input Given a directed graph $G = (V, E)$ with n vertices

Goal Does G have a Hamiltonian cycle?

- 2- A Hamiltonian cycle is a cycle in the graph that visits every vertex in G exactly once

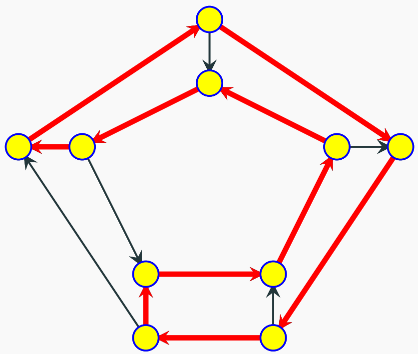


Directed Hamiltonian Cycle

Input Given a directed graph $G = (V, E)$ with n vertices

Goal Does G have a Hamiltonian cycle?

- 2- A Hamiltonian cycle is a cycle in the graph that visits every vertex in G exactly once



$C(G(V,E), V[\,])$

for (v in $V[\,]$)

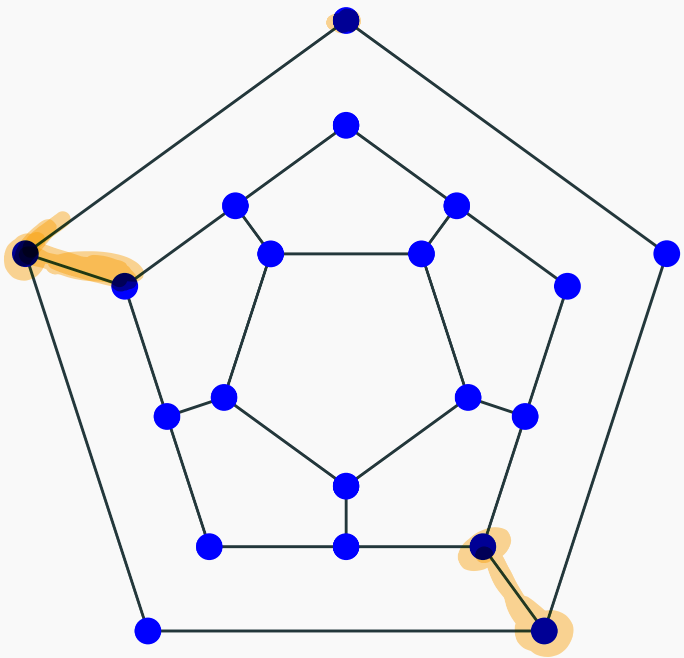
check if $(v(n-1), v(n)) \in E$

mark $V[n]$ ++

Proves $HC \in NP$

Want to prove $HC \in NP$-hard

Is the following graph Hamiltonianan?



a Yes.

b No.

Directed Hamiltonian Cycle is NP-Complete

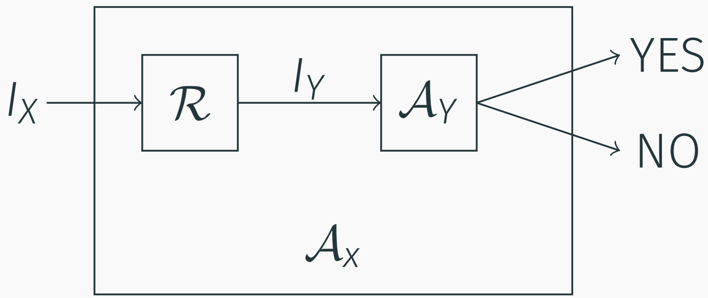
- Directed Hamiltonian Cycle is in *NP*: exercise
- Hardness: We will show

  3-SAT $\leq_P$ Directed Hamiltonian Cycle

NP-hard

Directed Hamiltonian Cycle is NP-Complete

- Directed Hamiltonian Cycle is in *NP*: exercise
- Hardness: We will show
  3-SAT $\leq_P$ Directed Hamiltonian Cycle



A_Y = $ORAC_{HC}$

A_X = $Decider_{SAT}$

I_X = φ (3CNF)

I_Y = $G(V,E)$

have HC if φ is satisfiable
no HC if φ is not satisfiable

Reduction

Given 3-SAT formula φ create a graph G_φ such that

- G_φ has a Hamiltonian cycle if and only if φ is satisfiable
- G_φ should be constructible from φ by a polynomial time algorithm $\mathcal{A}$

Notation: φ has n variables $x_1, x_2, \ldots, x_n$ and m clauses $C_1, C_2, \ldots, C_m$.

Reduction: First Ideas

- Viewing SAT: Assign values to n variables, and each clauses has 3 ways in which it can be satisfied. $(x_1 \vee x_2 \vee x_3)$
- Construct graph with 2^n Hamiltonian cycles, where each cycle corresponds to some boolean assignment.
- Then add more graph structure to encode constraints on assignments imposed by the clauses.

Reduction: Encoding idea I

Need to create a graph from any arbitrary boolean assignment.

Consider the expression:

always satisfied

$$f(x_1) = 1 \tag{3}$$

1 literal

Reduction: Encoding idea I

Need to create a graph from any arbitrary boolean assignment.

Consider the expression:

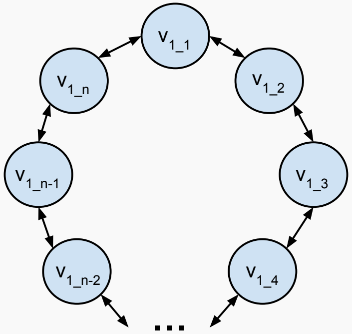
$$f(x_1) = 1 \tag{3}$$

2 ways to SAT f(x)

$x_1 = 0$

$x_1 = 1$

We create a cyclic graph that always has a hamiltonian:

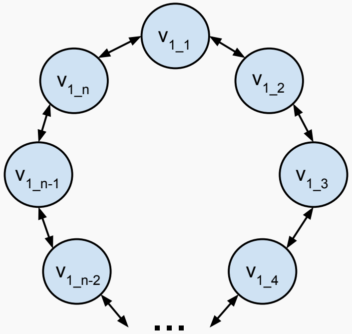


Reduction: Encoding idea I

Need to create a graph from any arbitrary boolean assignment. Consider the expression:

$$f(x_1) = 1 \tag{3}$$

We create a cyclic graph that always has a hamiltonian:



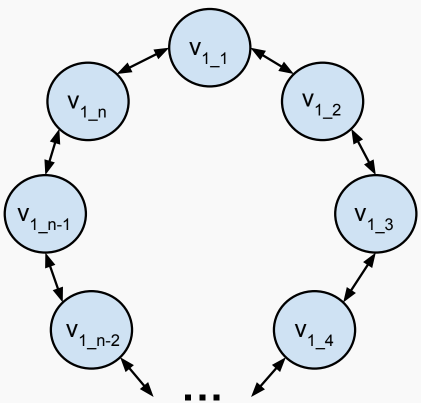
But how do we encode the variable?

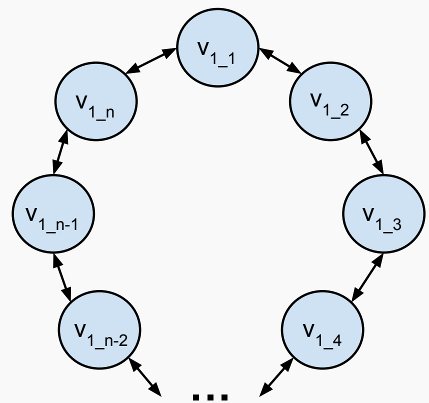
Reduction: Encoding idea I

Need to create a graph from any arbitrary boolean assignment. Consider the expression:

$$f(x_1) = 1 \tag{4}$$

Maybe we can encode the variable x_1 in terms of the cycle direction:





Reduction: Encoding idea I

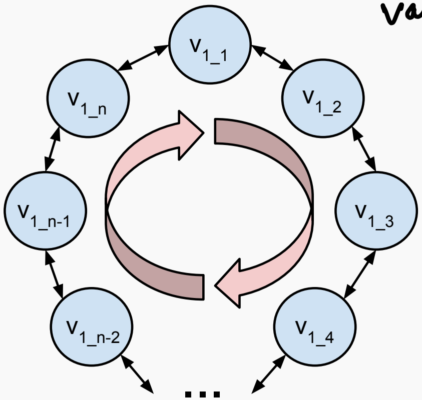
Need to create a graph from any arbitrary boolean assignment. Consider the expression:

$$f(x_1) = 1 \tag{4}$$

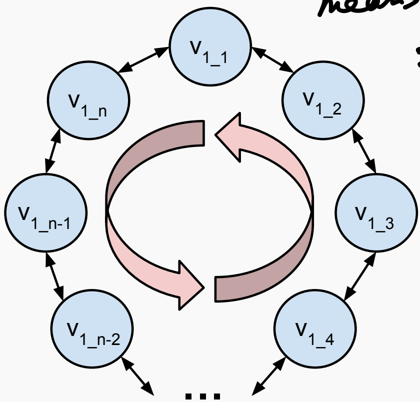
Maybe we can encode the variable x_1 in terms of the cycle direction:

Clockwise means variable = 1

Counter clockwise means variable = 0



If $x_1 = 1$



If $x_1 = 0$

Reduction: Encoding idea II

How do we encode multiple variables?

$[x_1, x_2] = \begin{array}{c} 00 \\ 01 \\ 10 \\ 11 \end{array}$

$$f(x_1, x_2) = 1 \tag{5}$$

Maybe two circles? Now we need to connect them so that we have a single hamiltonian path

Reduction: Encoding idea II

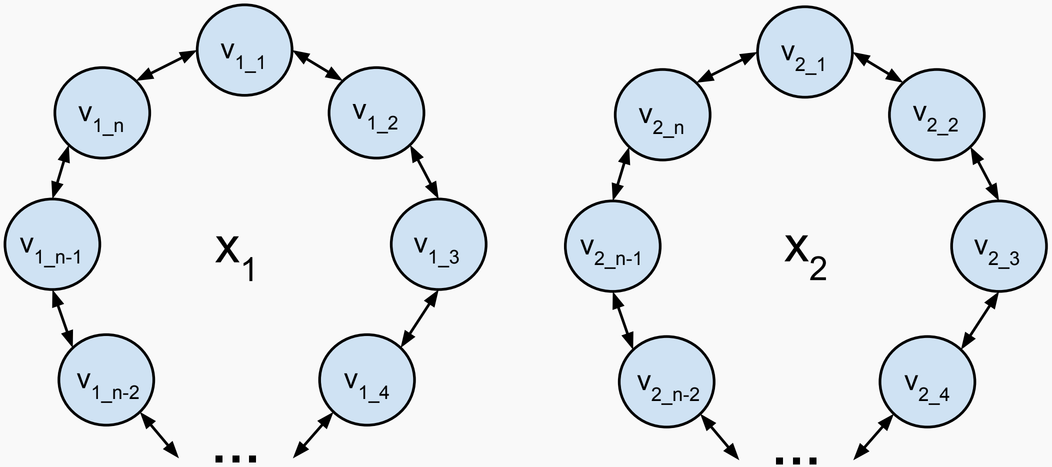
How do we encode multiple variables?

$$f(x_1, x_2) = 1 \tag{5}$$

Maybe two circles? Now we need to connect them so that we have a single hamiltonian path

$x_1 = 0$

$x_2 = 1$

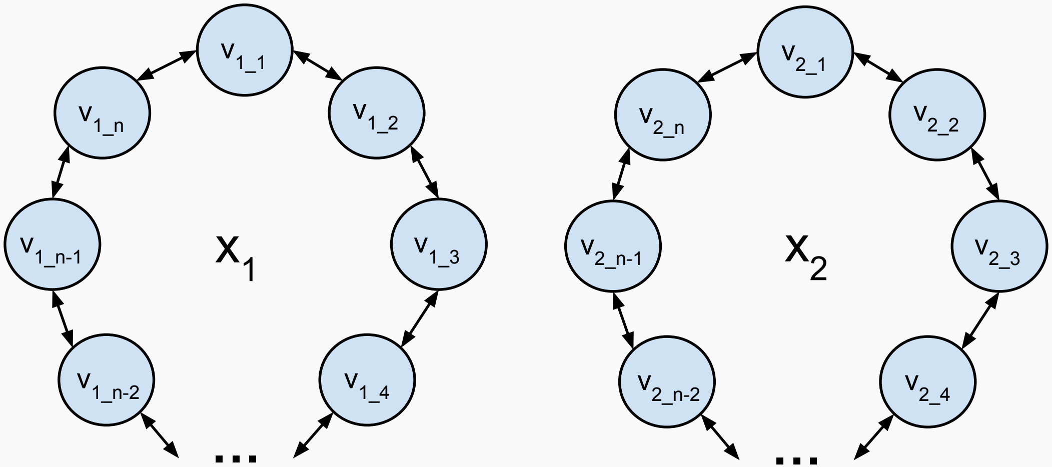


Reduction: Encoding idea II

How do we encode multiple variables?

$$f(x_1, x_2) = 1 \tag{6}$$

Now we need to connect them so that we have a single hamiltonian path



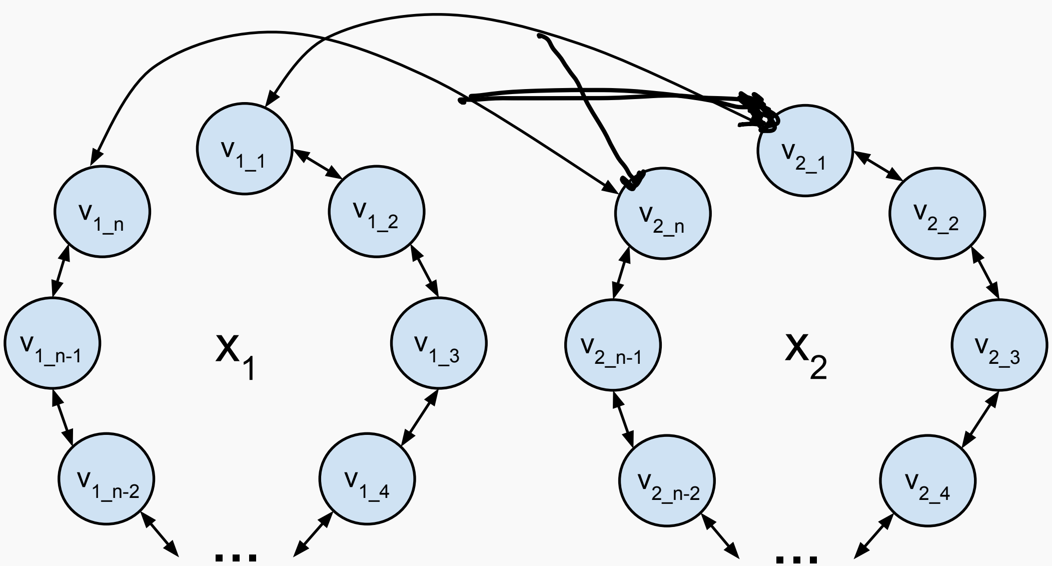
Reduction: Encoding idea II

How do we encode multiple variables?

$$f(x_1, x_2) = 1 \tag{6}$$

Now we need to connect them so that we have a single hamiltonian path

$x_1 x_2 = 01$ $\cancel{x}\, x_2 = 11$

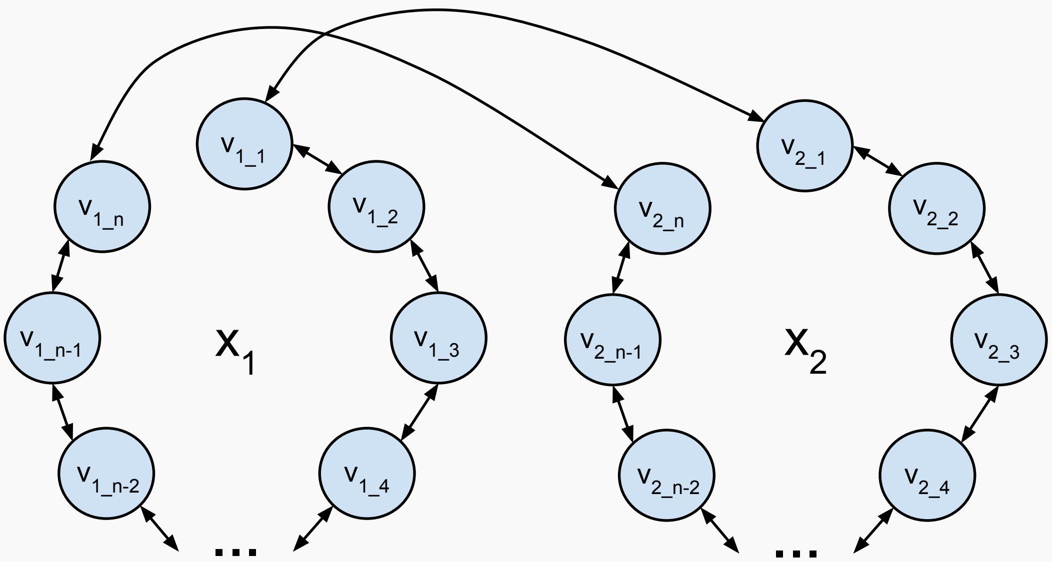


Reduction: Encoding idea II

How do we encode multiple variables?

$$f(x_1, x_2) = 1 \tag{7}$$

Would be nice to have a single start/stop node.

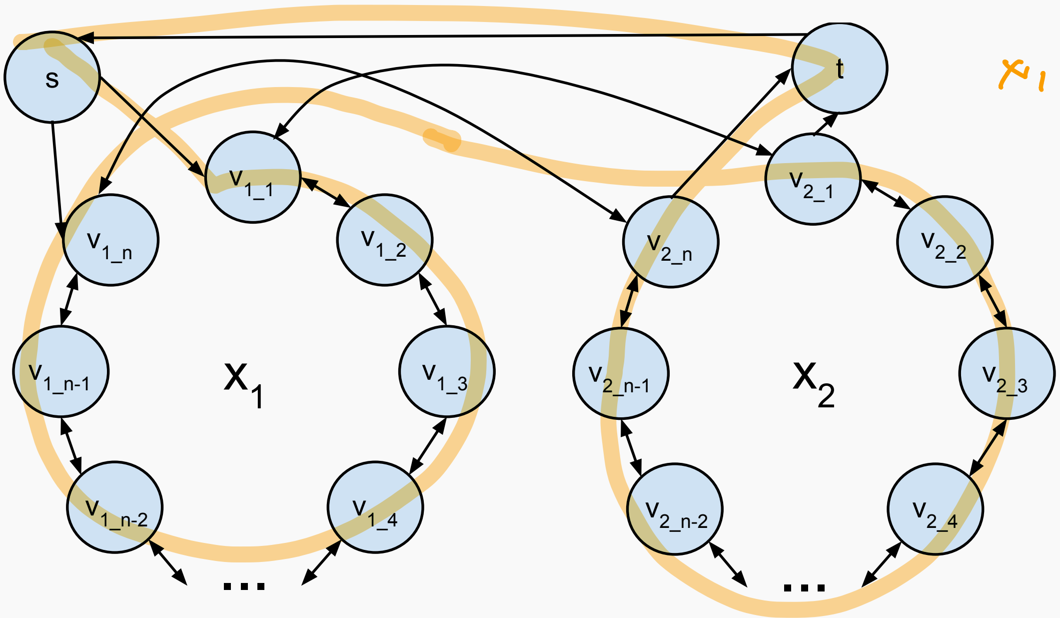


Reduction: Encoding idea II

How do we encode multiple variables?

$$f(x_1, x_2) = 1 \tag{7}$$

Would be nice to have a single start/stop node.



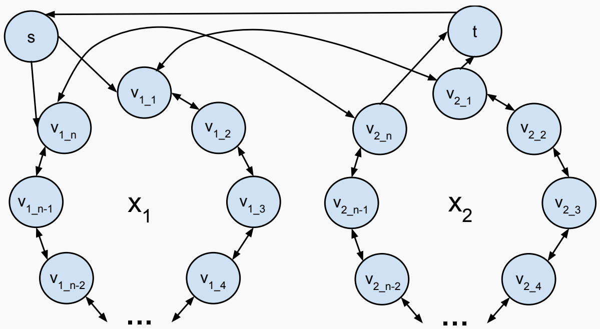
$x_1 x_2 = 01$

Reduction: Encoding idea II

How do we encode multiple variables?

$$f(x_1, x_2) = 1 \tag{8}$$

Getting a bit messy. Let's reorganize:

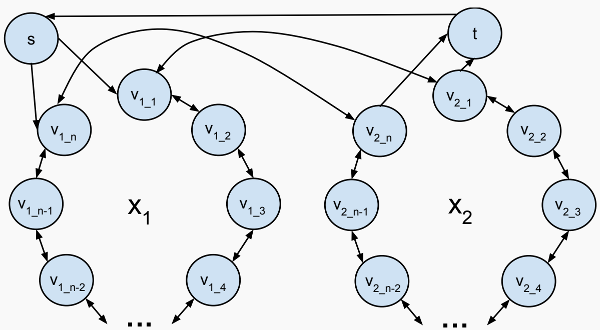


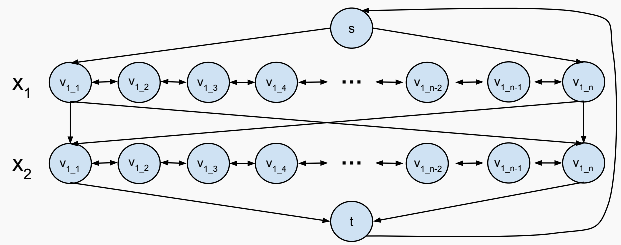
Reduction: Encoding idea II

How do we encode multiple variables?

$$f(x_1, x_2) = 1 \tag{8}$$

Getting a bit messy. Let's reorganize:





Reduction: Encoding idea II

How do we encode multiple variables?

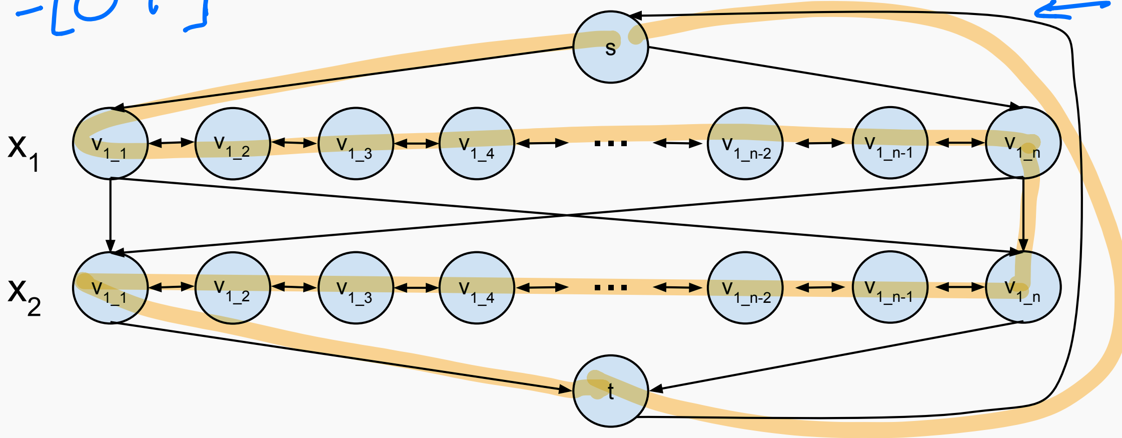
$$f(x_1, x_2) = 1 \tag{9}$$

This is how we encode variable assignments in a variable loop!

$x_1, x_2 = [0\ 1]$

$\rightarrow 0$

$\leftarrow 1$



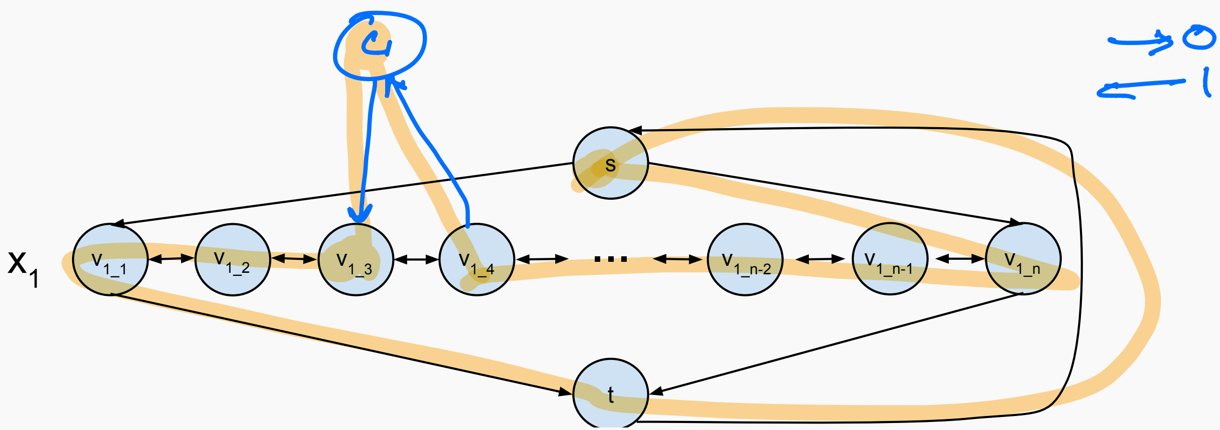
has 2^n HC

Reduction: Encoding idea III

How do we handle clauses?

$$f(x_1) = (x_1) \quad c_1 \qquad x_1 = 1 \tag{10}$$

Lets go back to our one variable graph:



Reduction: Encoding idea III

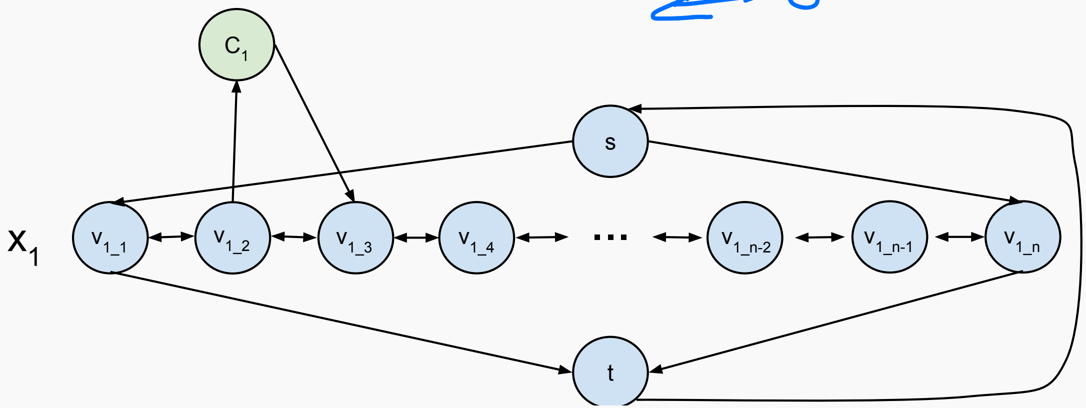
How do we handle clauses?

$$f(x_1) = x_1 \tag{11}$$

Add node for clause:

$\longrightarrow 1$

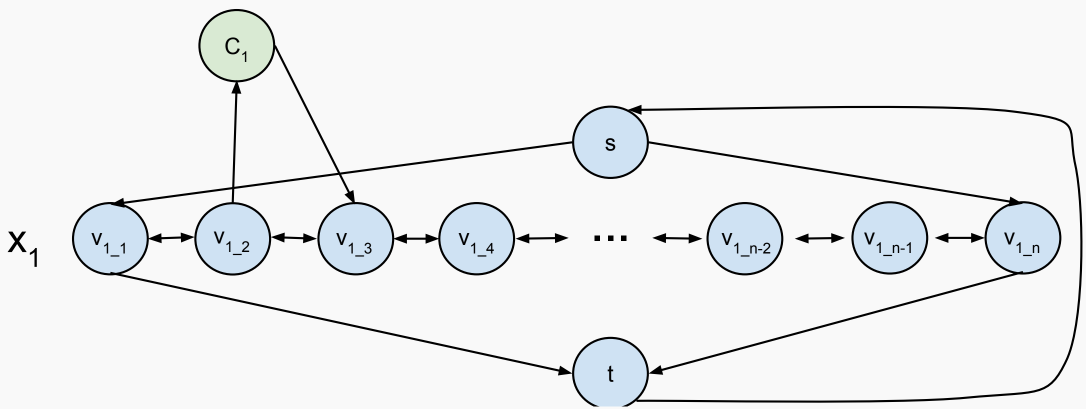
$\longleftarrow 0$



How do we handle clauses?

$$f(x_1, x_2) = (x_1 \vee \overline{x_2}) \tag{12}$$

What do we do if the clause has two literals:



Reduction: Encoding idea III

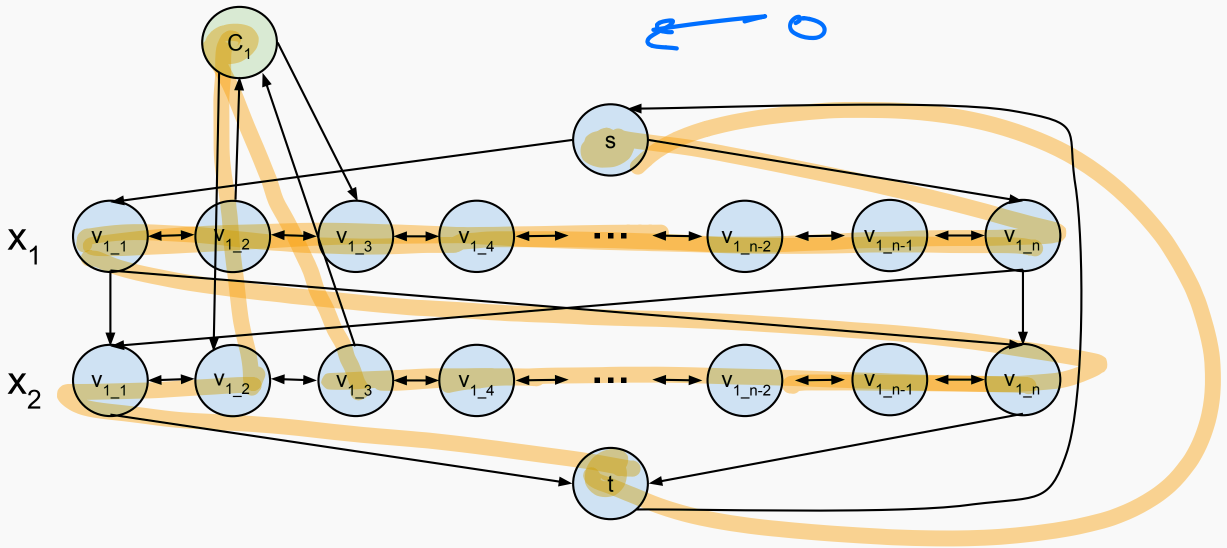
How do we handle clauses?

$$f(x_1, x_2) = (x_1 \vee \overline{x_2}) \qquad [0, 0] \tag{12}$$

What do we do if the clause has two literals:

$\rightarrow 1$

$\leftarrow 0$



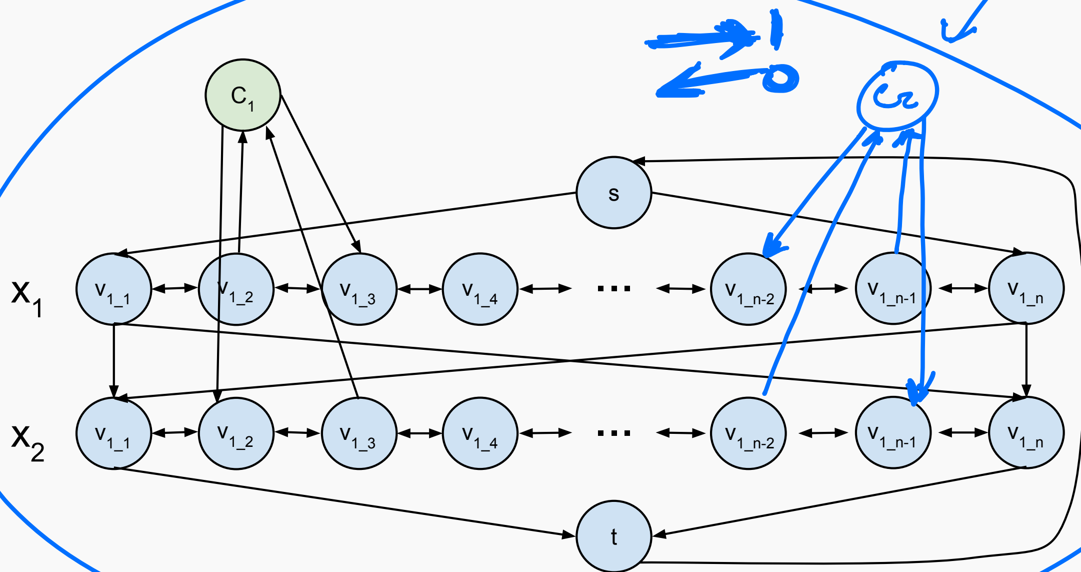
Reduction: Encoding idea III

How do we handle clauses? $f(x_1, x_2) = 1$

$$f(x_1, x_2) = (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2) \tag{13}$$

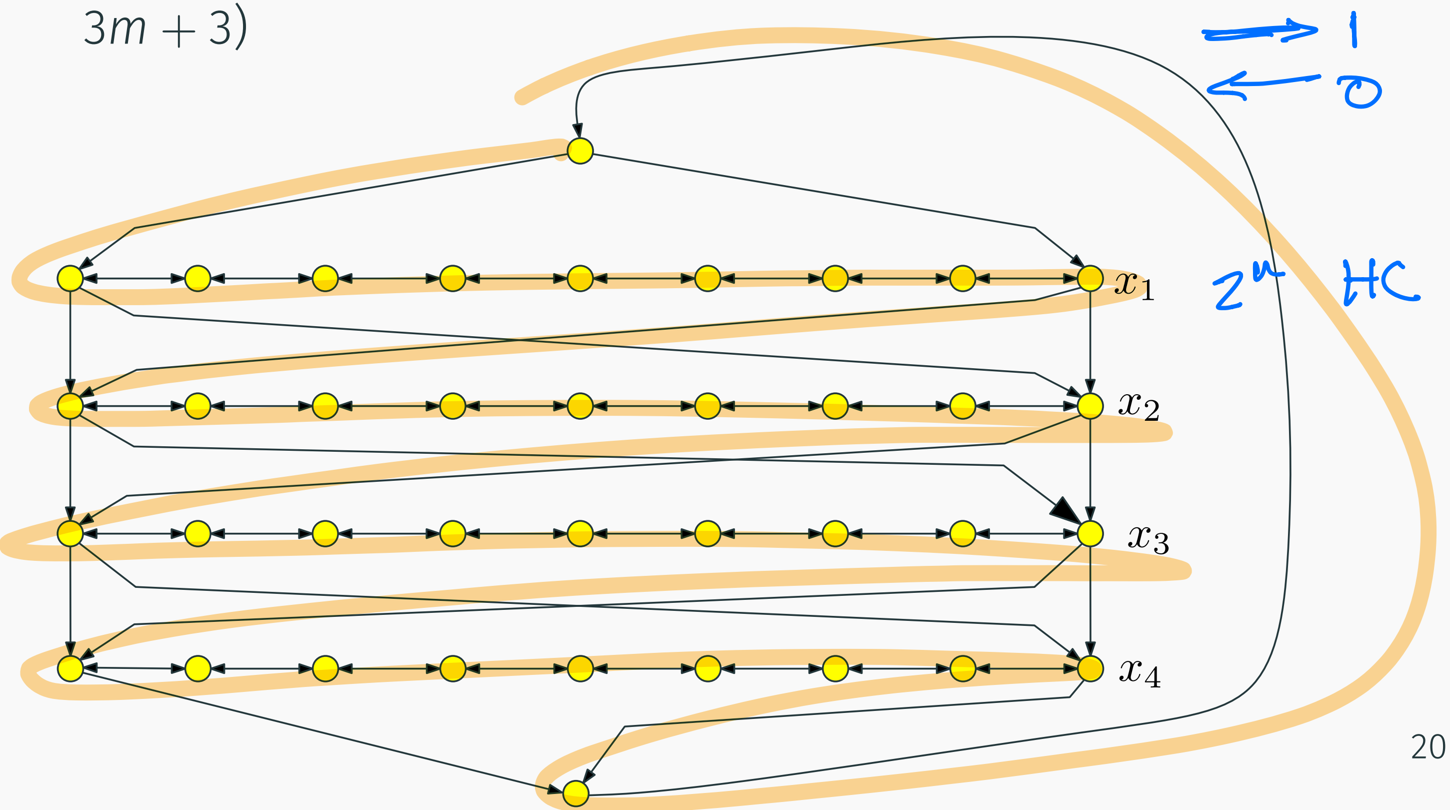
What if the expression has multiple clauses:

Gadgets



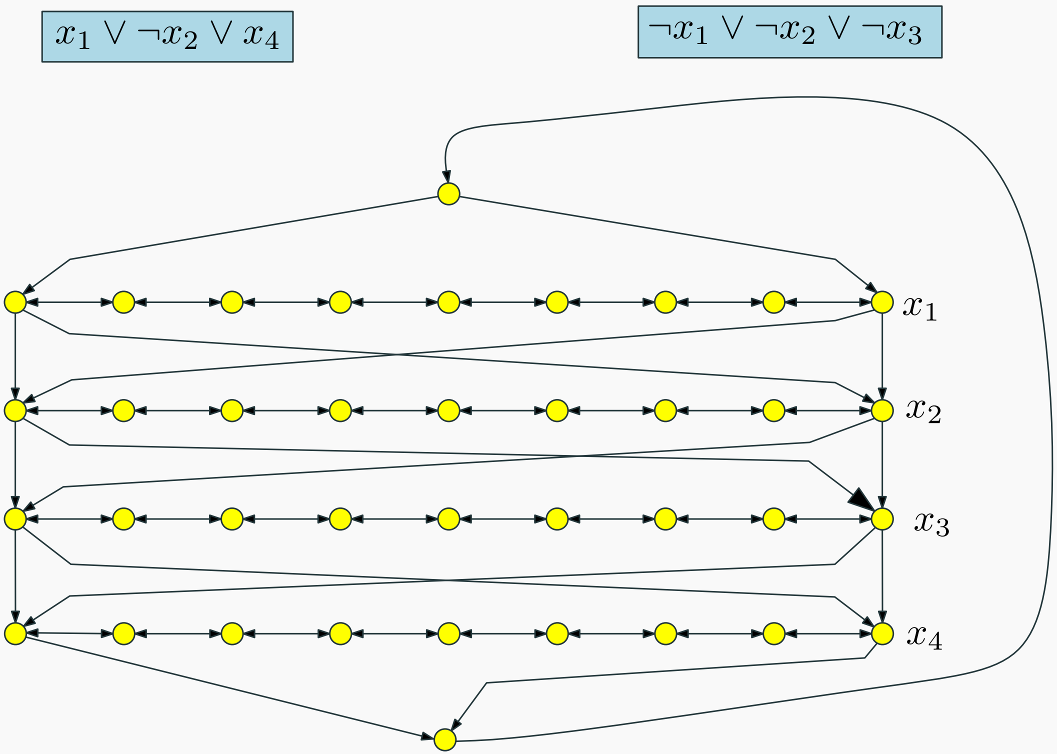
The Reduction: Review I

- Traverse path i from left to right iff x_i is set to true
- Each path has $3(m+1)$ nodes where m is number of clauses in φ; nodes numbered from left to right (1 to $3m+3$)



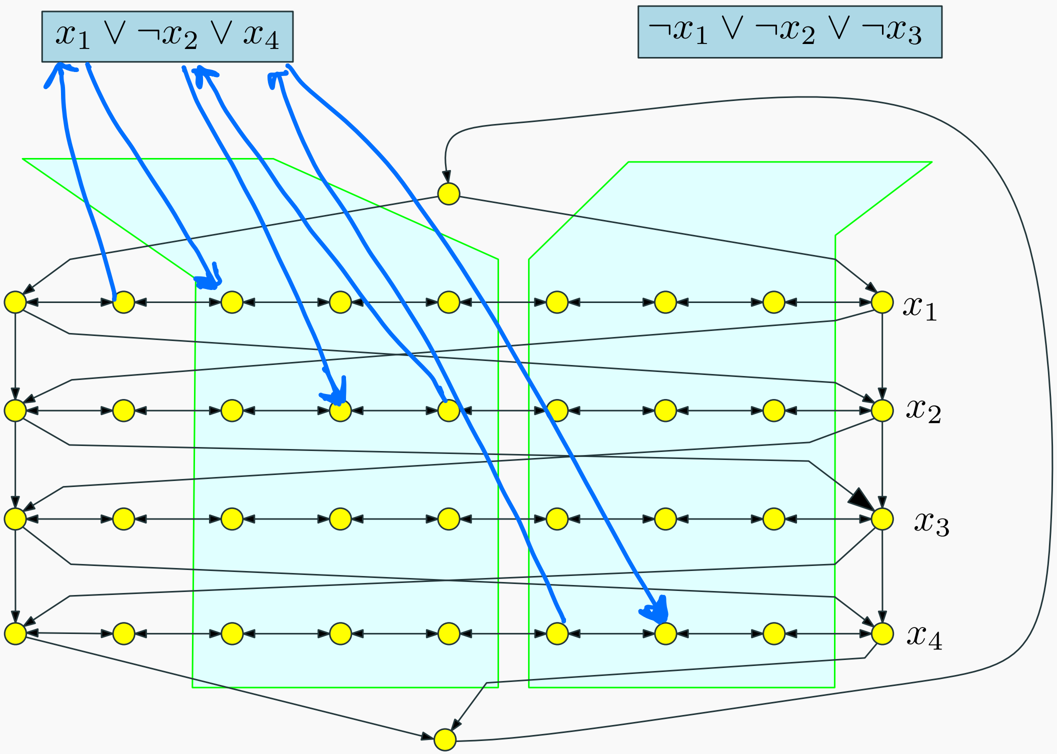
The Reduction algorithm: Review II

Add vertex c_j for clause C_j. c_j has edge *from* vertex $3j$ and *to* vertex $3j + 1$ on path i if x_i appears in clause C_j, and has edge *from* vertex $3j + 1$ and *to* vertex $3j$ if $\neg x_i$ appears in C_j.



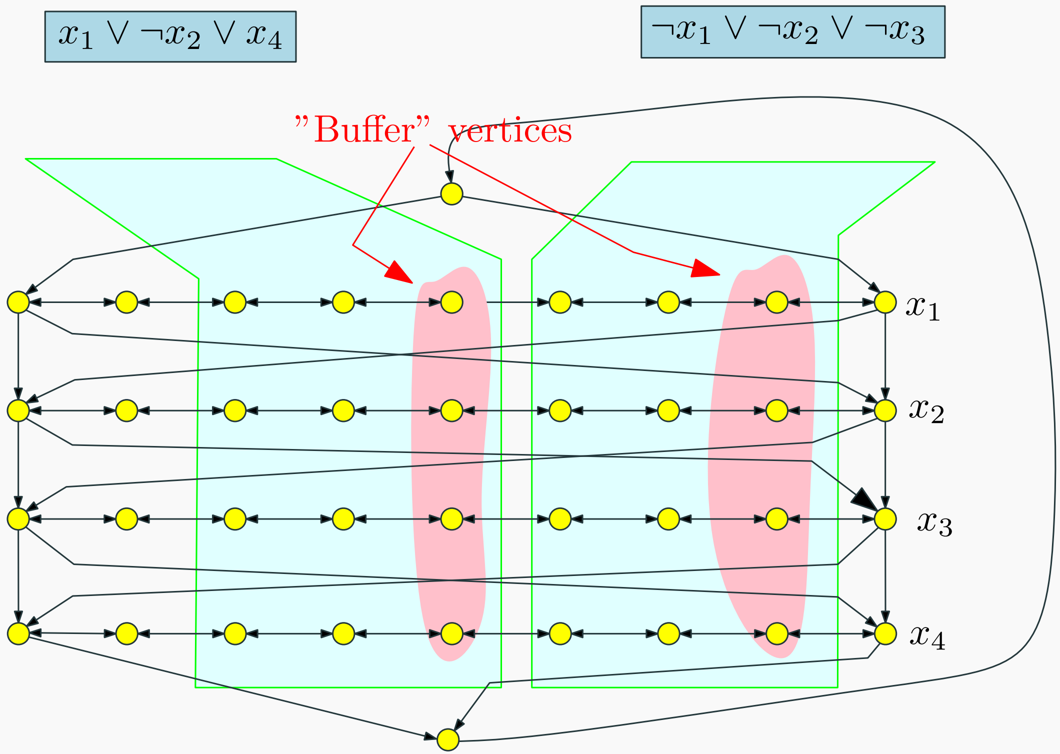
The Reduction algorithm: Review II

Add vertex c_j for clause C_j. c_j has edge *from* vertex $3j$ and *to* vertex $3j+1$ on path i if x_i appears in clause C_j, and has edge *from* vertex $3j+1$ and *to* vertex $3j$ if $\neg x_i$ appears in C_j.



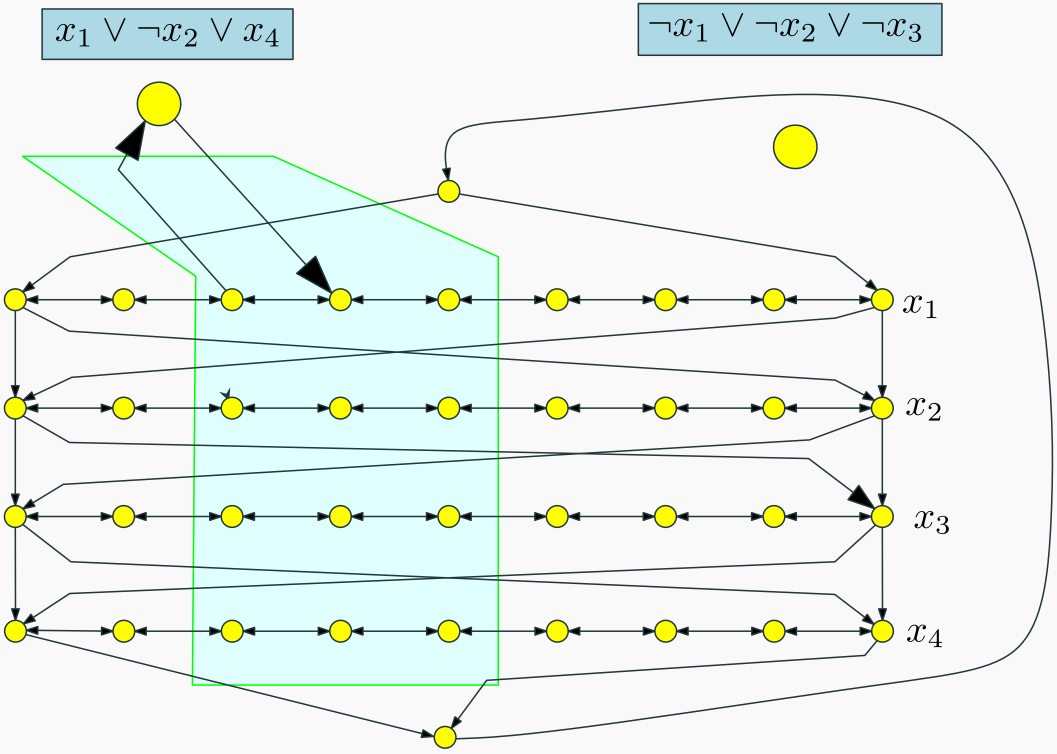
The Reduction algorithm: Review II

Add vertex c_j for clause C_j. c_j has edge *from* vertex $3j$ and *to* vertex $3j+1$ on path i if x_i appears in clause C_j, and has edge *from* vertex $3j+1$ and *to* vertex $3j$ if $\neg x_i$ appears in C_j.

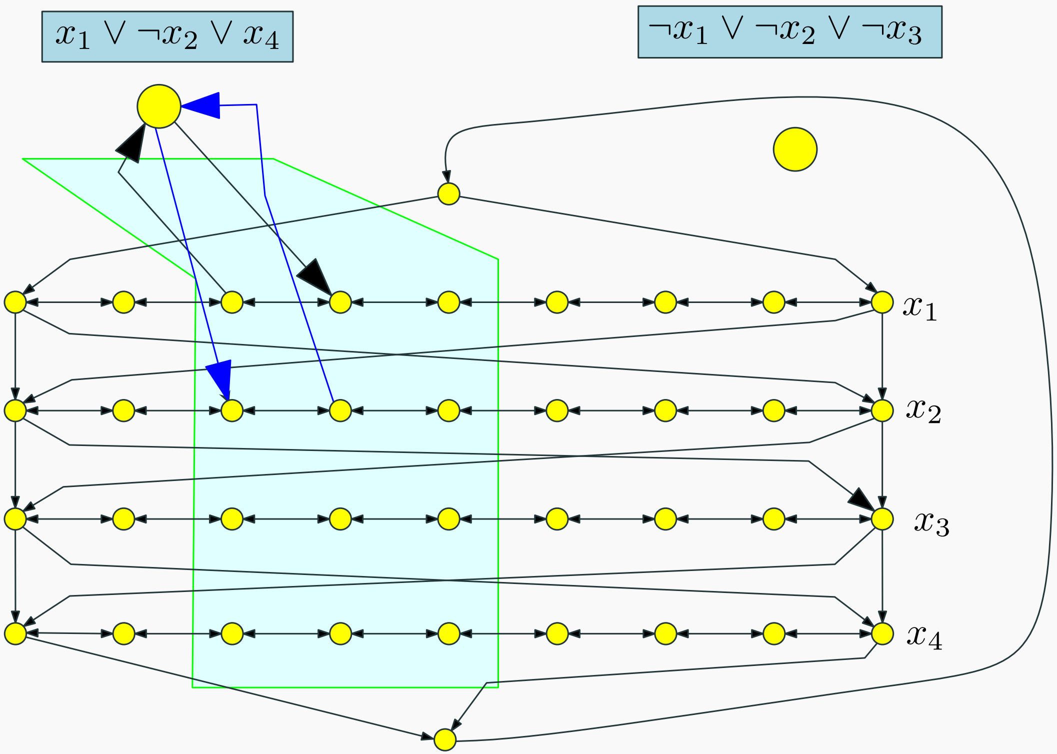


The Reduction algorithm: Review II

Add vertex c_j for clause C_j. c_j has edge *from* vertex $3j$ and *to* vertex $3j + 1$ on path i if x_i appears in clause C_j, and has edge *from* vertex $3j + 1$ and *to* vertex $3j$ if $\neg x_i$ appears in C_j.

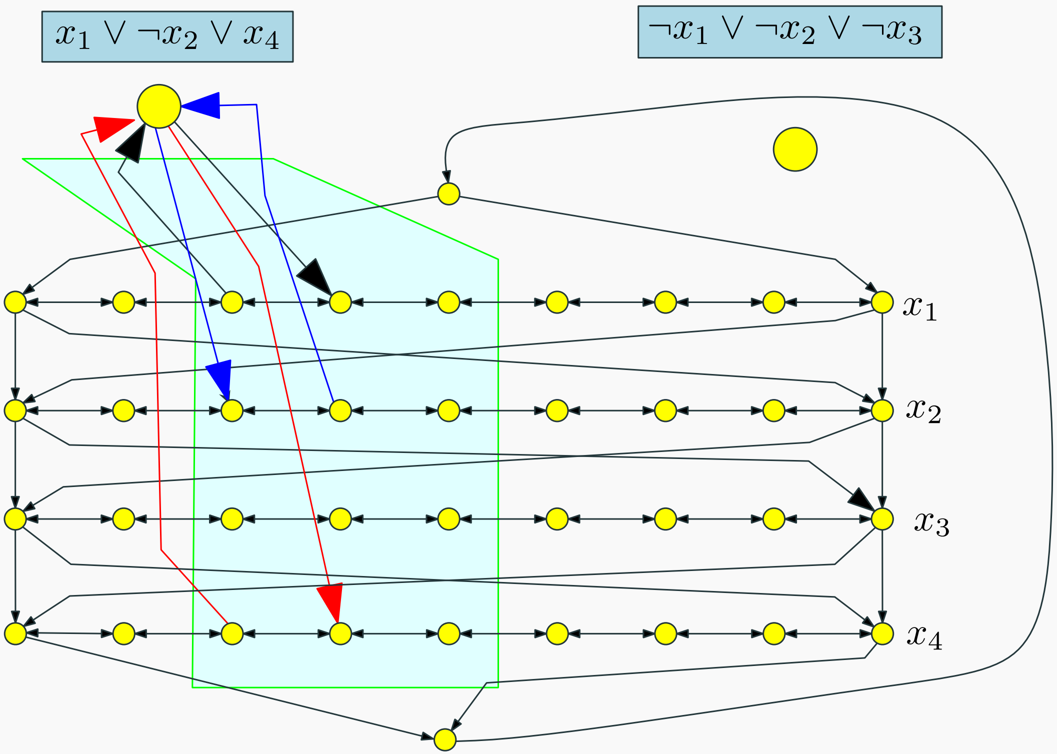


Add vertex c_j for clause C_j. c_j has edge *from* vertex $3j$ and *to* vertex $3j+1$ on path i if x_i appears in clause C_j, and has edge *from* vertex $3j+1$ and *to* vertex $3j$ if $\neg x_i$ appears in C_j.



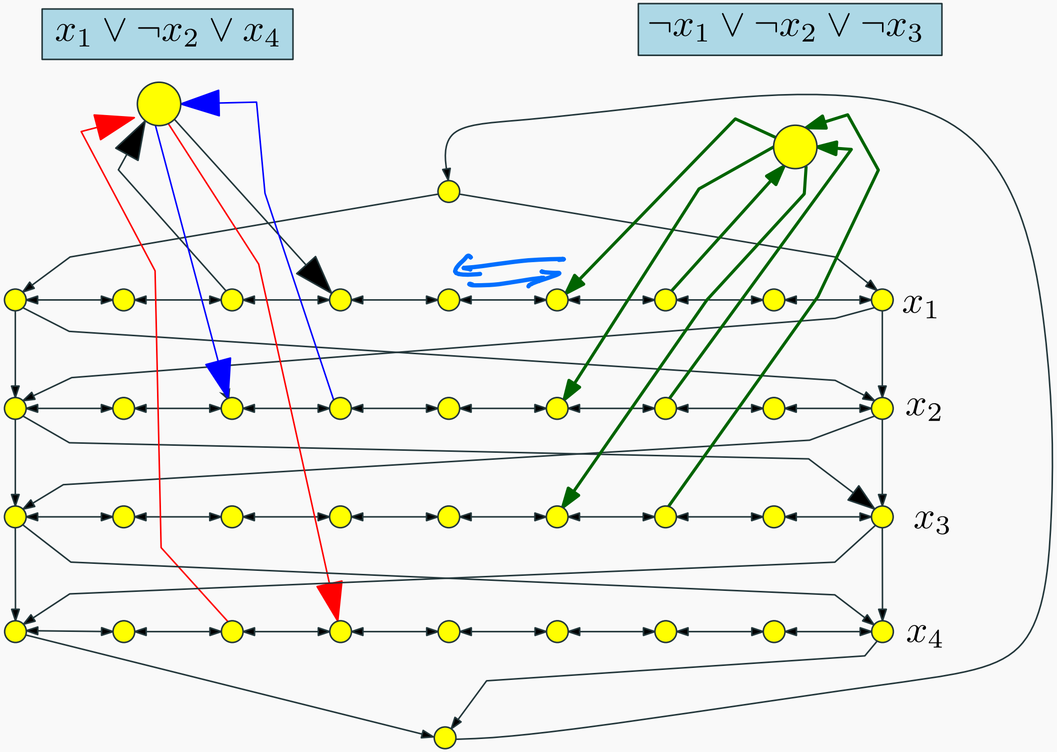
The Reduction algorithm: Review II

Add vertex c_j for clause C_j. c_j has edge *from* vertex $3j$ and *to* vertex $3j+1$ on path i if x_i appears in clause C_j, and has edge *from* vertex $3j+1$ and *to* vertex $3j$ if $\neg x_i$ appears in C_j.



The Reduction algorithm: Review II

Add vertex c_j for clause C_j. c_j has edge *from* vertex $3j$ and *to* vertex $3j + 1$ on path i if x_i appears in clause C_j, and has edge *from* vertex $3j + 1$ and *to* vertex $3j$ if $\neg x_i$ appears in C_j.



Correctness Proof

Theorem
φ has a satisfying assignment iff G_φ has a Hamiltonian cycle.

Based on proving following two lemmas.

Lemma
If φ has a satisfying assignment then G_φ has a Hamilton cycle.

Lemma
If G_φ has a Hamilton cycle then φ has a satisfying assignment.

Satisfying assignment $\rightarrow$ Hamiltonian Cycle

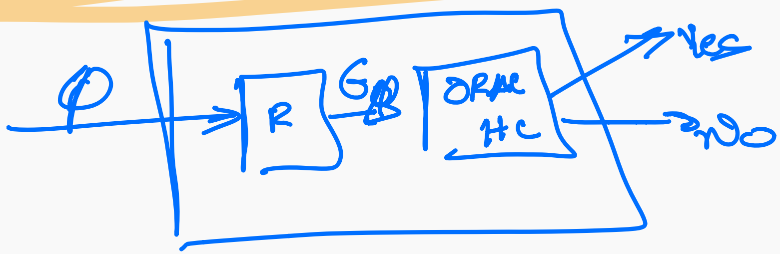
Lemma
If φ has a satisfying assignment then G_φ has a Hamilton cycle.

Proof.

$\Rightarrow$ Let a be the satisfying assignment for φ. Define Hamiltonian cycle as follows

- If $a(x_i) = 1$ then traverse path i from left to right
- If $a(x_i) = 0$ then traverse path i from right to left
- For each clause, path of at least one variable is in the "right" direction to splice in the node corresponding to clause

$\square$



Hamiltonian Cycle $\rightarrow$ Satisfying assignment

Suppose Π is a Hamiltonian cycle in G_φ

Definition
We say Π is *canonical* if for each clause vertex c_j the edge of Π entering c_j and edge of Π leaving c_j are from the same path corresponding to some variable x_i. Otherwise Π is *non-canonical* or emphcheating.

Hamiltonian Cycle $\rightarrow$ Satisfying assignment

Suppose Π is a Hamiltonian cycle in G_φ

Definition
We say Π is *canonical* if for each clause vertex c_j the edge of Π entering c_j and edge of Π leaving c_j are from the same path corresponding to some variable x_i. Otherwise Π is *non-canonical* or emphcheating.

Lemma
Every Hamilton cycle in G_φ is canonical.

Proof of Lemma

Lemma
Every Hamilton cycle in G_φ is canonical.

- If Π enters c_j (vertex for clause C_j) from vertex $3j$ on path i then it must leave the clause vertex on edge to $3j + 1$ on the *same path* i
    - If not, then only unvisited neighbor of $3j + 1$ on path i is $3j + 2$
    - Thus, we don't have two unvisited neighbors (one to enter from, and the other to leave) to have a Hamiltonian Cycle
- Similarly, if Π enters c_j from vertex $3j + 1$ on path i then it must leave the clause vertex c_j on edge to $3j$ on path i

Lemma
Any canonical Hamilton cycle in G_φ corresponds to a satisfying truth assignment to φ.

Consider a canonical Hamilton cycle Π.

- For every clause vertex c_j, vertices visited immediately before and after c_j are connected by an edge on same path corresponding to some variable x_i
- We can remove c_j from cycle, and get Hamiltonian cycle in $G - c_j$
- Hamiltonian cycle from Π in $G - \{c_1, \ldots c_m\}$ traverses each path in only one direction, which determines truth assignment
- Easy to verify that this truth assignment satisfies φ

Hamiltonian cycle in undirected graph

Hamiltonian Cycle in *Undirected* Graphs

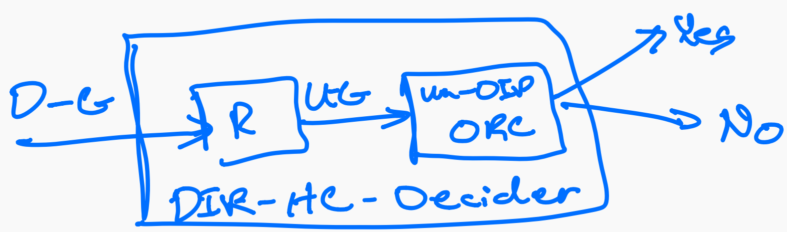
Problem

Input *Given undirected graph $G = (V, E)$*

Goal *Does G have a Hamiltonian cycle? That is, is there a cycle that visits every vertex exactly one (except start and end vertex)?*

Directed HC $\leq_p$ Undirected HC

We've proved Directed-HC is NP-complete



Theorem
Hamiltonian cycle problem for undirected *graphs is NP-Complete.*

Proof.

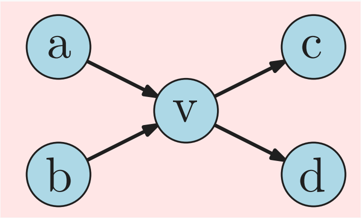
- The problem is in **NP**; proof left as exercise.
- Hardness proved by reducing Directed Hamiltonian Cycle to this problem □

Reduction Sketch

Goal: Given directed graph G, need to construct undirected graph G' such that G has Hamiltonian Path iff G' has Hamiltonian path

Reduction

-
-

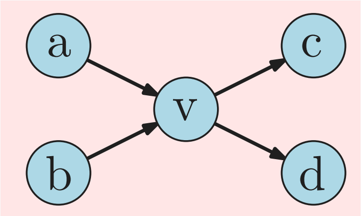


Reduction Sketch

Goal: Given directed graph G, need to construct undirected graph G' such that G has Hamiltonian Path iff G' has Hamiltonian path

Reduction

- Replace each vertex v by 3 vertices: v_{in}, v, and v_{out}
-

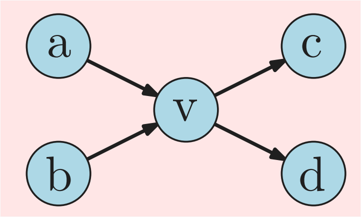


Reduction Sketch

Goal: Given directed graph G, need to construct undirected graph G' such that G has Hamiltonian Path iff G' has Hamiltonian path

Reduction

- Replace each vertex v by 3 vertices: v_{in}, v, and v_{out}
- A directed edge (a, b) is replaced by edge (a_{out}, b_{in})

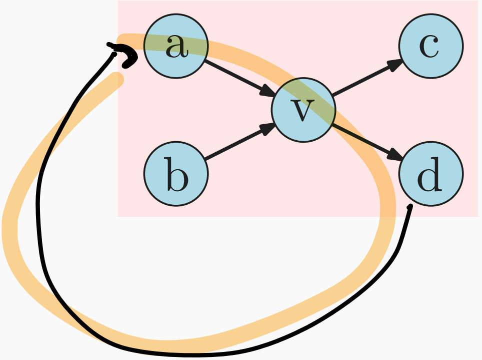


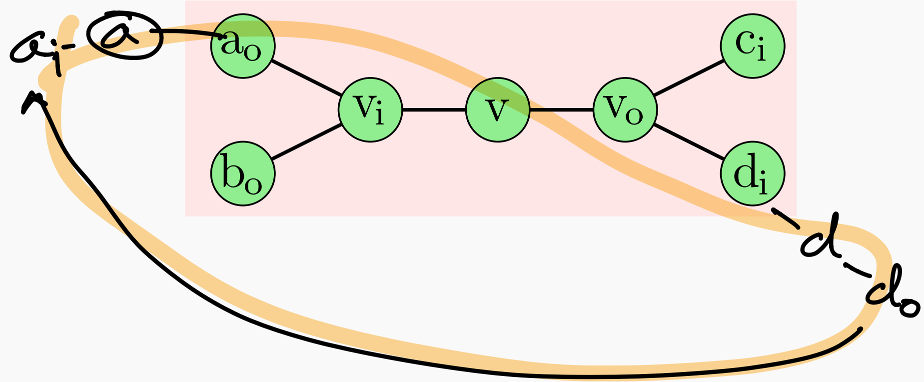
Reduction Sketch

Goal: Given directed graph G, need to construct undirected graph G' such that G has Hamiltonian Path iff G' has Hamiltonian path

Reduction

- Replace each vertex v by 3 vertices: v_{in}, v, and v_{out}
- A directed edge (a, b) is replaced by edge (a_{out}, b_{in})





Reduction: Wrapup

- The reduction is polynomial time (exercise)
- The reduction is correct (exercise)

Hamiltonian Path

Input Given a graph $G = (V, E)$ with n vertices

Goal Does G have a Hamiltonian path?

- A Hamiltonian path is a path in the graph that visits every vertex in G exactly once

Hamiltonian Path

Input Given a graph $G = (V, E)$ with n vertices

Goal Does G have a Hamiltonian path?

- A Hamiltonian path is a path in the graph that visits every vertex in G exactly once

Theorem
Directed Hamiltonian Path and Undirected Hamiltonian Path are NP-Complete.

Easy to modify the reduction from **3-SAT** to **Halitonian Cycle** or do a reduction from **Halitonian Cycle**

Hamiltonian Path

Input Given a graph $G = (V, E)$ with n vertices

Goal Does G have a Hamiltonian path?

- A Hamiltonian path is a path in the graph that visits every vertex in G exactly once

Theorem
Directed Hamiltonian Path and Undirected Hamiltonian Path are NP-Complete.

Easy to modify the reduction from **3-SAT** to **Halitonian Cycle** or do a reduction from **Halitonian Cycle**

Implies that **Longest Simple Path** in a graph is NP-Complete.

NP-Completeness of Graph Coloring

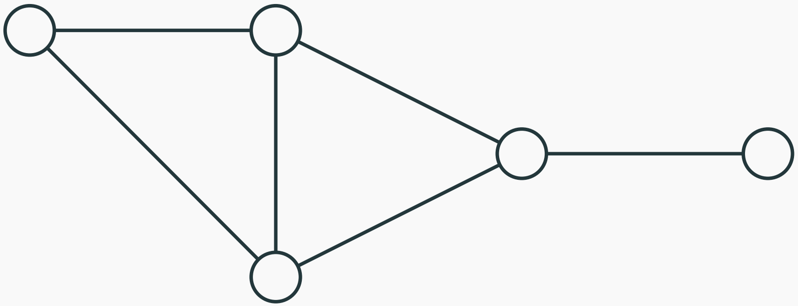
Problem: Graph Coloring

> **Instance:** $G = (V, E)$: Undirected graph, integer k.
>
> **Question:** Can the vertices of the graph be colored using k colors so that vertices connected by an edge do not get the same color?

Problem: 3 Coloring

Instance: $G = (V, E)$: Undirected graph.

Question: Can the vertices of the graph be colored using 3 colors so that vertices connected by an edge do not get the same color?

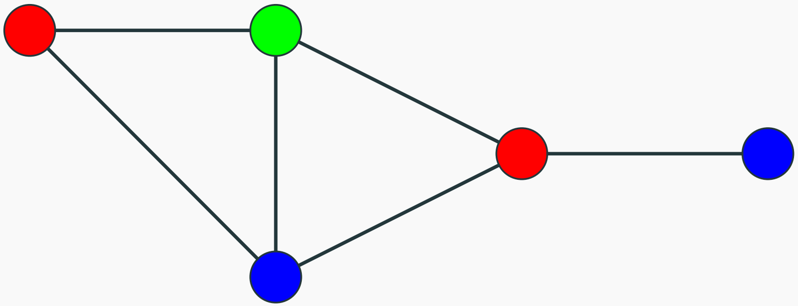


'

Problem: 3 Coloring

Instance: $G = (V, E)$: Undirected graph.

Question: Can the vertices of the graph be colored using 3 colors so that vertices connected by an edge do not get the same color?



'

Graph Coloring

Observation: If G is colored with k colors then each color class (nodes of same color) form an independent set in G. Thus, G can be partitioned into k independent sets iff G is k-colorable.

Graph 2-Coloring can be decided in polynomial time.

G is 2-colorable iff G is bipartite! There is a linear time algorithm to check if G is bipartite using Breadth-first-Search

Problems related to graph coloring

Graph Coloring and Register Allocation

Register Allocation
Assign variables to (at most) k registers such that variables needed at the same time are not assigned to the same register

Interference Graph
Vertices are variables, and there is an edge between two vertices, if the two variables are "live" at the same time.

Observations

- [Chaitin] Register allocation problem is equivalent to coloring the interference graph with k colors
- Moreover, 3-COLOR $\leq_P k -$ Register Allocation, for any $k \geq 3$

Class Room Scheduling

Given n classes and their meeting times, are k rooms sufficient?

Reduce to Graph k-Coloring problem

Create graph G

- a node v_i for each class i
- an edge between v_i and v_j if classes i and j *conflict*

Exercise: G is k-colorable iff k rooms are sufficient

Frequency Assignments in Cellular Networks

Cellular telephone systems that use Frequency Division Multiple Access (FDMA) (example: GSM in Europe and Asia and AT&T in USA)

- Breakup a frequency range $[a, b]$ into disjoint *bands* of frequencies $[a_0, b_0], [a_1, b_1], \ldots, [a_k, b_k]$
- Each cell phone tower (simplifying) gets one band
- Constraint: nearby towers cannot be assigned same band, otherwise signals will interference

Frequency Assignments in Cellular Networks

Cellular telephone systems that use Frequency Division Multiple Access (FDMA) (example: GSM in Europe and Asia and AT&T in USA)

- Breakup a frequency range $[a, b]$ into disjoint *bands* of frequencies $[a_0, b_0], [a_1, b_1], \ldots, [a_k, b_k]$
- Each cell phone tower (simplifying) gets one band
- Constraint: nearby towers cannot be assigned same band, otherwise signals will interference

Problem: given k bands and some region with n towers, is there a way to assign the bands to avoid interference?

Can reduce to k-coloring by creating intereference/conflict graph on towers.

Showing hardness of 3 COLORING

3-Coloring is NP-Complete

- 3-Coloring is in NP.
  - Non-deterministically guess a 3-coloring for each node
  - Check if for each edge (u, v), the color of u is different from that of v.
- Hardness: We will show 3-SAT $\leq_P$ 3-Coloring.

Reduction Idea

Start with 3SAT formula (i.e., 3CNF formula) φ with n variables $x_1, \ldots, x_n$ and m clauses $C_1, \ldots, C_m$. Create graph G_φ such that G_φ is 3-colorable iff φ is satisfiable

- need to establish truth assignment for $x_1, \ldots, x_n$ via colors for some nodes in G_φ.
- create triangle with node True, False, Base
- for each variable x_i two nodes v_i and $\bar{v}_i$ connected in a triangle with common Base
- If graph is 3-colored, either v_i or $\bar{v}_i$ gets the same color as True. Interpret this as a truth assignment to v_i
- Need to add constraints to ensure clauses are satisfied (next phase)

Reduction Idea I - Simple 3-color gadget

We want to create a gadget that:

- Is 3 colorable if at least one of the literals is true
- Not 3-colorable if none of the literals are true

Reduction Idea I - Simple 3-color gadget

We want to create a gadget that:

- Is 3 colorable if at least one of the literals is true
- Not 3-colorable if none of the literals are true

Let's start off with the simplest SAT we cna think of:

$$f(x_1, x_2) = (x_1 \vee x_2) \tag{14}$$

Reduction Idea I - Simple 3-color gadget

We want to create a gadget that:

- Is 3 colorable if at least one of the literals is true
- Not 3-colorable if none of the literals are true

Let's start off with the simplest SAT we cna think of:

$$f(x_1, x_2) = (x_1 \vee x_2) \tag{14}$$

Assume green=true and red=false,

Reduction Idea I - Simple 3-color gadget

We want to create a gadget that:

- Is 3 colorable if at least one of the literals is true
- Not 3-colorable if none of the literals are true

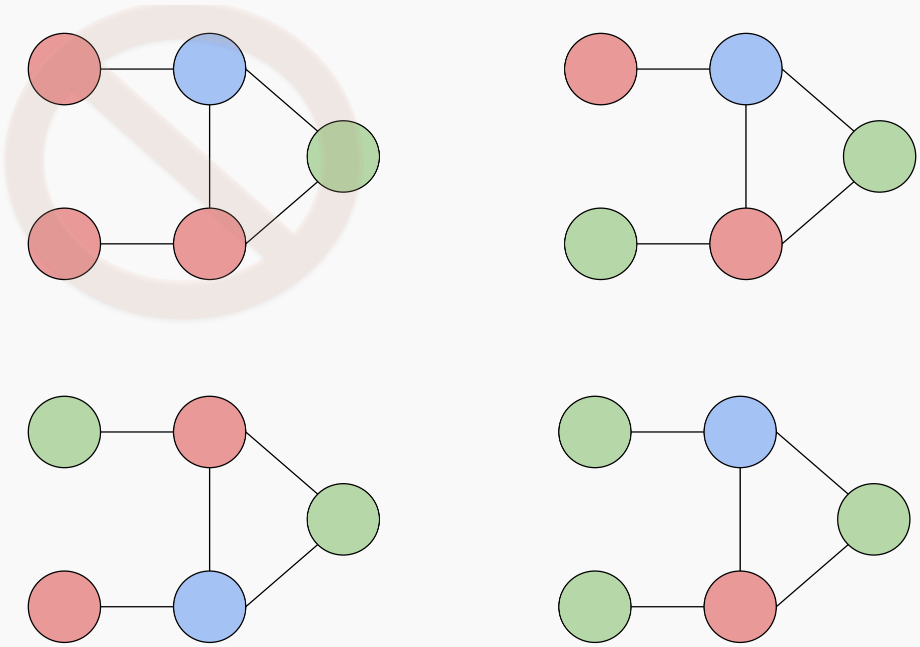
Let's try some stuff:

Reduction Idea I - Simple 3-color gadget

We want to create a gadget that:

- Is 3 colorable if at least one of the literals is true
- Not 3-colorable if none of the literals are true

Seems to work:



Reduction Idea I - Simple 3-color gadget

We want to create a gadget that:

- Is 3 colorable if at least one of the literals is true
- Not 3-colorable if none of the literals are true

Reduction Idea I - Simple 3-color gadget

We want to create a gadget that:

- Is 3 colorable if at least one of the literals is true
- Not 3-colorable if none of the literals are true

How do we do the same thing for 3 variables?:

$$f(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \tag{15}$$

Reduction Idea I - Simple 3-color gadget

We want to create a gadget that:

- Is 3 colorable if at least one of the literals is true
- Not 3-colorable if none of the literals are true

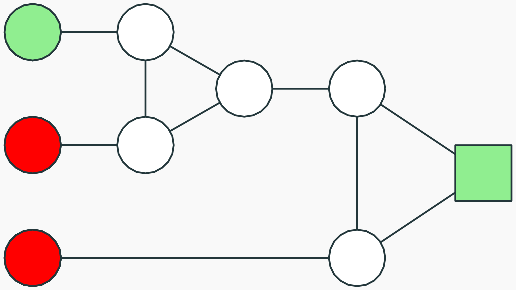
How do we do the same thing for 3 variables?:

$$f(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \tag{15}$$

Assume green=true and red=false,

3 color this gadget II

You are given three colors: red, green and blue. Can the following graph be three colored in a valid way (assuming that some of the nodes are already colored as indicated).

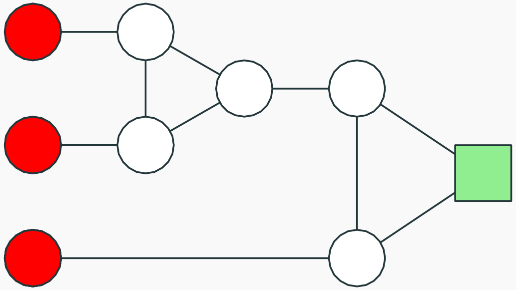


a Yes.

b No.

3 color this gadget.

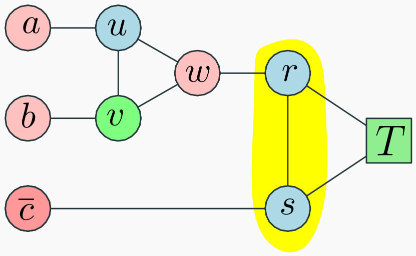
You are given three colors: red, green and blue. Can the following graph be three colored in a valid way (assuming that some of the nodes are already colored as indicated).



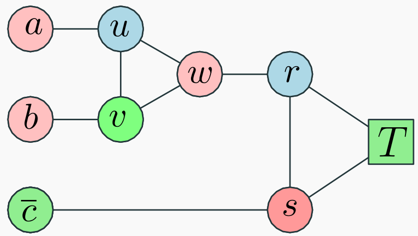
a Yes.

b No.

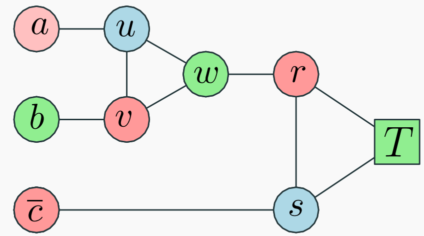
3-coloring of the clause gadget



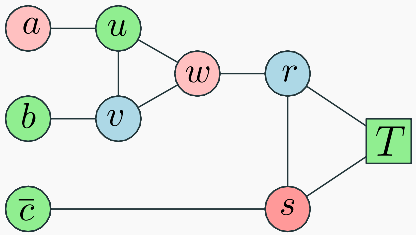
FFF - **BAD**



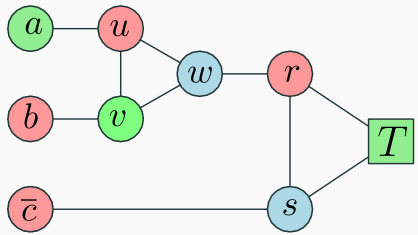
FFT



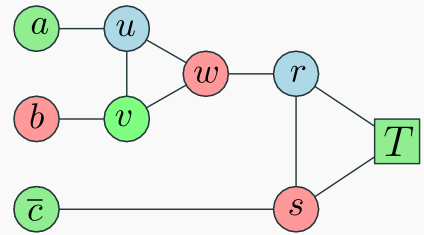
FTF



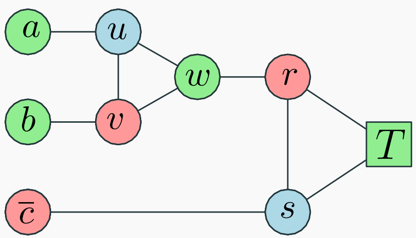
FTT



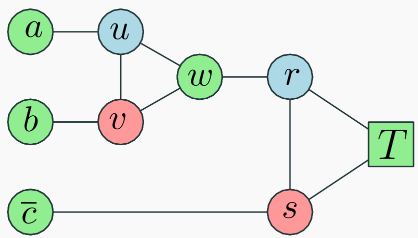
TFF



TFT



TTF



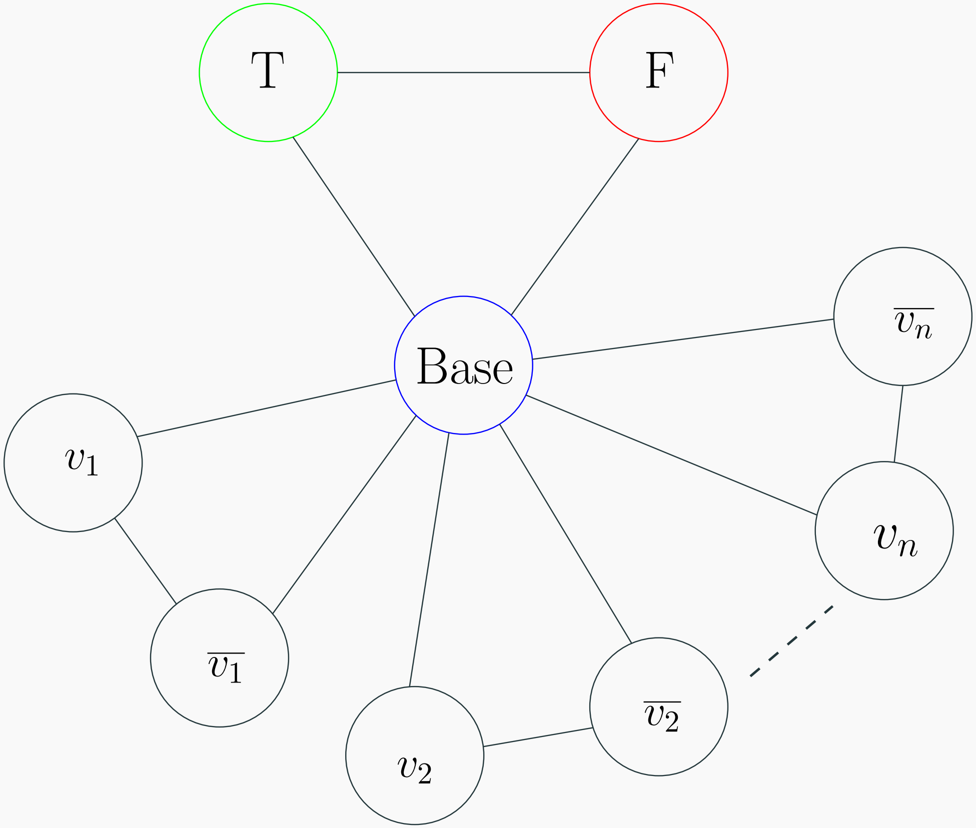
TTT

Reduction Idea II - Literal Assignment I

Next we need a gadget that assigns literals. Our previously constructed gadget assumes:

- All literals are either red or green.
- Need to limit graph so only x_1 or $\overline{x_1}$ is green. Other must be red

Reduction Idea II - Literal Assignment II

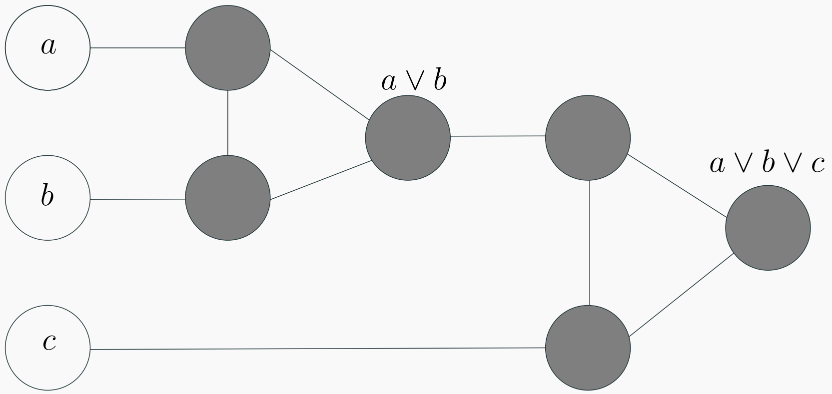


Review Clause Satisfiability Gadget

For each clause $C_j = (a \vee b \vee c)$, create a small gadget graph

- gadget graph connects to nodes corresponding to a, b, c
- needs to implement OR

OR-gadget-graph:



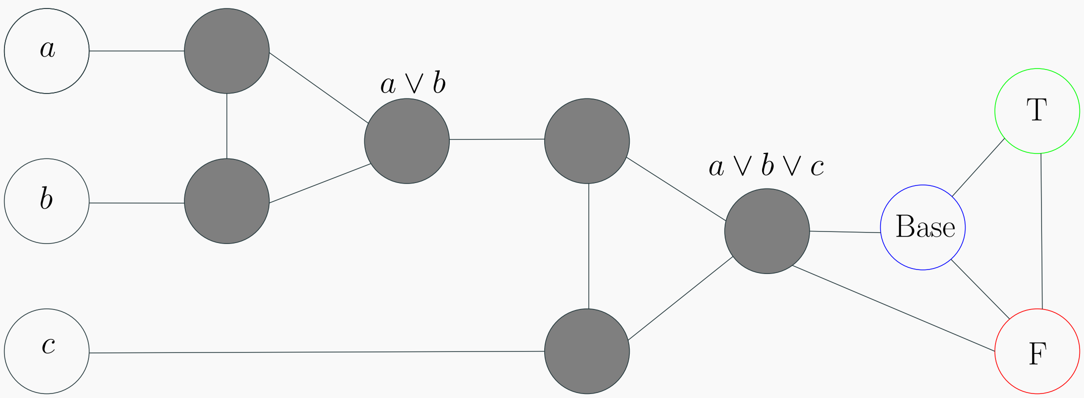
OR-Gadget Graph

Property: if a, b, c are colored False in a 3-coloring then output node of OR-gadget has to be colored False.

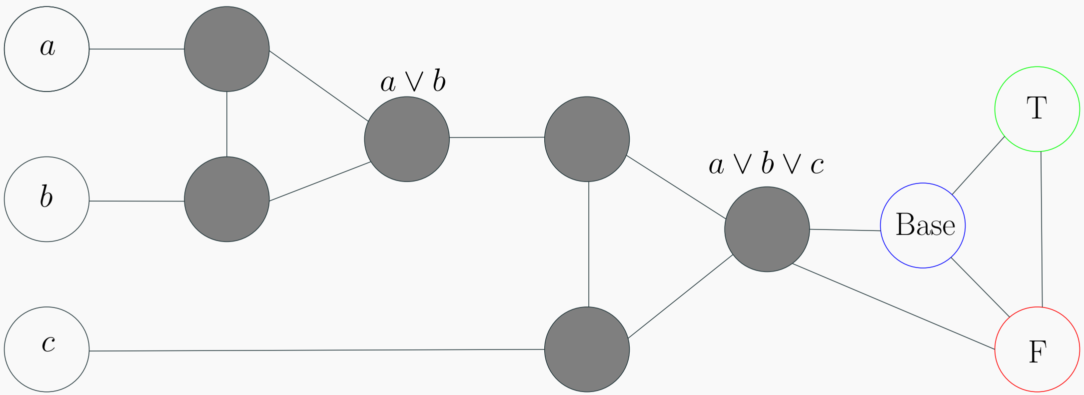
Property: if one of a, b, c is colored True then OR-gadget can be 3-colored such that output node of OR-gadget is colored True.

Reduction

- create triangle with nodes True, False, Base
- for each variable x_i two nodes v_i and $\bar{v}_i$ connected in a triangle with common Base
- for each clause $C_j = (a \vee b \vee c)$, add OR-gadget graph with input nodes a, b, c and connect output node of gadget to both False and Base



Reduction



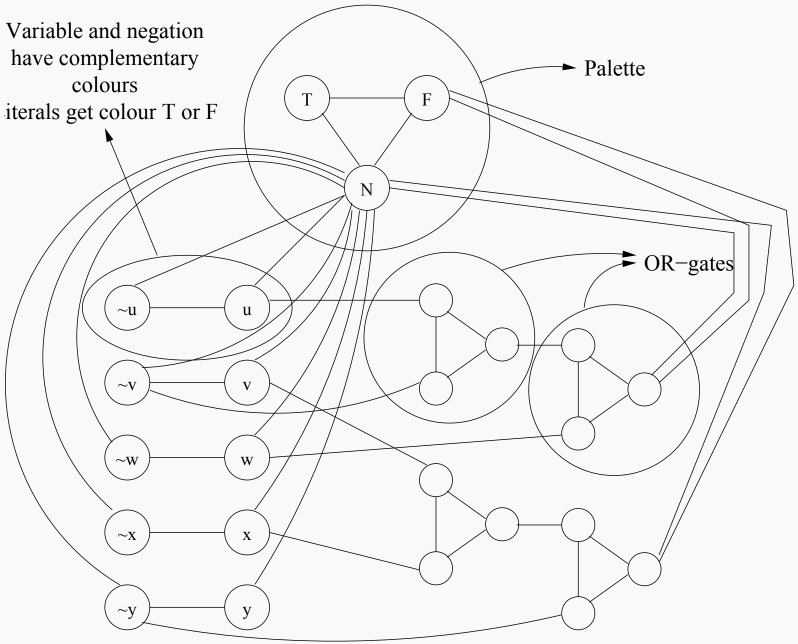
Lemma

No legal 3-coloring of above graph (with coloring of nodes T, F, B fixed) in which a, b, c are colored False. If any of a, b, c are colored True then there is a legal 3-coloring of above graph.

Reduction Outline

Example

$\varphi = (u \vee \neg v \vee w) \wedge (v \vee x \vee \neg y)$



Correctness of Reduction

φ is satisfiable implies G_φ is 3-colorable

- if x_i is assigned True, color v_i True and $\bar{v}_i$ False

Correctness of Reduction

φ is satisfiable implies G_φ is 3-colorable

- if x_i is assigned True, color v_i True and $\bar{v}_i$ False
- for each clause $C_j = (a \vee b \vee c)$ at least one of a, b, c is colored True. OR-gadget for C_j can be 3-colored such that output is True.

Correctness of Reduction

φ is satisfiable implies G_φ is 3-colorable

- if x_i is assigned True, color v_i True and $\bar{v}_i$ False
- for each clause $C_j = (a \vee b \vee c)$ at least one of a, b, c is colored True. OR-gadget for C_j can be 3-colored such that output is True.

Correctness of Reduction

φ is satisfiable implies G_φ is 3-colorable

- if x_i is assigned True, color v_i True and $\bar{v}_i$ False
- for each clause $C_j = (a \vee b \vee c)$ at least one of a, b, c is colored True. OR-gadget for C_j can be 3-colored such that output is True.

G_φ is 3-colorable implies φ is satisfiable

- if v_i is colored True then set x_i to be True, this is a legal truth assignment

Correctness of Reduction

φ is satisfiable implies G_φ is 3-colorable

- if x_i is assigned True, color v_i True and $\bar{v}_i$ False
- for each clause $C_j = (a \vee b \vee c)$ at least one of a, b, c is colored True. OR-gadget for C_j can be 3-colored such that output is True.

G_φ is 3-colorable implies φ is satisfiable

- if v_i is colored True then set x_i to be True, this is a legal truth assignment
- consider any clause $C_j = (a \vee b \vee c)$. it cannot be that all a, b, c are False. If so, output of OR-gadget for C_j has to be colored False but output is connected to Base and False!

Graph generated in reduction from 3SAT to 3COLOR