NP and NP Completeness

Lecture 23

Wednesday, April 29, 2020

LaTeXed: January 19, 2020 04:28

Part I

NP-Completeness

NP: Non-deterministic polynomial

Definition

A decision problem is in **NP**, *if it has a polynomial time certifier, for all the all the YES instances.*

Definition

A decision problem is in **co-NP**, *if it has a polynomial time certifier, for all the all the NO instances.*

Example

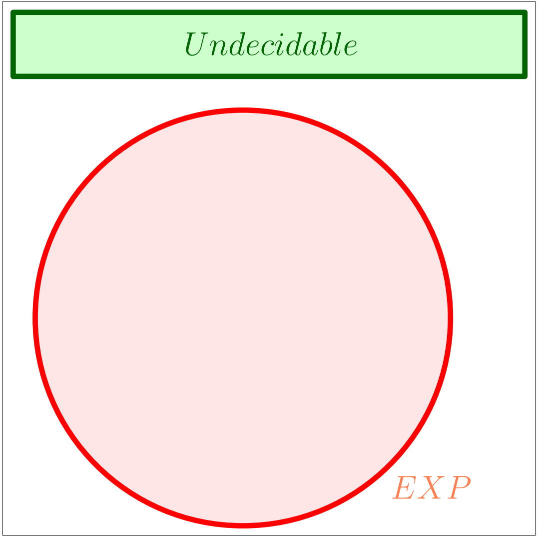
1. **3SAT** is in **NP**.
2. But **Not3SAT** is in **co-NP**.

In the beginning...

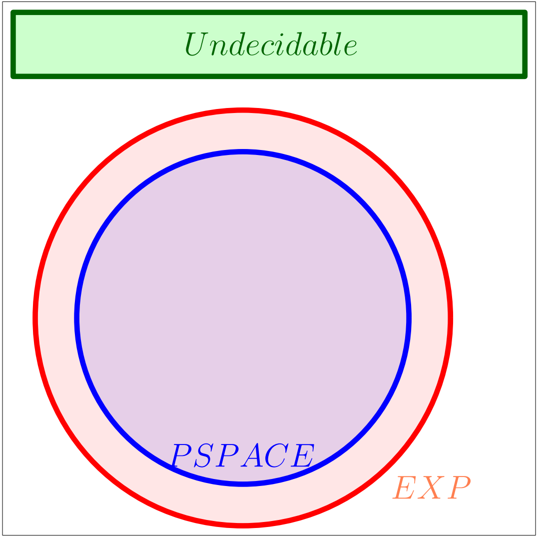
In the beginning...

$Undecidable$

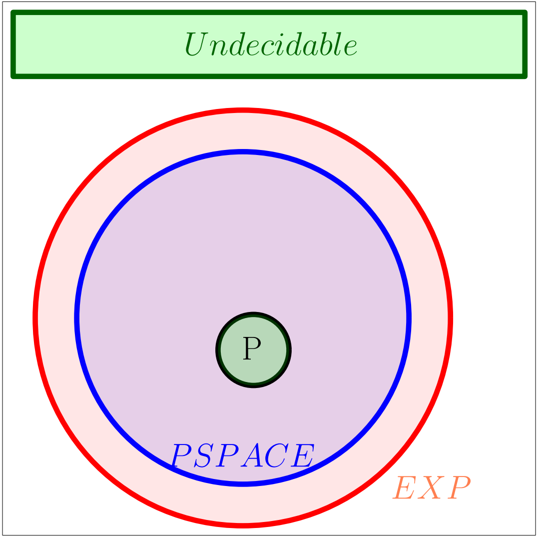
In the beginning...



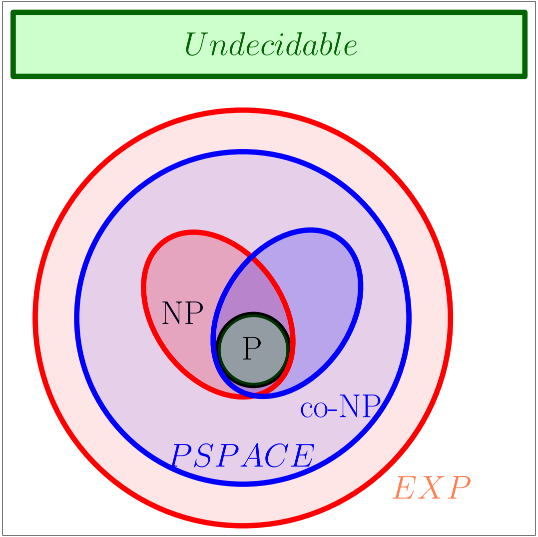
In the beginning...



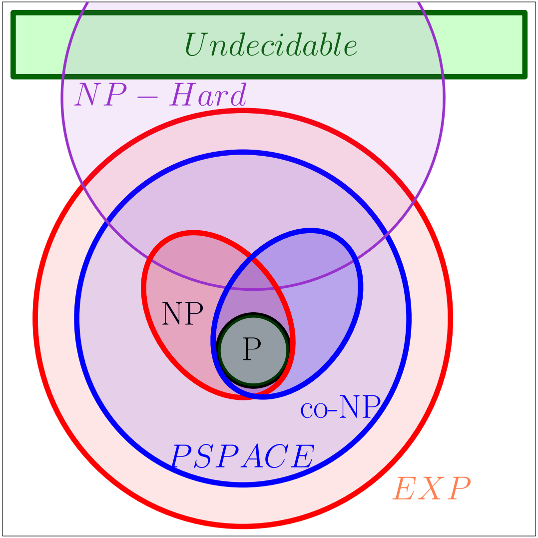
In the beginning...



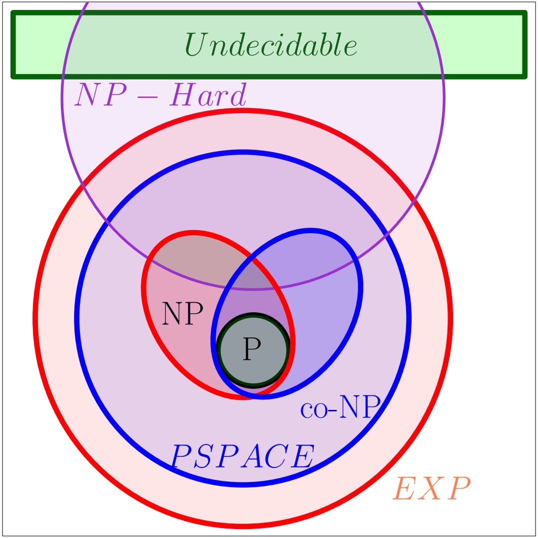
In the beginning...



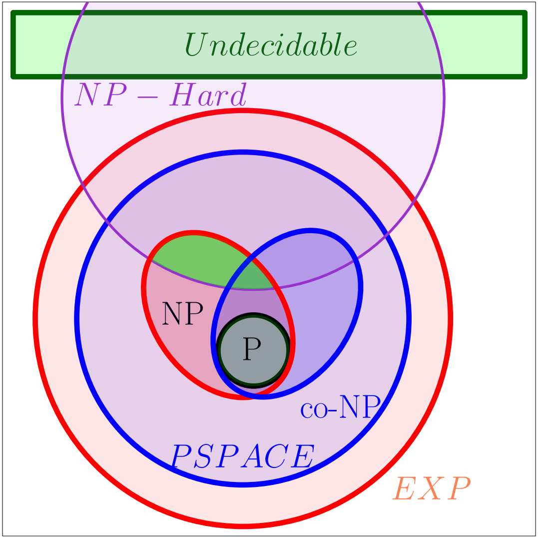
In the beginning...



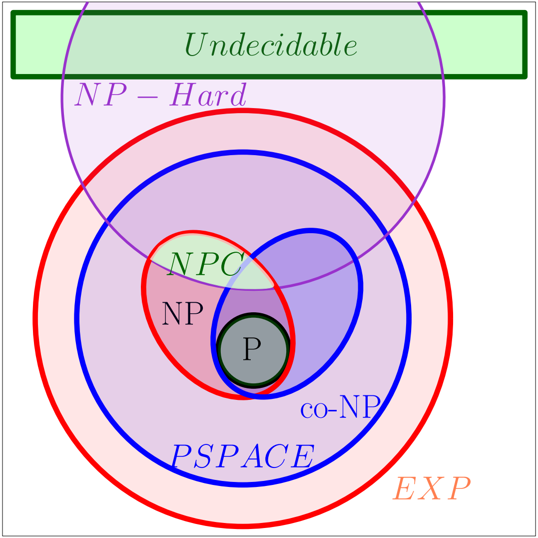
In the beginning...



In the beginning...



In the beginning...



“Hardest” Problems

Question

What is the hardest problem in **NP**? How do we define it?

Towards a definition

1. Hardest problem must be in **NP**.
2. Hardest problem must be at least as “difficult” as every other problem in **NP**.

NP-Complete Problems

Definition

A problem X is said to be **NP-Complete** if

1. $X \in \mathbf{NP}$, and
2. (Hardness) For any $Y \in \mathbf{NP}$, $\mathbf{Y} \leq_P \mathbf{X}$.

Solving NP-Complete Problems

Proposition

Suppose X is **NP-Complete***. Then X can be solved in polynomial time if and only if* **P = NP***.*

Proof.

$\Rightarrow$ Suppose X can be solved in polynomial time

1. Let $Y \in$ **NP**. We know $Y \leq_P X$.
2. We showed that if $Y \leq_P X$ and X can be solved in polynomial time, then Y can be solved in polynomial time.
3. Thus, every problem $Y \in$ **NP** is such that $Y \in P$; $NP \subseteq P$.
4. Since **P** $\subseteq$ **NP**, we have **P = NP**.

$\Leftarrow$ Since **P = NP**, and $X \in$ **NP**, we have a polynomial time algorithm for X. $\square$

NP-Hard Problems

Definition

A problem X is said to be **NP-Hard** if

1. (Hardness) For any $Y \in \mathbf{NP}$, we have that $\mathbf{Y} \leq_P \mathbf{X}$.

An **NP-Hard** problem need not be in **NP**!

Example: Halting problem is **NP-Hard** (why?) but not **NP-Complete**.

Consequences of proving NP-Completeness

If X is **NP-Complete**

1. Since we believe $\mathbf{P} \neq \mathbf{NP}$,
2. and solving X implies $\mathbf{P} = \mathbf{NP}$.

X is unlikely to be efficiently solvable.

Consequences of proving NP-Completeness

If X is **NP-Complete**

1. Since we believe $\mathbf{P} \neq \mathbf{NP}$,
2. and solving X implies $\mathbf{P} = \mathbf{NP}$.

X is unlikely to be efficiently solvable.

At the very least, many smart people before you have failed to find an efficient algorithm for X.

Consequences of proving **NP-Complete**ness

If X is **NP-Complete**

1. Since we believe **P** $\neq$ **NP**,
2. and solving X implies **P** $=$ **NP**.

X is unlikely to be efficiently solvable.

At the very least, many smart people before you have failed to find an efficient algorithm for X.

Consequences of proving NP-Completeness

If X is **NP-Complete**

1. Since we believe $\mathbf{P} \neq \mathbf{NP}$,
2. and solving X implies $\mathbf{P} = \mathbf{NP}$.

X is unlikely to be efficiently solvable.

At the very least, many smart people before you have failed to find an efficient algorithm for X.
(This is proof by mob opinion — take with a grain of salt.)

NP-Complete Problems

Question

Are there any problems that are **NP-Complete**?

Answer

Yes! Many, many problems are **NP-Complete**.

Cook-Levin Theorem

Theorem (Cook-Levin)

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

Cook-Levin Theorem

Theorem (Cook-Levin)

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

Need to show

1. **SAT** is in **NP**.
2. *every* **NP** problem X reduces to **SAT**.

Will see proof in next lecture.

Steve Cook won the Turing award for his theorem.

Proving that a problem X is NP-Complete

To prove X is **NP-Complete**, show

1. Show that X is in **NP**.
2. Give a polynomial-time reduction *from* a known **NP-Complete** problem such as **SAT** *to* X

Proving that a problem X is NP-Complete

To prove X is **NP-Complete**, show

1. Show that X is in **NP**.
2. Give a polynomial-time reduction *from* a known **NP-Complete** problem such as **SAT** *to* X

SAT $\leq_P X$ implies that every **NP** problem $Y \leq_P X$. Why?

Proving that a problem X is NP-Complete

To prove X is **NP-Complete**, show

1. Show that X is in **NP**.
2. Give a polynomial-time reduction *from* a known **NP-Complete** problem such as **SAT** *to* X

SAT $\leq_P X$ implies that every **NP** problem $Y \leq_P X$. Why?
Transitivity of reductions:

$Y \leq_P SAT$ and $SAT \leq_P X$ and hence $Y \leq_P X$.

3-SAT is NP-Complete

- **3-SAT** is in NP
- **SAT** $\leq_P$ **3-SAT** as we saw

NP-Completeness via Reductions

1. **SAT** is **NP-Complete** due to Cook-Levin theorem
2. **SAT** $\leq_P$ **3-SAT**
3. **3-SAT** $\leq_P$ **Independent Set**
4. **Independent Set** $\leq_P$ **Vertex Cover**
5. **Independent Set** $\leq_P$ **Clique**
6. **3-SAT** $\leq_P$ **3-Color**
7. **3-SAT** $\leq_P$ **Hamiltonian Cycle**

NP-Completeness via Reductions

1. **SAT** is **NP-Complete** due to Cook-Levin theorem
2. **SAT** $\leq_P$ **3-SAT**
3. **3-SAT** $\leq_P$ **Independent Set**
4. **Independent Set** $\leq_P$ **Vertex Cover**
5. **Independent Set** $\leq_P$ **Clique**
6. **3-SAT** $\leq_P$ **3-Color**
7. **3-SAT** $\leq_P$ **Hamiltonian Cycle**

Hundreds and thousands of different problems from many areas of science and engineering have been shown to be **NP-Complete**.

A surprisingly frequent phenomenon!

Part II

Reducing **3-SAT** to **Independent Set**

Independent Set

Problem: Independent Set

Instance: A graph G, integer k. **Question:** Is there an independent set in G of size k?

3SAT $\leq_P$ Independent Set

The reduction **3SAT $\leq_P$ Independent Set**

Input: Given a $3\mathrm{CNF}$ formula φ

Goal: Construct a graph G_φ and number k such that G_φ has an independent set of size k if and only if φ is satisfiable.

3SAT $\leq_P$ Independent Set

The reduction 3SAT $\leq_P$ Independent Set

Input: Given a $3\mathrm{CNF}$ formula φ
Goal: Construct a graph G_φ and number k such that G_φ has an independent set of size k if and only if φ is satisfiable.
G_φ should be constructable in time polynomial in size of φ

3SAT $\leq_P$ Independent Set

The reduction 3SAT $\leq_P$ Independent Set

Input: Given a $3\mathrm{CNF}$ formula φ
Goal: Construct a graph G_φ and number k such that G_φ has an independent set of size k if and only if φ is satisfiable.
G_φ should be constructable in time polynomial in size of φ

Importance of reduction: Although **3SAT** is much more expressive, it can be reduced to a seemingly specialized Independent Set problem.

Notice: We handle only $3\mathrm{CNF}$ formulas – reduction would not work for other kinds of boolean formulas.

Interpreting **3SAT**

There are two ways to think about **3SAT**

Interpreting 3SAT

There are two ways to think about **3SAT**

1. Find a way to assign 0/1 (false/true) to the variables such that the formula evaluates to true, that is each clause evaluates to true.

Interpreting **3SAT**

There are two ways to think about **3SAT**

1. Find a way to assign 0/1 (false/true) to the variables such that the formula evaluates to true, that is each clause evaluates to true.
2. Pick a literal from each clause and find a truth assignment to make all of them true

Interpreting 3SAT

There are two ways to think about **3SAT**

1. Find a way to assign 0/1 (false/true) to the variables such that the formula evaluates to true, that is each clause evaluates to true.
2. Pick a literal from each clause and find a truth assignment to make all of them true. You will fail if two of the literals you pick are in conflict, i.e., you pick x_i and $\neg x_i$

We will take the second view of **3SAT** to construct the reduction.

The Reduction

1. G_φ will have one vertex for each literal in a clause

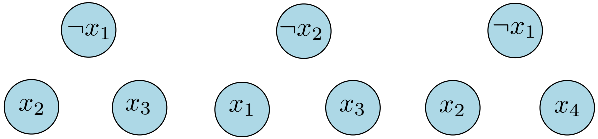


Figure: Graph for
$\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

The Reduction

1. G_φ will have one vertex for each literal in a clause
2. Connect the 3 literals in a clause to form a triangle; the independent set will pick at most one vertex from each clause, which will correspond to the literal to be set to true

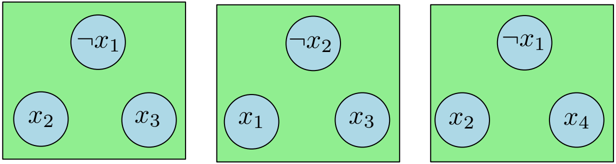


Figure: Graph for
$\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

The Reduction

1. G_φ will have one vertex for each literal in a clause
2. Connect the 3 literals in a clause to form a triangle; the independent set will pick at most one vertex from each clause, which will correspond to the literal to be set to true

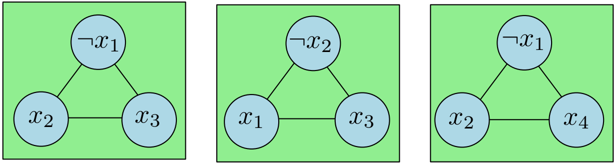


Figure: Graph for
$\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

The Reduction

1. G_φ will have one vertex for each literal in a clause
2. Connect the 3 literals in a clause to form a triangle; the independent set will pick at most one vertex from each clause, which will correspond to the literal to be set to true
3. Connect 2 vertices if they label complementary literals; this ensures that the literals corresponding to the independent set do not have a conflict

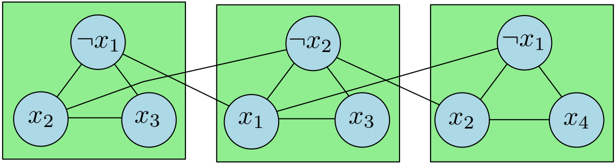


Figure: Graph for
$\varphi = (\neg x_1 \lor x_2 \lor x_3) \land (x_1 \lor \neg x_2 \lor x_3) \land (\neg x_1 \lor x_2 \lor x_4)$

The Reduction

1. G_φ will have one vertex for each literal in a clause
2. Connect the 3 literals in a clause to form a triangle; the independent set will pick at most one vertex from each clause, which will correspond to the literal to be set to true
3. Connect 2 vertices if they label complementary literals; this ensures that the literals corresponding to the independent set do not have a conflict
4. Take k to be the number of clauses

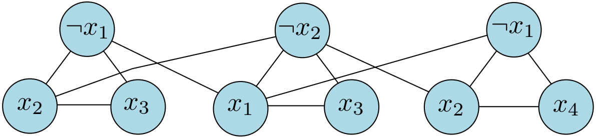


Figure: Graph for
$\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

Correctness

Proposition

φ is satisfiable iff G_φ has an independent set of size k ($=$ number of clauses in φ).

Proof.

$\Rightarrow$ Let a be the truth assignment satisfying φ

Correctness

Proposition

φ is satisfiable iff G_φ has an independent set of size k ($=$ number of clauses in φ).

Proof.

$\Rightarrow$ Let a be the truth assignment satisfying φ

1. Pick one of the vertices, corresponding to true literals under a, from each triangle. This is an independent set of the appropriate size. Why? □

Correctness (contd)

Proposition

φ is satisfiable iff G_φ has an independent set of size k ($=$ number of clauses in φ).

Proof.

$\Leftarrow$ Let S be an independent set of size k

1. S must contain *exactly* one vertex from each clause
2. S cannot contain vertices labeled by conflicting literals
3. Thus, it is possible to obtain a truth assignment that makes in the literals in S true; such an assignment satisfies one literal in every clause $\square$

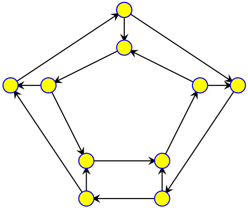
Part III

NPCompleteness of Hamiltonian Cycle

Directed Hamiltonian Cycle

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

Goal Does G have a Hamiltonian cycle?

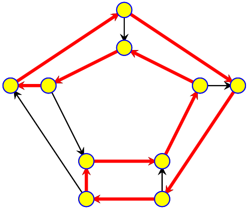


Directed Hamiltonian Cycle

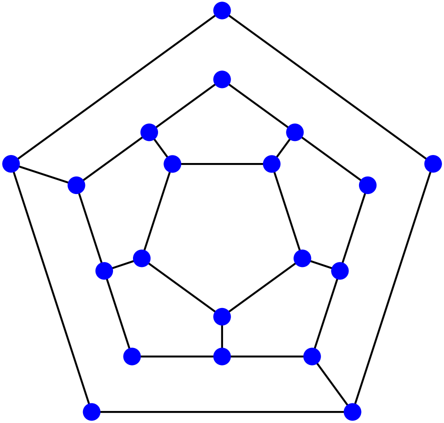
Input Given a directed graph $G = (V, E)$ with n vertices

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 $\boldsymbol{NP}$: exercise
- 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 $\mathcal{A}$

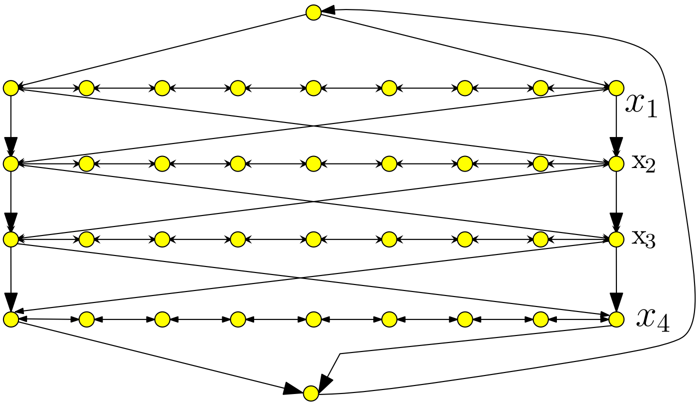
Notation: φ has n variables $x_1, x_2, \ldots, x_n$ and m clauses $C_1, C_2, \ldots, C_m$.

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 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 $3m+3$)

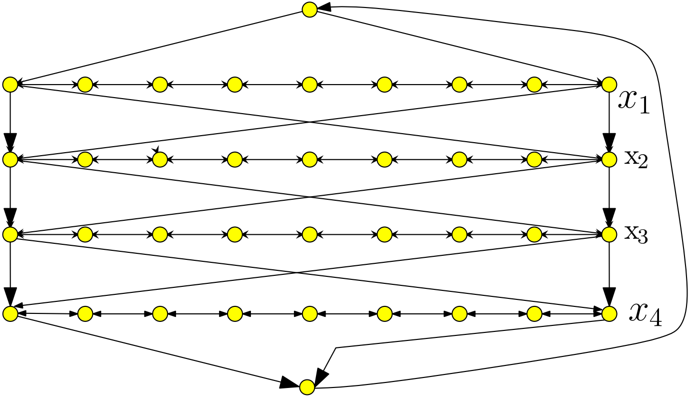


The Reduction: Phase II

- 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.

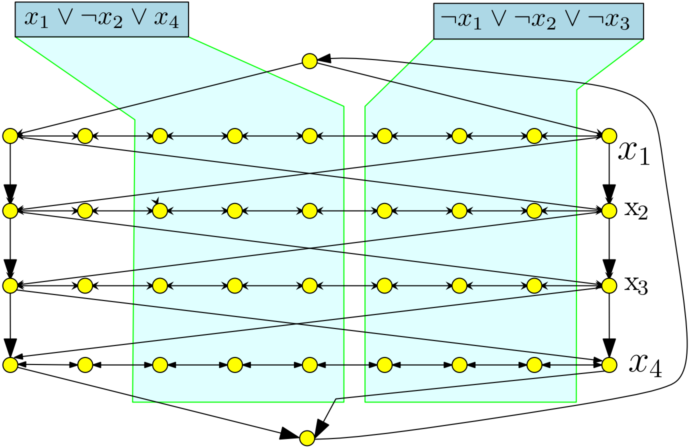
$x_1 \vee \neg x_2 \vee x_4$

$\neg x_1 \vee \neg x_2 \vee \neg x_3$



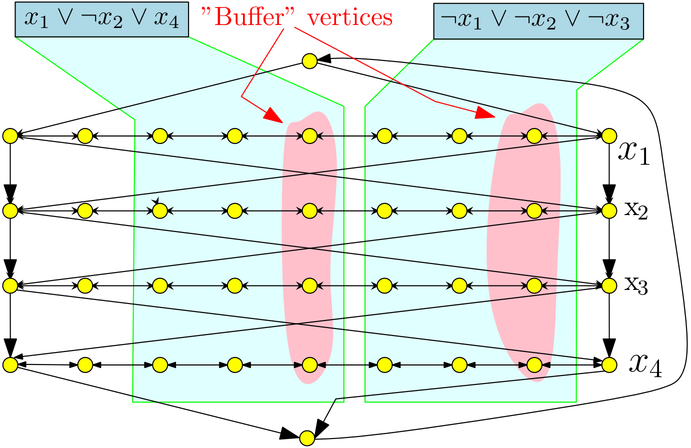
The Reduction: Phase II

- 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.



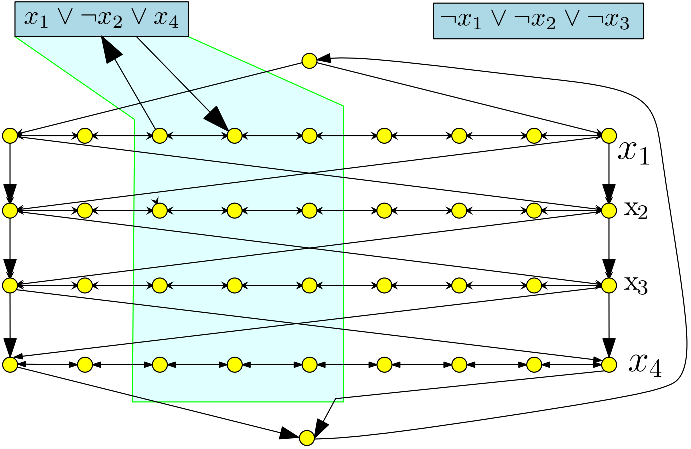
The Reduction: Phase II

- 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.



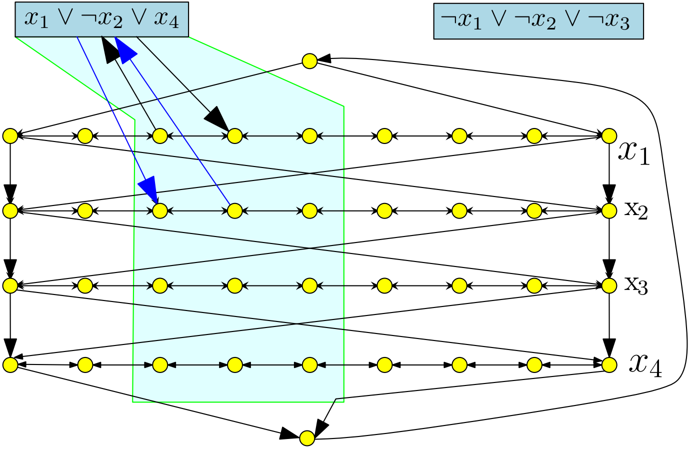
The Reduction: Phase II

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



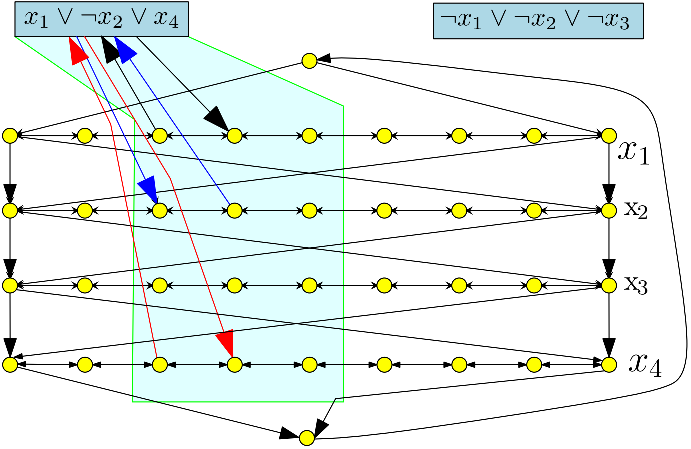
The Reduction: Phase II

- 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.



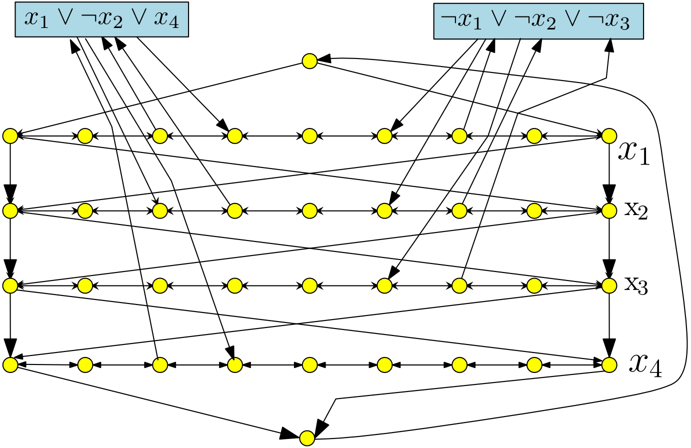
The Reduction: Phase II

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

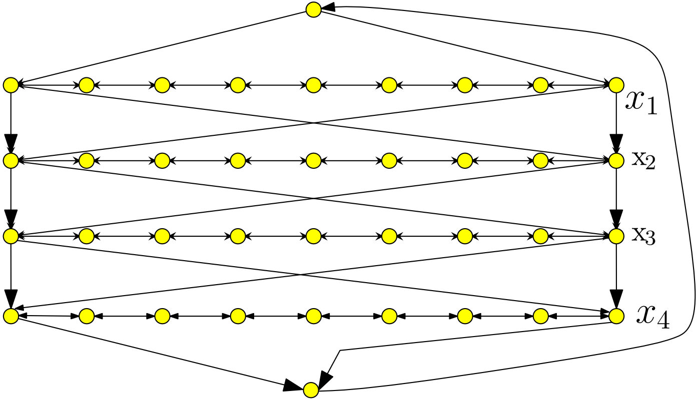
- If $a(x_i) = 1$ then traverse path i from left to right
- If $a(x_i) = 0$ then traverse path i from right to left
- For each clause, path of at least one variable is in the "right" direction to splice in the node corresponding to clause □

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 $\Longrightarrow$ 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

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***.*

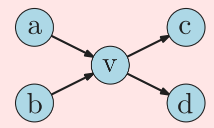
Proof.

- 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

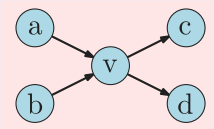


Reduction Sketch

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}

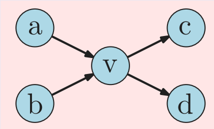


Reduction Sketch

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})

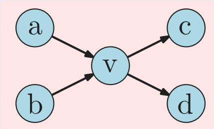


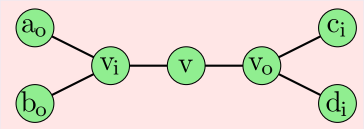
Reduction Sketch

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)

Part IV

NP-Completeness of Graph Coloring

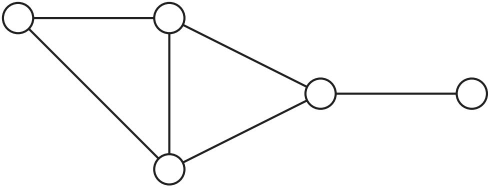
Problem: Graph Coloring

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?

Graph 3-Coloring

Problem: 3 Coloring

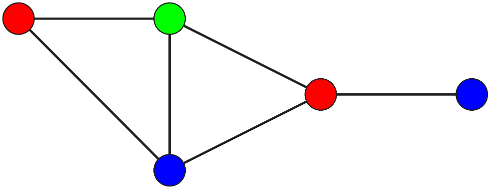
> **Instance:** $G = (V, E)$: Undirected graph.
> **Question:** Can the vertices of the graph be colored using 3 colors so that vertices connected by an edge do not get the same color?



Graph 3-Coloring

Problem: 3 Coloring

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



Graph Coloring

Graph Coloring

1. Observation: If G is colored with k colors then each color class (nodes of same color) form an independent set in G.

Graph Coloring

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

Graph Coloring

1. Observation: If G is colored with k colors then each color class (nodes of same color) form an independent set in G.
2. G can be partitioned into k independent sets iff G is k-colorable.
3. Graph 2-Coloring can be decided in polynomial time.

Graph Coloring

1. Observation: If G is colored with k colors then each color class (nodes of same color) form an independent set in G.
2. G can be partitioned into k independent sets iff G is k-colorable.
3. Graph $\mathbf{2}$-Coloring can be decided in polynomial time.
4. G is $\mathbf{2}$-colorable iff G is bipartite!

Graph Coloring

1. Observation: If G is colored with k colors then each color class (nodes of same color) form an independent set in G.
2. G can be partitioned into k independent sets iff G is k-colorable.
3. Graph 2-Coloring can be decided in polynomial time.
4. G is 2-colorable iff G is bipartite!
5. 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

Class Room Scheduling

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

Class Room Scheduling

1. Given n classes and their meeting times, are k rooms sufficient?
2. Reduce to Graph k-Coloring problem

Class Room Scheduling

1. Given n classes and their meeting times, are k rooms sufficient?
2. Reduce to Graph k-Coloring problem
3. 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

1. Given n classes and their meeting times, are k rooms sufficient?
2. Reduce to Graph k-Coloring problem
3. 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*
4. Exercise: G is k-colorable iff k rooms are sufficient.

Frequency Assignments in Cellular Networks

1. 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

1. 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
2. Problem: given k bands and some region with n towers, is there a way to assign the bands to avoid interference?

Frequency Assignments in Cellular Networks

1. 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
2. Problem: given k bands and some region with n towers, is there a way to assign the bands to avoid interference?
3. Can reduce to k-coloring by creating interference/conflict graph on towers.

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

Reduction Idea

1. φ: Given **3SAT** formula (i.e., $\mathbf{3}\mathrm{CNF}$ formula).

Reduction Idea

1. φ: Given **3SAT** formula (i.e., $\mathbf{3}\mathrm{CNF}$ formula).
2. φ: variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$.

Reduction Idea

1. φ: Given **3SAT** formula (i.e., $\mathbf{3}\mathrm{CNF}$ formula).
2. φ: variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$.
3. Create graph G_φ s.t. G_φ 3-colorable $\iff$ φ satisfiable.
   - encode assignment $x_1, \ldots, x_n$ in colors assigned nodes of G_φ.

Reduction Idea

1. φ: Given **3SAT** formula (i.e., $\mathbf{3}\mathrm{CNF}$ formula).
2. φ: variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$.
3. Create graph G_φ s.t. G_φ 3-colorable $\iff$ φ satisfiable.
   - encode assignment $x_1, \ldots, x_n$ in colors assigned nodes of G_φ.
   - create triangle with node True, False, Base

Reduction Idea

1. φ: Given **3SAT** formula (i.e., $\mathbf{3}\mathrm{CNF}$ formula).
2. φ: variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$.
3. Create graph G_φ s.t. G_φ 3-colorable $\iff$ φ satisfiable.
   - encode assignment $x_1, \ldots, x_n$ in colors assigned nodes of 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

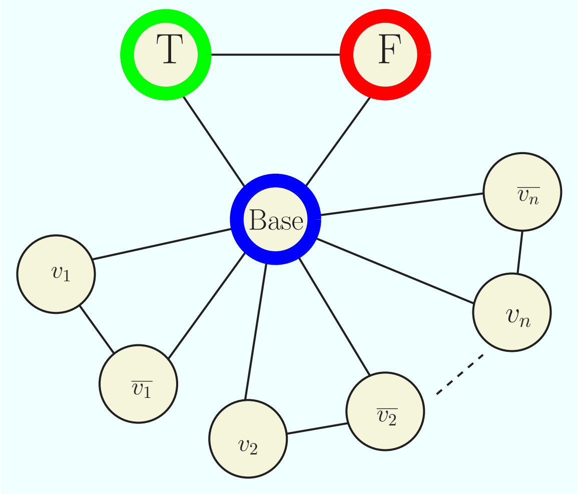
Reduction Idea

1. φ: Given **3SAT** formula (i.e., $3\mathrm{CNF}$ formula).
2. φ: variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$.
3. Create graph G_φ s.t. G_φ 3-colorable $\iff$ φ satisfiable.
    - encode assignment $x_1, \ldots, x_n$ in colors assigned nodes of 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

1. φ: Given **3SAT** formula (i.e., $3\mathrm{CNF}$ formula).
2. φ: variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$.
3. Create graph G_φ s.t. G_φ 3-colorable $\iff$ φ satisfiable.
   - encode assignment $x_1, \ldots, x_n$ in colors assigned nodes of 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)

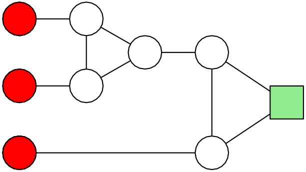
Figure



3 color this gadget.

Clicker question

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).



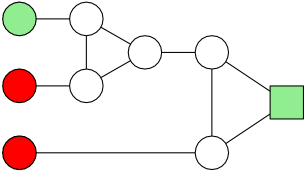
(A) Yes.

(B) No.

3 color this gadget II

Clicker question

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.

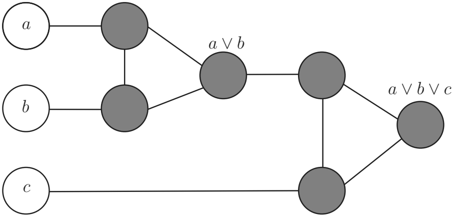
(B) No.

Clause Satisfiability Gadget

1. 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

Clause Satisfiability Gadget

1. 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
2. OR-gadget-graph:



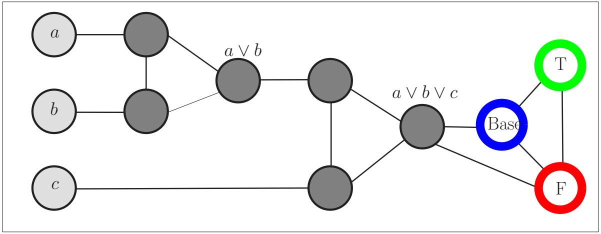
OR-Gadget Graph

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

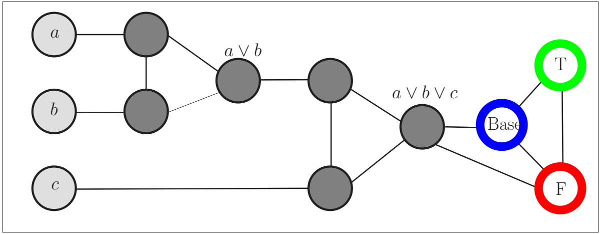
Property: if one of a, b, c is colored True then OR-gadget can be 3-colored such that output node of OR-gadget is colored True.

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 $C_j = (a \lor b \lor c)$, add OR-gadget graph with input nodes a, b, c and connect output node of gadget to both False and Base



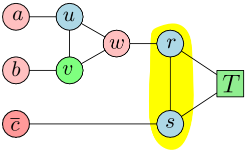
Reduction



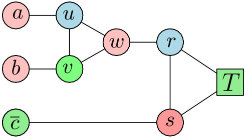
Claim

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

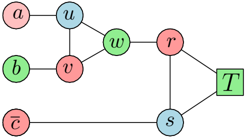
3 coloring of the clause gadget



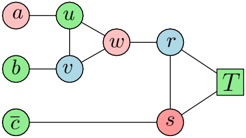
FFF - **BAD**



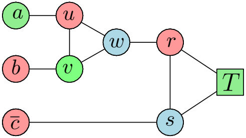
FFT



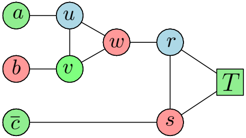
FTF



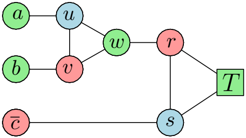
FTT



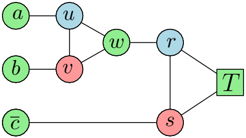
TFF



TFT



TTF

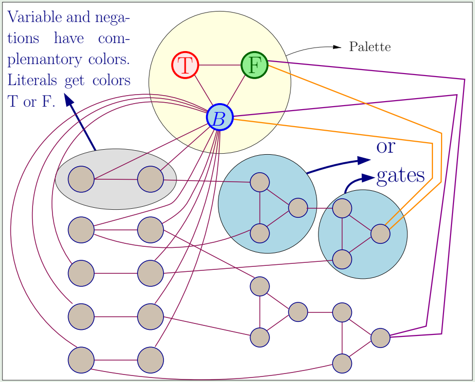


TTT

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

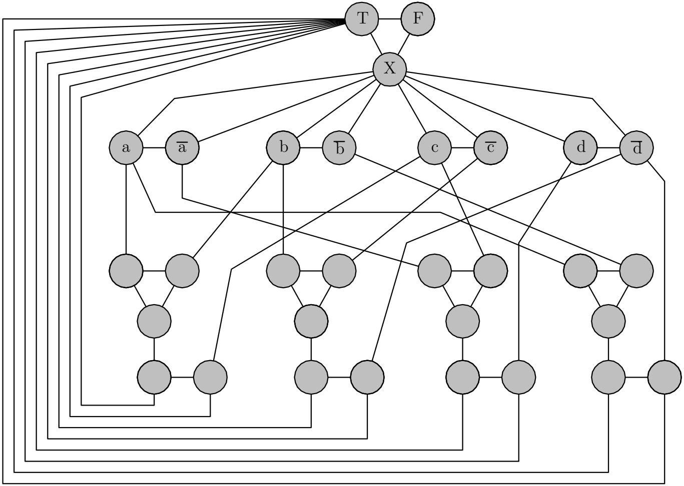
- if x_i is assigned True, color v_i True and $\bar{v}_i$ False
- for each clause $C_j = (a \vee b \vee c)$ at least one of a, b, c is colored True. OR-gadget for C_j can be 3-colored such that output is True.

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

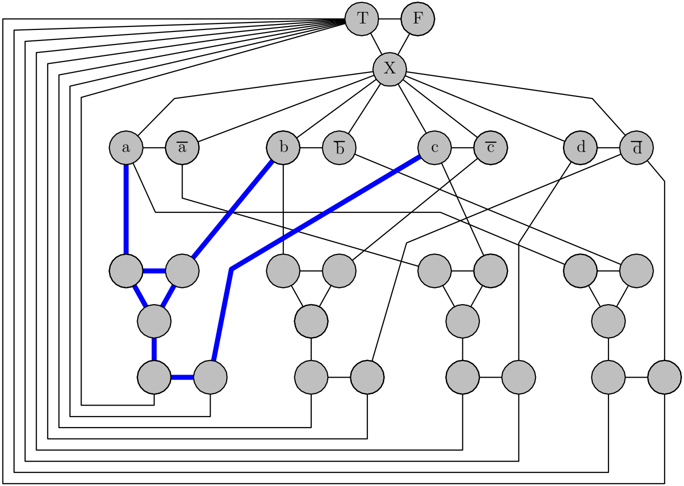
$$(a \lor b \lor c) \land \left(b \lor \overline{c} \lor \overline{d}\right) \land (\overline{a} \lor c \lor d) \land \left(a \lor \overline{b} \lor \overline{d}\right)$$



Graph generated in reduction...

... from 3SAT to 3COLOR

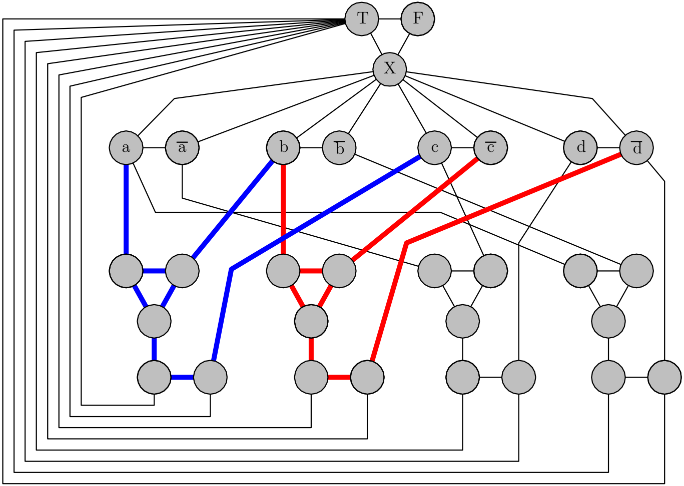
$$(a \vee b \vee c) \wedge \left(b \vee \overline{c} \vee \overline{d}\right) \wedge (\overline{a} \vee c \vee d) \wedge \left(a \vee \overline{b} \vee \overline{d}\right)$$



Graph generated in reduction...

... from 3SAT to 3COLOR

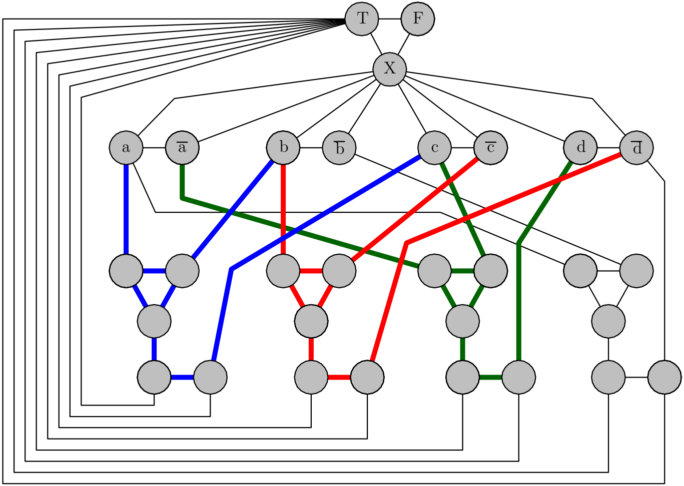
$$(a \lor b \lor c) \land \left(b \lor \overline{c} \lor \overline{d}\right) \land (\overline{a} \lor c \lor d) \land \left(a \lor \overline{b} \lor \overline{d}\right)$$



Graph generated in reduction...

... from 3SAT to 3COLOR

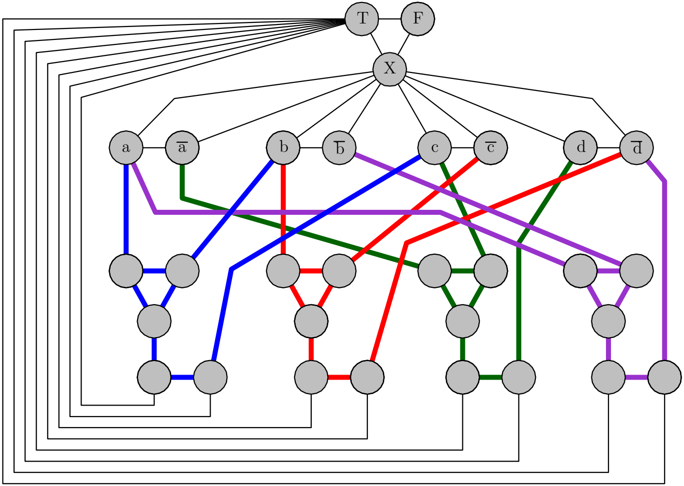
$$(a \vee b \vee c) \wedge \left(b \vee \overline{c} \vee \overline{d}\right) \wedge (\overline{a} \vee c \vee d) \wedge \left(a \vee \overline{b} \vee \overline{d}\right)$$



Graph generated in reduction...

... from 3SAT to 3COLOR

$$(a \lor b \lor c) \land \left(b \lor \overline{c} \lor \overline{d}\right) \land (\overline{a} \lor c \lor d) \land \left(a \lor \overline{b} \lor \overline{d}\right)$$



Graph generated in reduction...

... from 3SAT to 3COLOR

$$(a \vee b \vee c) \wedge \left(b \vee \overline{c} \vee \overline{d}\right) \wedge (\overline{a} \vee c \vee d) \wedge \left(a \vee \overline{b} \vee \overline{d}\right)$$