BBM402-Lecture 9: More NP-Complete Problems

Lecturer: Lale Özkahya

Resources for the presentation:
https://courses.engr.illinois.edu/cs473/fa2016/lectures.html
https://courses.engr.illinois.edu/cs374/fa2015/lectures.html

Recap

NP: languages that have non-deterministic polynomial time algorithms

Recap

NP: languages that have non-deterministic polynomial time algorithms

A language L is **NP-Complete** iff

- L is in **NP**
- for every L' in **NP**, $L' \leq_P L$

Recap

NP: languages that have non-deterministic polynomial time algorithms

A language L is **NP-Complete** iff

- L is in **NP**
- for every L' in **NP**, $L' \leq_P L$

L is **NP-Hard** if for every L' in **NP**, $L' \leq_P L$.

Recap

NP: languages that have non-deterministic polynomial time algorithms

A language L is **NP-Complete** iff

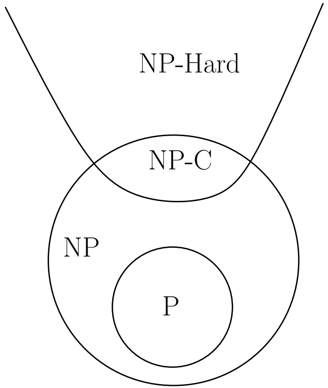
- L is in **NP**
- for every L' in **NP**, $L' \leq_P L$

L is **NP-Hard** if for every L' in **NP**, $L' \leq_P L$.

Theorem (Cook-Levin)

SAT *is* **NP-Complete**.

Pictorial View



P and NP

Possible scenarios:

1. $\mathbf{P} = \mathbf{NP}$.
2. $\mathbf{P} \neq \mathbf{NP}$

P and NP

Possible scenarios:

1. $\mathbf{P} = \mathbf{NP}$.
2. $\mathbf{P} \neq \mathbf{NP}$

Question: Suppose $\mathbf{P} \neq \mathbf{NP}$. Is every problem in $\mathbf{NP} \setminus \mathbf{P}$ also **NP-Complete**?

P and NP

Possible scenarios:

1. $\mathbf{P} = \mathbf{NP}$.
2. $\mathbf{P} \neq \mathbf{NP}$

Question: Suppose $\mathbf{P} \neq \mathbf{NP}$. Is every problem in $\mathbf{NP} \setminus \mathbf{P}$ also **NP-Complete**?

Theorem (Ladner)

*If $\mathbf{P} \neq \mathbf{NP}$ then there is a problem/language $X \in \mathbf{NP} \setminus \mathbf{P}$ such that X is not **NP-Complete**.*

Today

NP-Completeness of three problems:

- **3**-Color
- Circuit SAT
- SAT (Cook-Levin Theorem)

Important: understanding the problems and that they are hard.

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

Part I

NP-Completeness of Graph Coloring

Graph Coloring

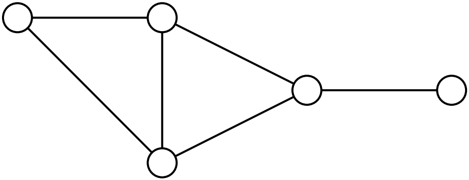
Problem: Graph Coloring

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?

Graph 3-Coloring

Problem: 3 Coloring

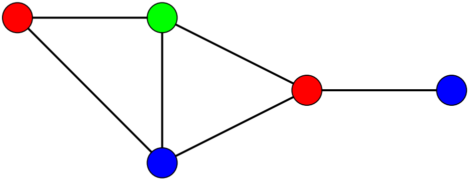
Instance: $G = (V, E)$: Undirected graph.
Question: Can the vertices of the graph be colored using $\mathbf{3}$ colors so that vertices connected by an edge do not get the same color?



Graph 3-Coloring

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 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 $\mathbf{2}$-Coloring can be decided in polynomial time.

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!

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 **BFS**

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?

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*

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)

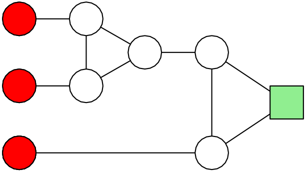
- Breakup a frequency range $[a, b]$ into disjoint *bands* of frequencies $[a_0, b_0], [a_1, b_1], \ldots, [a_k, b_k]$
- Each cell phone tower (simplifying) gets one band
- Constraint: nearby towers cannot be assigned same band, otherwise signals will interference

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.

3 color this gadget.

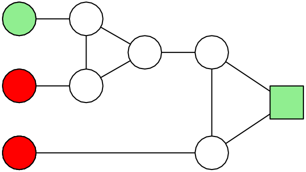
You are given three colors: red, green and blue. Can the following graph be three colored in a valid way (assuming that some of the nodes are already colored as indicated).



(A) Yes.
(B) No.

3 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-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., $3\mathrm{CNF}$ 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.

Reduction Idea

Start with **3SAT** formula (i.e., $3\mathrm{CNF}$ 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

Reduction Idea

Start with **3SAT** formula (i.e., $3\mathrm{CNF}$ 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

Reduction Idea

Start with **3SAT** formula (i.e., $3\mathrm{CNF}$ 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

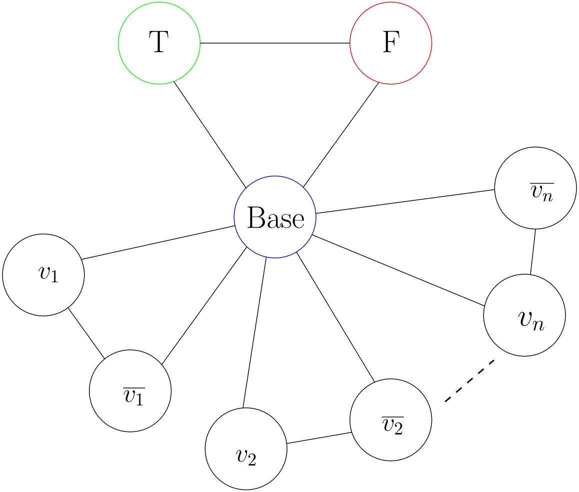
- need to establish truth assignment for $x_1, \ldots, x_n$ via colors for some nodes in G_φ.
- create triangle with node True, False, Base
- for each variable x_i two nodes v_i and $\bar{v}_i$ connected in a triangle with common Base
- If graph is 3-colored, either v_i or $\bar{v}_i$ gets the same color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

Reduction Idea

Start with **3SAT** formula (i.e., $3\mathrm{CNF}$ 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)

Figure

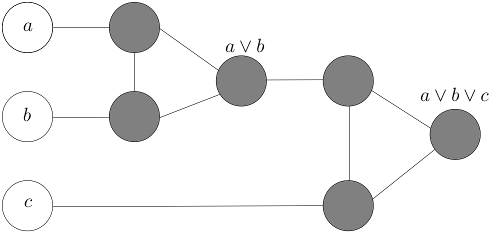


Clause Satisfiability Gadget

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:



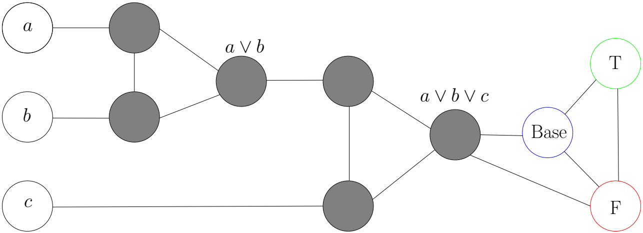
OR-Gadget Graph

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

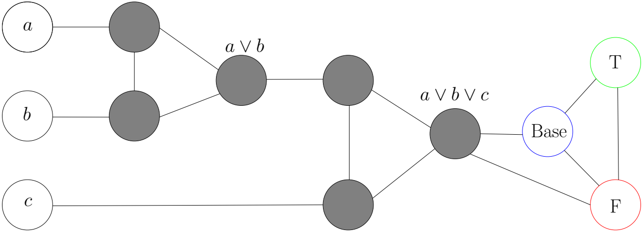
Property: if one of a, b, c is colored True then OR-gadget can be 3-colored such that output node of OR-gadget is colored True.

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



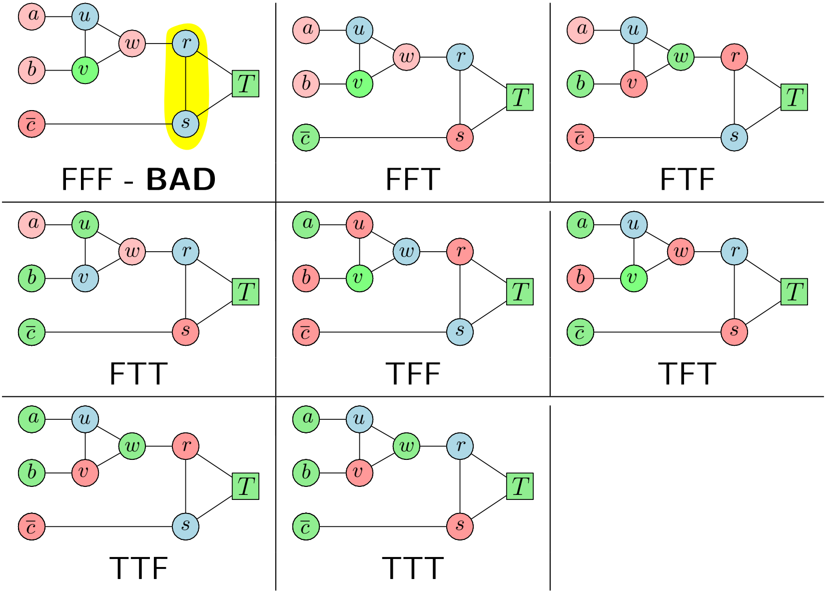
Reduction



Claim

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.*

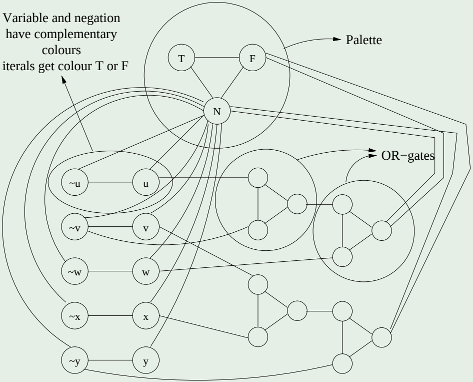
3 coloring of the clause gadget



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

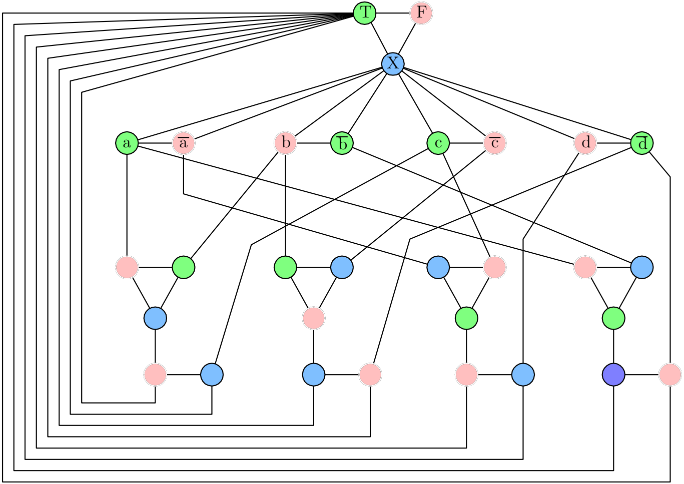
- if x_i is assigned True, color v_i True and $\bar{v}_i$ False
- for each clause $C_j = (a \vee b \vee c)$ at least one of a, b, c is colored True. OR-gadget for C_j can be 3-colored such that output is True.

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



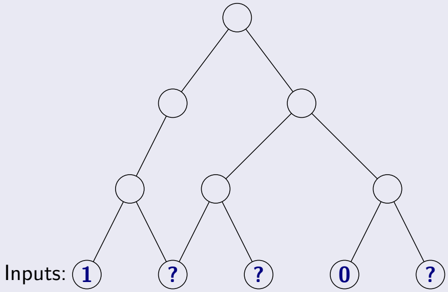
Part II

Circuit SAT

Circuits

Definition

A circuit is a directed *acyclic* graph with

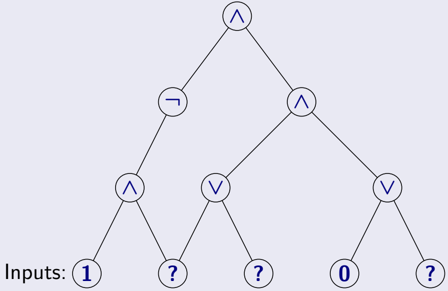


1. Input vertices (without incoming edges) labelled with **0**, **1** or a distinct variable.
2. Every other vertex is labelled $\lor$, $\land$ or $\neg$.
3. Single node output vertex with no outgoing edges.

Circuits

Definition

A circuit is a directed *acyclic* graph with

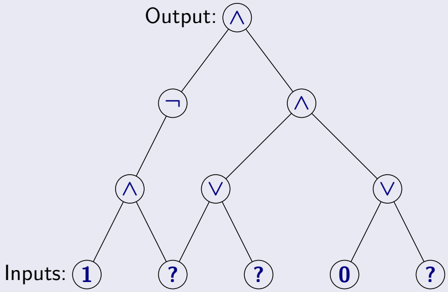


1. Input vertices (without incoming edges) labelled with **0**, **1** or a distinct variable.
2. Every other vertex is labelled $\vee$, $\wedge$ or $\neg$.
3. Single node output vertex with no outgoing edges.

Circuits

Definition

A circuit is a directed *acyclic* graph with



1. Input vertices (without incoming edges) labelled with **0**, **1** or a distinct variable.
2. Every other vertex is labelled $\vee$, $\wedge$ or $\neg$.
3. Single node output vertex with no outgoing edges.

CSAT: Circuit Satisfaction

Definition (Circuit Satisfaction (CSAT).)

Given a circuit as input, is there an assignment to the input variables that causes the output to get value $\mathbf{1}$?

CSAT: Circuit Satisfaction

Definition (Circuit Satisfaction (**CSAT**).)

Given a circuit as input, is there an assignment to the input variables that causes the output to get value **1**?

Claim

CSAT *is in* **NP**.

1. Certificate: Assignment to input variables.
2. Certifier: Evaluate the value of each gate in a topological sort of DAG and check the output gate value.

Circuit SAT vs SAT

CNF formulas are a rather restricted form of Boolean formulas.

Circuits are a much more powerful (and hence easier) way to express Boolean formulas

Circuit SAT vs SAT

CNF formulas are a rather restricted form of Boolean formulas.

Circuits are a much more powerful (and hence easier) way to express Boolean formulas

However they are equivalent in terms of polynomial-time solvability.

Theorem

SAT $\leq_P$ **3SAT** $\leq_P$ **CSAT**.

Theorem

CSAT $\leq_P$ **SAT** $\leq_P$ **3SAT**.

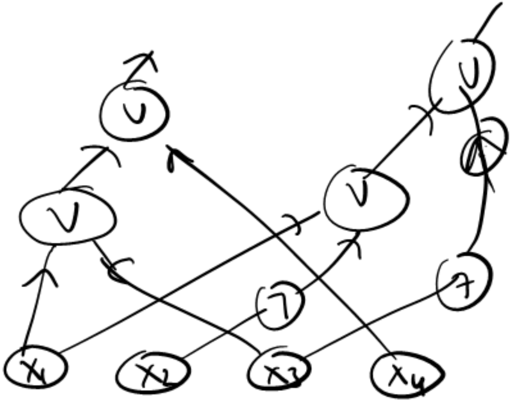
Converting a CNF formula into a Circuit

Given $3\mathrm{CNF}$ formulat φ with n variables and m clauses, create a Circuit C.

- Inputs to C are the n boolean variables $x_1, x_2, \ldots, x_n$
- Use NOT gate to generate literal $\neg x_i$ for each variable x_i
- For each clause $(\ell_1 \vee \ell_2 \vee \ell_3)$ use two OR gates to mimic formula
- Combine the outputs for the clauses using AND gates to obtain the final output

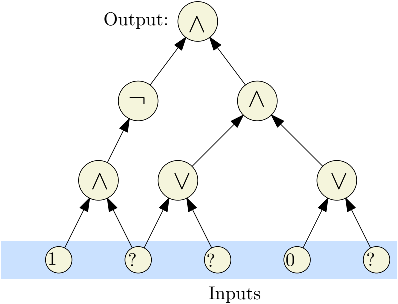
Example

$$\varphi = \Big(x_1 \lor \lor x_3 \lor x_4\Big) \land \Big(x_1 \lor \neg x_2 \lor \neg x_3\Big) \land \Big(\neg x_2 \lor \neg x_3 \lor x_4\Big)$$

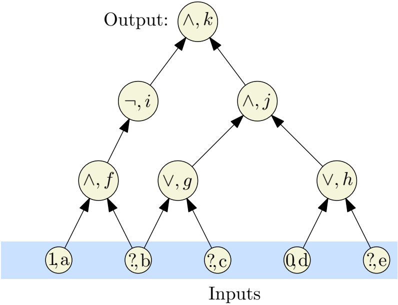


Converting a circuit into a CNF formula

Label the nodes



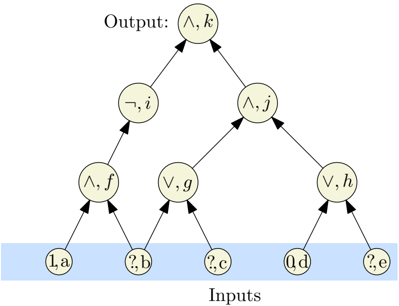
(A) Input circuit



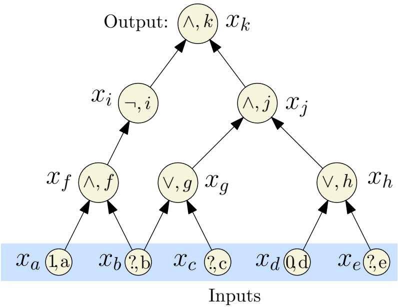
(B) Label the nodes.

Converting a circuit into a CNF formula

Introduce a variable for each node



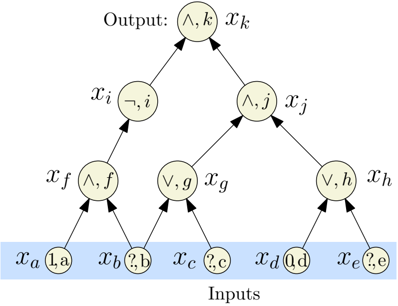
(B) Label the nodes.



(C) Introduce var for each node.

Converting a circuit into a CNF formula

Write a sub-formula for each variable that is true if the var is computed correctly.



(C) Introduce var for each node.

$(\mathbf{x_k})$ (Demand a sat' assignment!)

$(\mathbf{x_k = x_i \wedge x_{\cancel{k}}}$ j$)$

$(\mathbf{x_j = x_g \wedge x_h})$

$(\mathbf{x_i = \neg x_f})$

$(\mathbf{x_h = x_d \vee x_e})$

$(\mathbf{x_g = x_b \vee x_c})$

$(\mathbf{x_f = x_a \wedge x_b})$

$(\mathbf{x_d = 0})$

$(\mathbf{x_a = 1})$

(D) Write a sub-formula for each variable that is true if the var is computed correctly.

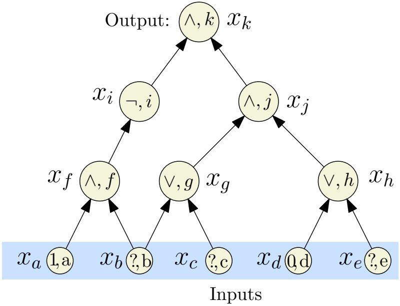
Converting a circuit into a CNF formula

Convert each sub-formula to an equivalent CNF formula

x_k	x_k
$x_k = x_i \wedge x_j$	$(\neg x_k \vee x_i) \wedge (\neg x_k \vee x_j) \wedge (x_k \vee \neg x_i \vee \neg x_j)$
$x_j = x_g \wedge x_h$	$(\neg x_j \vee x_g) \wedge (\neg x_j \vee x_h) \wedge (x_j \vee \neg x_g \vee \neg x_h)$
$x_i = \neg x_f$	$(x_i \vee x_f) \wedge (\neg x_i \vee x_f)$
$x_h = x_d \vee x_e$	$(x_h \vee \neg x_d) \wedge (x_h \vee \neg x_e) \wedge (\neg x_h \vee x_d \vee x_e)$
$x_g = x_b \vee x_c$	$(x_g \vee \neg x_b) \wedge (x_g \vee \neg x_c) \wedge (\neg x_g \vee x_b \vee x_c)$
$x_f = x_a \wedge x_b$	$(\neg x_f \vee x_a) \wedge (\neg x_f \vee x_b) \wedge (x_f \vee \neg x_a \vee \neg x_b)$
$x_d = 0$	$\neg x_d$
$x_a = 1$	x_a

Converting a circuit into a CNF formula

Take the conjunction of all the CNF sub-formulas



$$
\begin{aligned}
& x_k \wedge (\neg x_k \vee x_i) \wedge (\neg x_k \vee x_j) \\
& \wedge (x_k \vee \neg x_i \vee \neg x_j) \wedge (\neg x_j \vee x_g) \\
& \wedge (\neg x_j \vee x_h) \wedge (x_j \vee \neg x_g \vee \neg x_h) \\
& \wedge (x_i \vee x_f) \wedge (\neg x_i \vee x_f) \\
& \wedge (x_h \vee \neg x_d) \wedge (x_h \vee \neg x_e) \\
& \wedge (\neg x_h \vee x_d \vee x_e) \wedge (x_g \vee \neg x_b) \\
& \wedge (x_g \vee \neg x_c) \wedge (\neg x_g \vee x_b \vee x_c) \\
& \wedge (\neg x_f \vee x_a) \wedge (\neg x_f \vee x_b) \\
& \wedge (x_f \vee \neg x_a \vee \neg x_b) \wedge (\neg x_d) \wedge x_a
\end{aligned}
$$

We got a CNF formula that is satisfiable if and only if the original circuit is satisfiable.

Reduction: CSAT $\leq_P$ SAT

1. For each gate (vertex) v in the circuit, create a variable x_v
2. Case $\neg$: v is labeled $\neg$ and has one incoming edge from u (so $x_v = \neg x_u$). In SAT formula generate, add clauses $(x_u \lor x_v)$, $(\neg x_u \lor \neg x_v)$. Observe that

$$x_v = \neg x_u \text{ is true} \quad \Longleftrightarrow \quad \begin{matrix} (x_u \lor x_v) \\ (\neg x_u \lor \neg x_v) \end{matrix} \text{ both true.}$$

1. Case $\lor$: So $x_v = x_u \lor x_w$. In **SAT** formula generated, add clauses $(x_v \lor \neg x_u)$, $(x_v \lor \neg x_w)$, and $(\neg x_v \lor x_u \lor x_w)$. Again, observe that

$$\left(x_v = x_u \lor x_w\right) \text{ is true} \iff \begin{array}{l} (x_v \lor \neg x_u), \\ (x_v \lor \neg x_w), \\ (\neg x_v \lor x_u \lor x_w) \end{array} \text{ all true.}$$

Reduction: CSAT $\leq_P$ SAT

Continued...

1. Case $\wedge$: So $x_v = x_u \wedge x_w$. In **SAT** formula generated, add clauses $(\neg x_v \vee x_u)$, $(\neg x_v \vee x_w)$, and $(x_v \vee \neg x_u \vee \neg x_w)$. Again observe that

$$x_v = x_u \wedge x_w \text{ is true} \iff \begin{array}{l} (\neg x_v \vee x_u), \\ (\neg x_v \vee x_w), \\ (x_v \vee \neg x_u \vee \neg x_w) \end{array} \quad \text{all true.}$$

Reduction: CSAT $\leq_P$ SAT

Continued...

1. If v is an input gate with a fixed value then we do the following. If $x_v = 1$ add clause x_v. If $x_v = 0$ add clause $\neg x_v$
2. Add the clause x_v where v is the variable for the output gate

Correctness of Reduction

Need to show circuit C is satisfiable iff φ_C is satisfiable

$\Rightarrow$ Consider a satisfying assignment a for C

1. Find values of all gates in C under a
2. Give value of gate v to variable x_v; call this assignment a'
3. a' satisfies φ_C (exercise)

$\Leftarrow$ Consider a satisfying assignment a for φ_C

1. Let a' be the restriction of a to only the input variables
2. Value of gate v under a' is the same as value of x_v in a
3. Thus, a' satisfies C

Part III

Proof of Cook-Levin Theorem

Cook-Levin Theorem

Theorem (Cook-Levin)

SAT *is* **NP-Complete**.

We have already seen that **SAT** is in **NP**.

Need to prove that *every* language $L \in \mathbf{NP}$, $L \leq_P \mathbf{SAT}$

Cook-Levin Theorem

Theorem (Cook-Levin)

SAT *is* **NP-Complete**.

We have already seen that **SAT** is in **NP**.

Need to prove that *every* language $L \in \mathbf{NP}$, $L \leq_P \mathbf{SAT}$

Difficulty: Infinite number of languages in **NP**. Must *simultaneously* show a *generic* reduction strategy.

High-level Plan

What does it mean that $L \in \mathbf{NP}$?
$L \in NP$ implies that there is a non-deterministic TM M and polynomial $p()$ such that

$$L = \{x \in \Sigma^* \mid M \text{ accepts } x \text{ in at most } p(|x|) \text{ steps}\}$$

High-level Plan

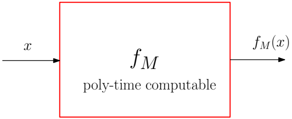
What does it mean that $L \in \mathbf{NP}$?
$L \in NP$ implies that there is a non-deterministic TM M and polynomial $p()$ such that

$$L = \{x \in \Sigma^* \mid M \text{ accepts } x \text{ in at most } p(|x|) \text{ steps}\}$$

We will describe a reduction f_M that depends on M, p such that:

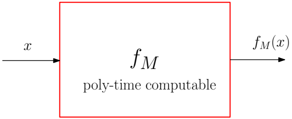
- f_M takes as input a string x and outputs a SAT formula $f_M(x)$
- f_M runs in time polynomial in $|x|$
- $x \in L$ if and only if $f_M(x)$ is satisfiable

Plan continued



$f_M(x)$ is satisfiable if and only if $x \in L$
$f_M(x)$ is satisfiable if and only if non-det M accepts x in $p(|x|)$ steps

Plan continued



$f_M(x)$ is satisfiable if and only if $x \in L$
$f_M(x)$ is satisfiable if and only if non-det M accepts x in $p(|x|)$ steps

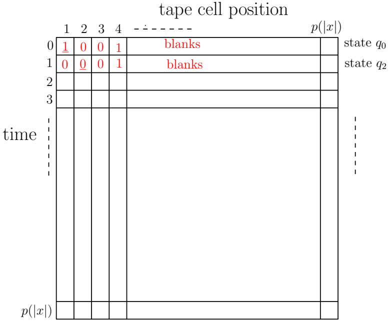
BIG IDEA

- $f_M(x)$ will express "M on input x accepts in $p(|x|)$ steps"
- $f_M(x)$ will encode a computation history of M on x

$f_M(x)$ will be a carefully constructed CNF formulat s.t if we have a satisfying assignment to it, then we will be able to see a complete accepting computation of M on x *down to the last detail* of where the head is, what transistion is chosen, what the tape contents are, at each step.

Tableu of Computation

M runs in time $p(|x|)$ on x. Entire computation of M on x can be represented by a "tableau"



Row i gives contents of all cells at time i

At time 0 tape has input x followed by blanks

Each row long enough to hold all cells M might ever have scanned.

Variable of $f_M(x)$

Four types of variable to describe computation of M on x

- $T(b, h, i)$: tape cell at position h holds symbol b at time i.
  $1 \leq h \leq p(|x|)$, $b \in \Gamma$, $0 \leq i \leq p(|x|)$
- $H(h, i)$: read/write head is at position h at time i.
  $1 \leq h \leq p(|x|)$, $0 \leq i \leq p(|x|)$
- $S(q, i)$ state of M is q at time i $q \in Q$, $0 \leq i \leq p(|x|)$
- $I(j, i)$ instruction number j is executed at time i
  M is non-deterministic, need to specify transitions in some way. Number transitions as $1, 2, \ldots, \ell$ where j'th transition is $< q_j, b_j, q'_j, b'_j, d_j >$ indication $(q'_j, b'_j, d_j) \in \delta(q_j, b_j)$, direction $d_j \in \{-1, 0, 1\}$.

Number of variables is $O(p(|x|)^2)$ where constant in $O()$ hides dependence on fixed machine M.

Notation

Some abbreviations for ease of notation

$\bigwedge_{k=1}^{m} x_k$ means $x_1 \wedge x_2 \wedge \ldots \wedge x_m$

$\bigvee_{k=1}^{m} x_k$ means $x_1 \vee x_2 \vee \ldots \vee x_m$

$\bigoplus(x_1, x_2, \ldots, x_k)$ is a formula that means exactly one of $x_1, x_2, \ldots, x_m$ is true. Can be converted to CNF form

Clauses of $f_M(x)$

$f_M(x)$ is the conjunction of **8** clause groups:

$$f_M(x) = \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4 \wedge \varphi_5 \wedge \varphi_6 \wedge \varphi_7 \wedge \varphi_8$$

where each φ_i is a CNF formula. Described in subsequent slides.
Property: $f_M(x)$ is satisfied iff there is a truth assignment to the variables that simultaneously satisfy $\varphi_1, \ldots, \varphi_8$.

φ_1

φ_1 asserts (is true iff) the variables are set T/F indicating that M starts in state q_0 at time $\mathbf{0}$ with tape contents containing x followed by blanks.

Let $x = a_1 a_2 \ldots a_n$

$\varphi_1 = S(q, 0)$ state at time 0 is q_0

$\bigwedge$ and

$\bigwedge_{h=1}^{n} T(a_h, h, 0)$ at time 0 cells 1 to n have a_1 to a_n

$\bigwedge_{h=n+1}^{p(|x|}) T(B, h, 0)$ at time 0 cells $n+1$ to $p(|x|)$ have blanks

$\bigwedge$ and

$H(1, 0)$ head at time 0 is in position 1

φ_2

φ_2 asserts M in exactly one state at any time i

$$\varphi_2 = \bigwedge_{i=0}^{p(|x|)} \left(\oplus (S(q_0, i), S(q_1, i), \ldots, S(q_{|Q|}, i)) \right)$$

φ_3

φ_3 asserts that each tape cell holds a unique symbol at any given time.

$$\varphi_3 = \bigwedge_{i=0}^{p(|x|)} \bigwedge_{h=1}^{p(|x|)} \oplus(T(b_1, h, i), T(b_2, h, i), \ldots, T(b_{|\Gamma|}, h, i))$$

For each time i and for each cell position h exactly one symbol $b \in \Gamma$ at cell position h at time i

φ_4

φ_4 asserts that the read/write head of M is in exactly one position at any time i

$$\varphi_4 = \bigwedge_{i=0}^{p(|x|)} \left(\oplus \left(H(1,i), H(2,i), \ldots, H(p(|x|), i)\right)\right)$$

φ_5

φ_5 asserts that M accepts

- Let q_a be unique accept state of M
- without loss of generality assume M runs all $p(|x|)$ steps

$$\varphi_5 = S(q_a, p(|x|))$$

State at time $p(|x|)$ is q_a the accept state.

If we don't want to make assumption of running for all steps

$$\varphi_5 = \bigvee_{i=1}^{p(|x|)} S(q_a, i)$$

which means M enters accepts state at some time.

φ_6

φ_6 asserts that M executes a unique instruction at each time

$$\varphi_6 = \bigwedge_{i=0}^{p(|x|)} \oplus(I(1,i), I(2,i), \ldots, I(m,i))$$

where m is max instruction number.

φ_7

φ_7 ensures that variables don't allow tape to change from one moment to next if the read/write head was not there.

"If head is **not** at position h at time i then at time $i + 1$ the symbol at cell h must be unchanged"

$$\varphi_7 = \bigwedge_i \bigwedge_h \bigwedge_{b \neq c} \left(\overline{H(h,i)} \Rightarrow \overline{T(b,h,i) \bigwedge T(c,h,i+1)} \right)$$

since $A \Rightarrow B$ is same as $\neg A \vee B$, rewrite above in CNF form

$$\varphi_7 = \bigwedge_i \bigwedge_h \bigwedge_{b \neq c} (H(h,i) \vee \neg T(b,h,i) \vee \neg T(c,h,i+1))$$

φ_8 asserts that changes in tableu/tape correspond to transitions of M (as Lenny says, this is the big cookie).

Let j'th instruction be $< q_j, b_j, q'_j, b'_j, d_j >$

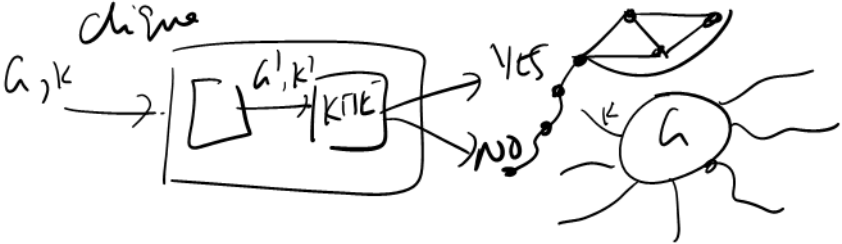
$\varphi_8 = \bigwedge_i \bigwedge_j (I(j,i) \Rightarrow S(q_j, i))$ If instr j executed at time i then state must be correct to do j
$\bigwedge$
$\bigwedge_i \bigwedge_j (I(j,i) \Rightarrow S(q'_j, i+1))$ and at next time unit, state must be the proper next state for instr j
$\bigwedge$
$\bigwedge_i \bigwedge_h \bigwedge_j [(I(j,i) \bigwedge H(h,i)) \Rightarrow T(b_j, h, i)]$ if j was executed and head was at position h, then cell h has correct symbol for j $\bigwedge$
$\bigwedge_i \bigwedge_j \bigwedge_h [(I(j,i) \wedge H(h,i)) \Rightarrow T(b'_j, h, i+1)]$ if j was done then at time i with head at h then at next time step symbol b'_j was indeed written in position h $\bigwedge$
$\bigwedge_i \bigwedge_j \bigwedge_h [(I(j,i) \wedge H(h,i)) \Rightarrow H(h+d_j, i+1)]$ and head is moved properly according to instr j.

Proof of Correctness

(Sketch)

- Given M, x, poly-time algorithm to construct $f_M(x)$
- if $f_M(x)$ is satisfiable then the truth assignment completely specifies an accepting computation of M on x
- if M accepts x then the accepting computation leads to an "obvious" truth assignment to $f_M(x)$. Simply assign the variables according to the state of M and cells at each time i.

Thus M accepts x if and only if $f_M(x)$ is satisfiable



Recap

NP: languages that have polynomial time certifiers/verifiers

A language L is **NP-Complete** iff

- L is in **NP**
- for every L' in **NP**, $L' \leq_P L$

L is **NP-Hard** if for every L' in **NP**, $L' \leq_P L$.

Theorem (Cook-Levin)

SAT *is* **NP-Complete**.

Recap contd

Theorem (Cook-Levin)

SAT *is* **NP-Complete**.

Establish **NP-Complete**ness via reductions:

1. **SAT** is **NP-Complete**.
2. **SAT** $\leq_P$ **3-SAT** and hence 3-SAT is **NP-Complete**.
3. **3-SAT** $\leq_P$ **Independent Set** (which is in **NP**) and hence **Independent Set** is **NP-Complete**.
4. **Clique** is **NP-Complete**
5. **Vertex Cover** is **NP-Complete**
6. **Set Cover** is **NP-Complete**
7. **Hamilton Cycle** and **Hamiltonian Path** are **NP-Complete**
8. **3-Color** is **NP-Complete**

Today

Prove

- **Hamiltonian Cycle** is **NP-Complete**
- **3-Coloring** is **NP-Complete**
- **Subset Sum** is **NP-Complete**

All via reductions from **3-SAT**

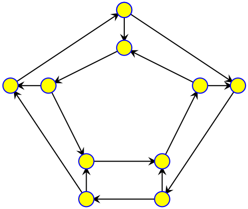
Part I

NP-Completeness of Hamiltonian Cycle

Directed Hamiltonian Cycle

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

Goal Does G have a Hamiltonian cycle?

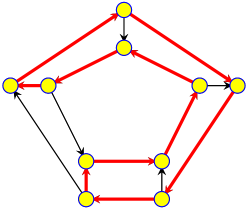


Directed Hamiltonian Cycle

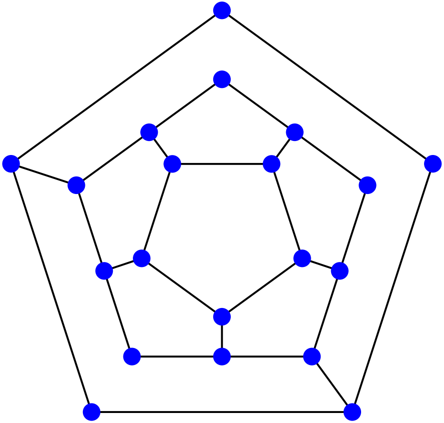
Input Given a directed graph $G = (V, E)$ with n vertices

Goal Does G have a Hamiltonian cycle?

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



Is the following graph Hamiltonianan?



(A) Yes.

(B) No.

Directed Hamiltonian Cycle is **NP-Complete**

- Directed Hamiltonian Cycle is in ***NP***
  - Certificate: Sequence of vertices
  - Certifier: Check if every vertex (except the first) appears exactly once, and that consecutive vertices are connected by a directed edge
- Hardness: We will show **3-SAT $\leq_P$ Directed Hamiltonian Cycle**

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 $\mathcal{A}$

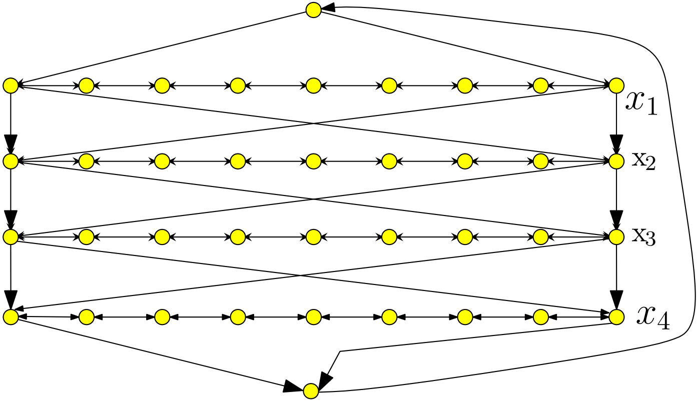
Notation: φ has n variables $x_1, x_2, \ldots, x_n$ and m clauses $C_1, C_2, \ldots, C_m$.

Reduction: First Ideas

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

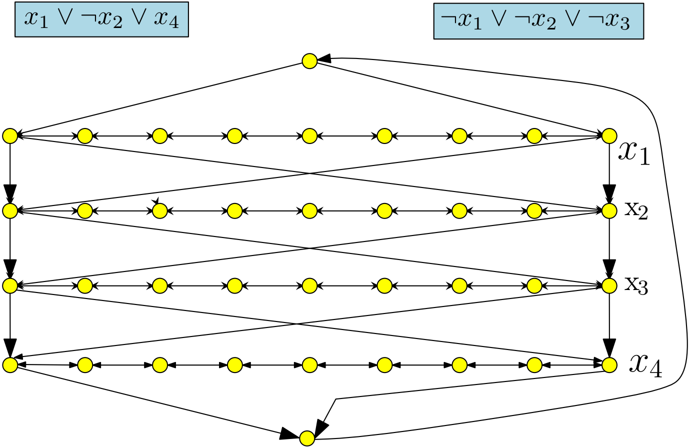
The Reduction: Phase I

- 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 $3m+3$)



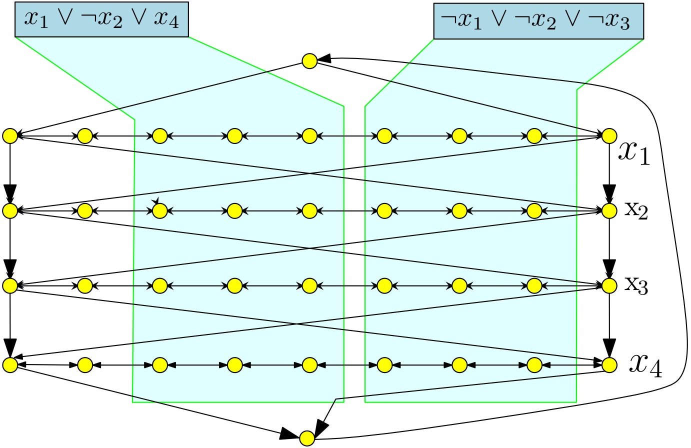
The Reduction: Phase II

- 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.



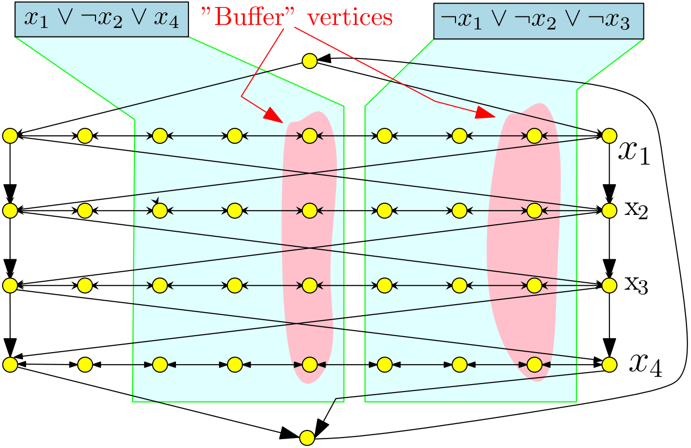
The Reduction: Phase II

- 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.



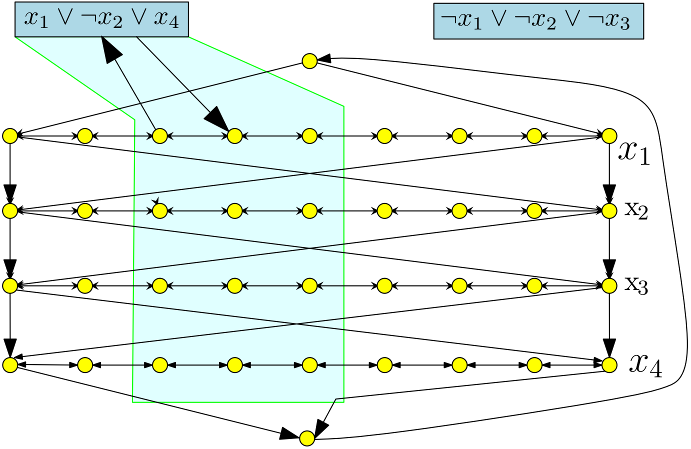
The Reduction: Phase II

- 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.



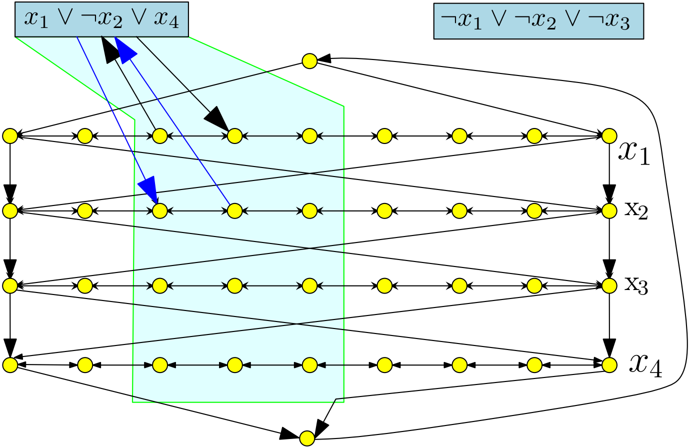
The Reduction: Phase II

- 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.



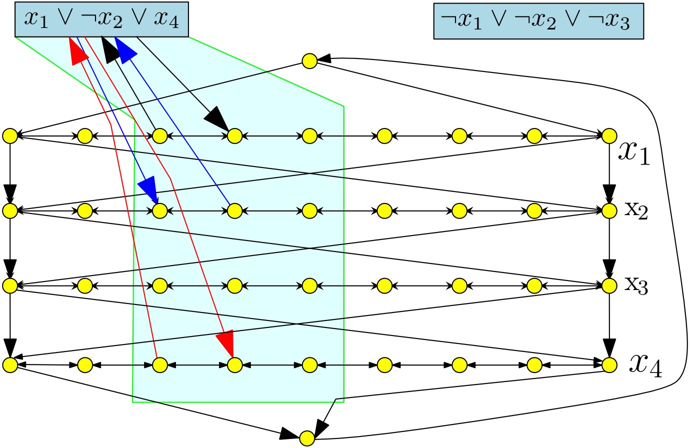
The Reduction: Phase II

- 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.



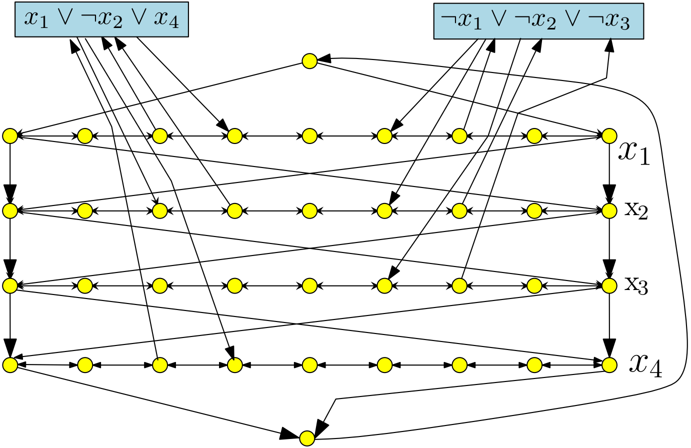
The Reduction: Phase II

- 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: Phase 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

Proposition

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

Proof.

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

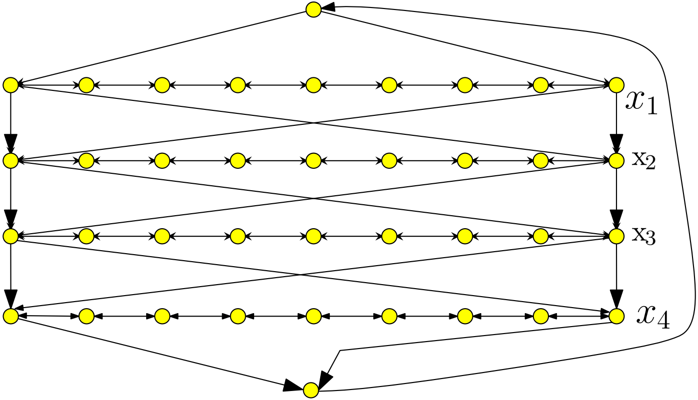
- If $a(x_i) = 1$ then traverse path i from left to right
- If $a(x_i) = 0$ then traverse path i from right to left
- For each clause, path of at least one variable is in the "right" direction to splice in the node corresponding to clause $\square$

Hamiltonian Cycle $\Rightarrow$ Satisfying assignment

Suppose Π is a Hamiltonian cycle in G_φ

- 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

Example



Hamiltonian Cycle $\implies$ Satisfying assignment (contd)

- Thus, vertices visited immediately before and after C_i are connected by an edge
- We can remove c_j from cycle, and get Hamiltonian cycle in $G - c_j$
- Consider Hamiltonian cycle in $G - \{c_1, \ldots c_m\}$; it traverses each path in only one direction, which determines the truth assignment

Is covering by cycles hard?

Given a directed graph G, deciding if G can be covered by vertex disjoint cycles (each of length at least two) is

(A) **NP-Hard**.

(B) **NP-Complete**.

(C) **P**.

(D) IDK.

Hamiltonian Cycle

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)?*

NP-Completeness

Theorem

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

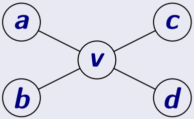
Proof.

- 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

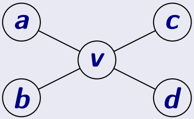


Reduction Sketch

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}

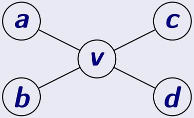


Reduction Sketch

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})

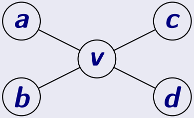


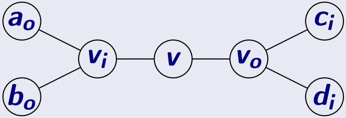
Reduction Sketch

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 directed 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

Exercise: Modify the reduction from **3-SAT** to **Hamilton cycle** to prove that **3-SAT** reduces to **Hamilton path**.

Exercise: Also prove that **Hamilton path** in undirected graphs is **NP-Complete**.

Part II

NP-Completeness of Graph Coloring

Graph Coloring

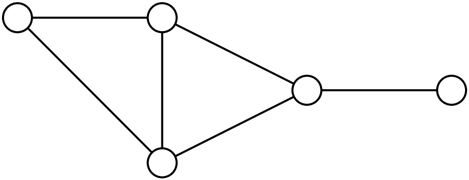
Problem: Graph Coloring

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?

Graph 3-Coloring

Problem: 3 Coloring

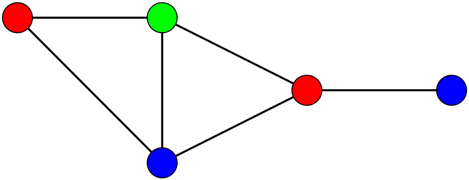
Instance: $G = (V, E)$: Undirected graph.
Question: Can the vertices of the graph be colored using **3** colors so that vertices connected by an edge do not get the same color?



Graph 3-Coloring

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 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.

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!

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 **BFS** (we saw this earlier).

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?

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*

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)

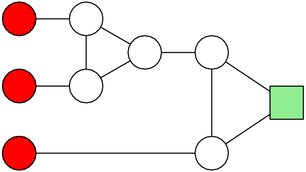
- Breakup a frequency range $[a, b]$ into disjoint *bands* of frequencies $[a_0, b_0], [a_1, b_1], \ldots, [a_k, b_k]$
- Each cell phone tower (simplifying) gets one band
- Constraint: nearby towers cannot be assigned same band, otherwise signals will interference

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.

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 the two nodes are already colored as indicated).

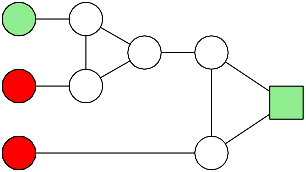


(A) Yes.

(B) No.

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 the two nodes are already colored as indicated).



(A) Yes.

(B) No.

3-Coloring is NP-Complete

- **3-Coloring** is in **NP**.
  - Certificate: for each node a color from $\{1, 2, 3\}$.
  - Certifier: 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., $\mathbf{3}\mathrm{CNF}$ 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.

Reduction Idea

Start with **3SAT** formula (i.e., $3\mathrm{CNF}$ 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

Reduction Idea

Start with **3SAT** formula (i.e., $3\mathrm{CNF}$ 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

Reduction Idea

Start with **3SAT** formula (i.e., $3\mathrm{CNF}$ 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

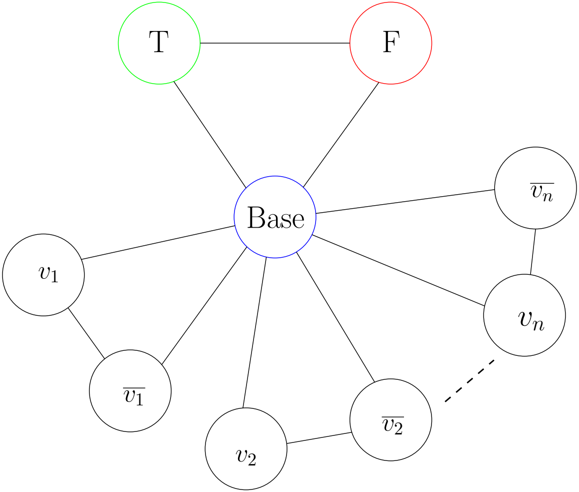
- need to establish truth assignment for $x_1, \ldots, x_n$ via colors for some nodes in G_φ.
- create triangle with node True, False, Base
- for each variable x_i two nodes v_i and $\bar{v}_i$ connected in a triangle with common Base
- If graph is 3-colored, either v_i or $\bar{v}_i$ gets the same color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

Reduction Idea

Start with **3SAT** formula (i.e., $3\mathrm{CNF}$ 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)

Figure

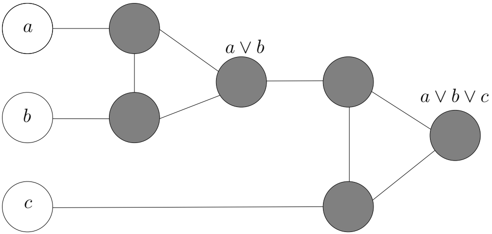


Clause Satisfiability Gadget

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:



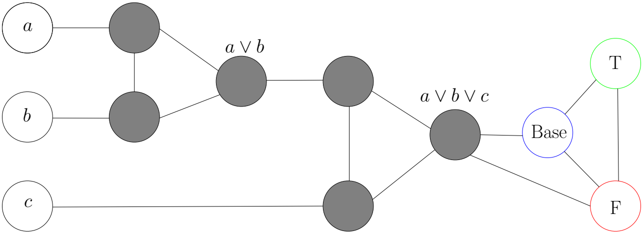
OR-Gadget Graph

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

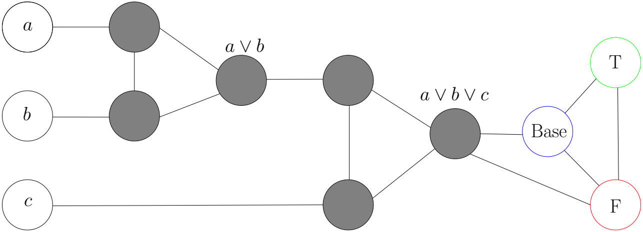
Property: if one of a, b, c is colored True then OR-gadget can be 3-colored such that output node of OR-gadget is colored True.

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



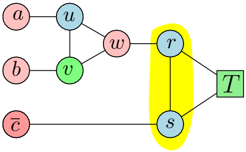
Reduction



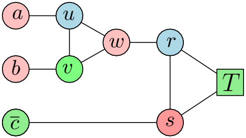
Claim

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

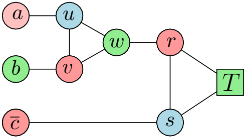
3 coloring of the clause gadget



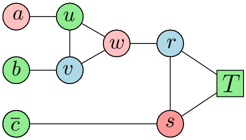
FFF - **BAD**



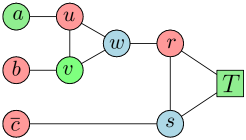
FFT



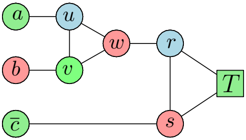
FTF



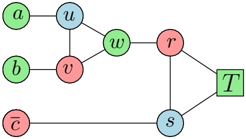
FTT



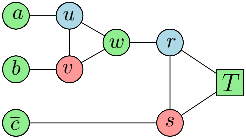
TFF



TFT



TTF

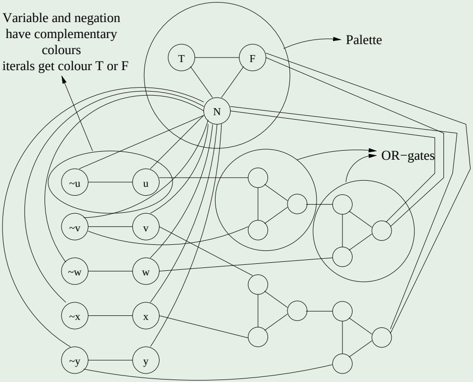


TTT

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 $C_j = (a \lor b \lor c)$ 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

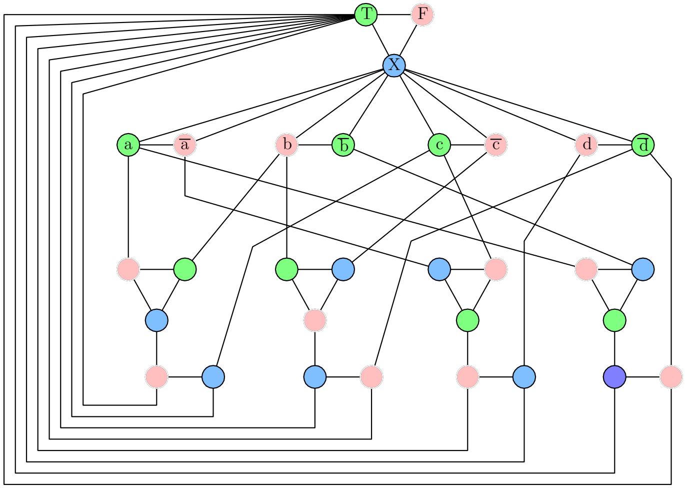
- if x_i is assigned True, color v_i True and $\bar{v}_i$ False
- for each clause $C_j = (a \vee b \vee c)$ at least one of a, b, c is colored True. OR-gadget for C_j can be 3-colored such that output is True.

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



Part III

Hardness of **Subset Sum**

Subset Sum

Problem: Subset Sum

Instance: S - set of positive integers, t: - an integer number (Target)
Question: Is there a subset $X \subseteq S$ such that $\sum_{x \in X} x = t$?

Claim

Subset Sum *is* **NP-Complete**.

Vec Subset Sum

We will prove following problem is **NP-Complete**...

Problem: Vec Subset Sum

Instance: S - set of n vectors of dimension k, each vector has non-negative numbers for its coordinates, and a target vector $\overrightarrow{t}$.
Question: Is there a subset $X \subseteq S$ such that $\sum_{\overrightarrow{x} \in X} \overrightarrow{x} = \overrightarrow{t}$?

Reduction from **3SAT**.

Vec Subset Sum

Handling a single clause

Think about vectors as being lines in a table.

First gadget

Selecting between two lines.

Target	??	??	01	???
a_1	??	??	01	??
a_2	??	??	01	??

Two rows for every variable x: selecting either $x = \mathbf{0}$ or $x = \mathbf{1}$.

Handling a clause...

We will have a column for every clause...

numbers	...	$C \equiv a \vee b \vee \overline{c}$	...
a	...	01	...
$\overline{a}$	...	00	...
b	...	01	...
$\overline{b}$	...	00	...
c	...	00	...
$\overline{c}$	...	01	...
C fix-up 1	000	07	000
C fix-up 2	000	08	000
C fix-up 3	000	09	000
TARGET		10	

3SAT to Vec Subset Sum

numbers	$a \vee \overline{a}$	$b \vee \overline{b}$	$c \vee \overline{c}$	$d \vee \overline{d}$	$D \equiv \overline{b} \vee c \vee \overline{d}$	$C \equiv a \vee b \vee \overline{c}$
a	1	0	0	0	00	01
$\overline{a}$	1	0	0	0	00	00
b	0	1	0	0	00	01
$\overline{b}$	0	1	0	0	01	00
c	0	0	1	0	01	00
$\overline{c}$	0	0	1	0	00	01
d	0	0	0	1	00	00
$\overline{d}$	0	0	0	1	01	01
C fix-up 1	0	0	0	0	00	07
C fix-up 2	0	0	0	0	00	08
C fix-up 3	0	0	0	0	00	09
D fix-up 1	0	0	0	0	07	00
D fix-up 2	0	0	0	0	08	00
D fix-up 3	0	0	0	0	09	00
TARGET	1	1	1	1	10	10

Vec Subset Sum to Subset Sum

numbers
010000000001
010000000000
000100000001
000100000100
000001000100
000001000001
000000010000
000000010101
000000000007
000000000008
000000000009
000000000700
000000000800
000000000900

Other NP-Complete Problems

- **3-Dimensional Matching**
- **3-Partition**

Read book.

Subset Sum and Knapsack

Knapsack: Given n items with item i having non-negative integer size s_i and non-negative integer profit p_i, a knapsack of capacity B, and a target profit P, is there a subset S of items that can be packed in the knapsack and the profit of S is at least P?

Exercise: Show **Knapsack** is **NP-Complete** via reduction from **Subset Sum**

Subset Sum and Knapsack

Subset Sum can be solved in $O(nB)$ time using dynamic programming (exercise).

Subset Sum and Knapsack

Subset Sum can be solved in $O(nB)$ time using dynamic programming (exercise).

Implies that problem is hard only when numbers $a_1, a_2, \ldots, a_n$ are exponentially large compared to n. That is, each a_i requires polynomial in n bits.

Number problems of the above type are said to be **weakly NP-Complete**.

Number problems which are **NP-Complete** even when the numbers are written in unary are **strongly NP-Complete**.

A Strongly NP-Complete Number Problem

3-Partition: Given $3n$ numbers $a_1, a_2, \ldots, a_{3n}$ and target B can the numbers be partitioned into n groups of 3 each such that the sum of numbers in each group is exactly B?

Can further assume that each number a_i is between $B/3$ and $2B/3$.

Can reduce **3-D-Matching** to **3-Partition** in polynomial time such that each number a_i can be written in unary.

Need to Know **NP-Complete** Problems

- **SAT** and **3-SAT**
- **Independent Set**
- **Vertex Cover**
- **Clique**
- **Set Cover**
- **Hamiltonian Cycle** in Directed/Undirected Graphs
- **3-Coloring**
- **3-D Matching**
- **Subset Sum** and **Knapsack**