NP Completeness

Lecture 23

November 19, 2015

Part I

NP-Completeness

P and **NP** and Turing Machines

1. **P**: set of decision problems that have polynomial time algorithms.
2. **NP**: set of decision problems that have polynomial time verification algorithms.

- Many natural problems we would like to solve are in NP.
- Every problem in NP has an exponential time algorithm (try verifying each possible certificate).
- $P \subseteq NP$
- So some problems in NP are in P (example, shortest path problem)

Big Question: Does every problem in NP have an efficient algorithm? Same as asking whether $P = NP$.

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

NP-Complete Problems

Definition

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

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

Recall reduction: $\mathbf{Y} \leq_P \mathbf{X}$ means that an instance of Y can be efficiently modeled as an instance of X.

Solving NP-Complete Problems

Proposition

Suppose X is NP-Complete. Then X can be solved in polynomial time if and only if $\mathsf{P} = \mathsf{NP}$.

Proof.

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

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

$\Leftarrow$ Since $\mathsf{P} = \mathsf{NP}$, and $X \in \mathsf{NP}$, we have a polynomial time algorithm for X. □

NP-Hard Problems

Definition

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

1. (Hardness) For any $Y \in$ **NP**, we have that **Y** $\leq_P$ **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-Complete**ness

If X is **NP-Complete**

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

Consequences of proving **NP-Complete**ness

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 **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 "natural" problems that are **NP-Complete**?

Answer

Yes! Many, many important 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!

NP-Completeness via Reductions

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

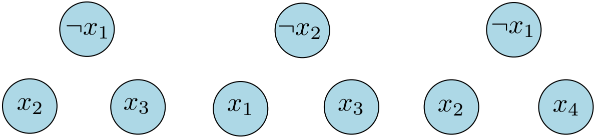


Figure: Graph for
$\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

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

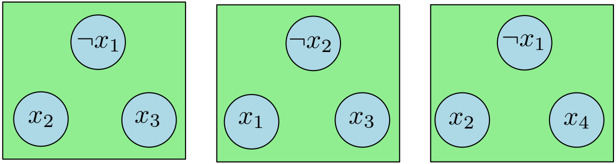


Figure: Graph for
$\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

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

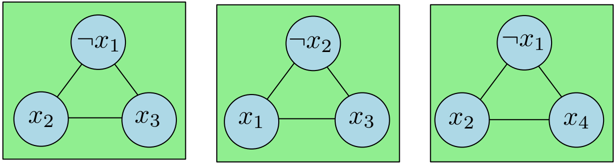


Figure: Graph for
$\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

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

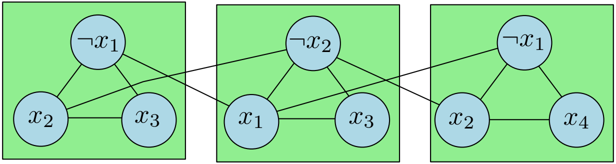


Figure: Graph for
$\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

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
4. Take k to be the number of clauses

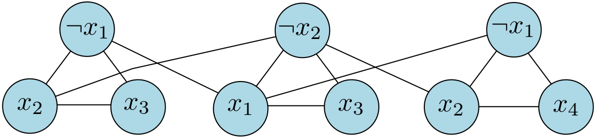


Figure: Graph for
$\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

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 □

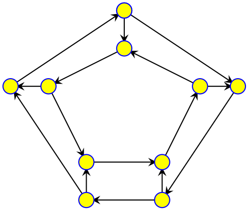
Part III

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?

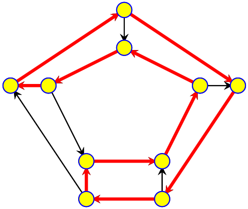


Directed Hamiltonian Cycle

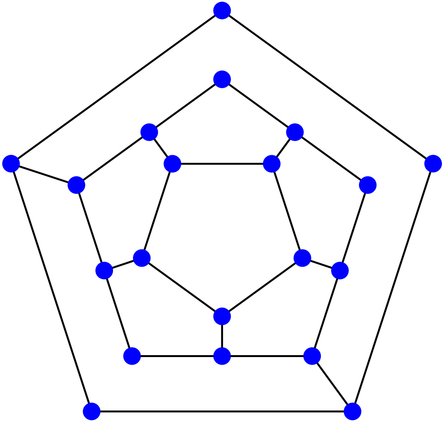
Input Given a directed graph $G = (V, E)$ with n vertices

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 Hamiltonian?



(A) Yes.

(B) No.

Directed Hamiltonian Cycle is **NP-Complete**

- Directed Hamiltonian Cycle is in ***NP***: Why?
- 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 $\mathcal{A}$

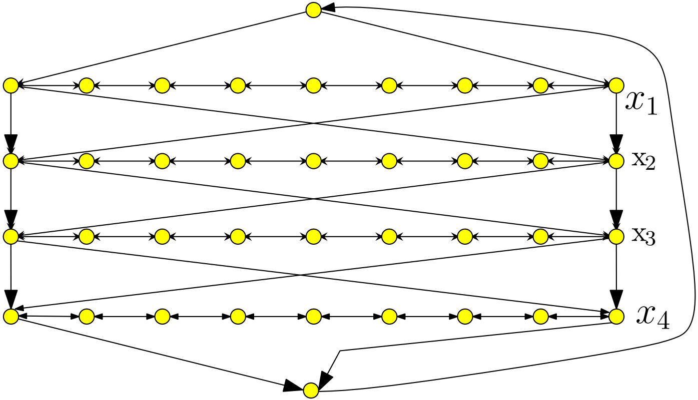
Notation: φ has n variables $x_1, x_2, \ldots, x_n$ and m clauses $C_1, C_2, \ldots, C_m$.

Reduction: First Ideas

- Viewing SAT: Assign values to n variables, and each clause has multiple 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.

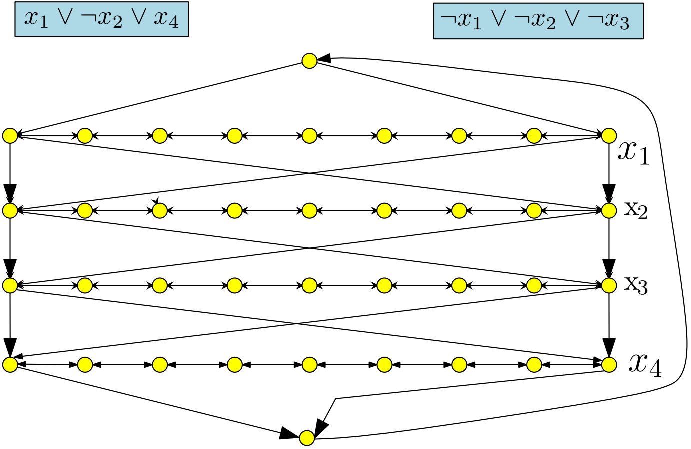
The Reduction: Phase I

- 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 $3m+3$)



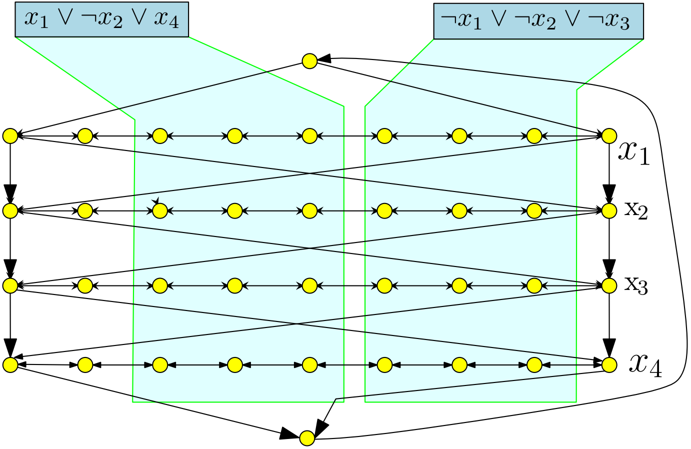
The Reduction: Phase II

- 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.



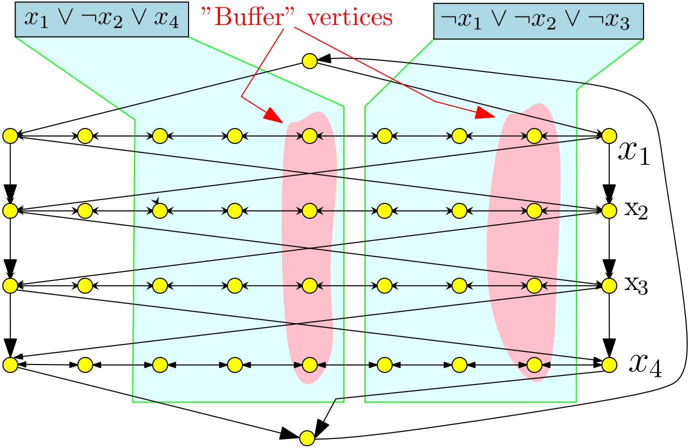
The Reduction: Phase II

- 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.



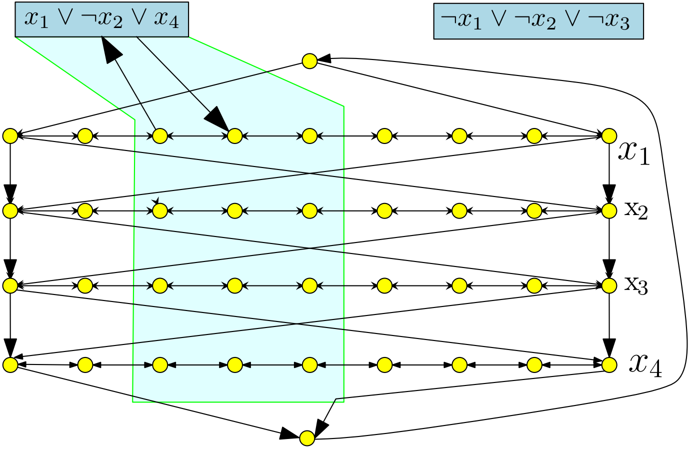
The Reduction: Phase II

- 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.



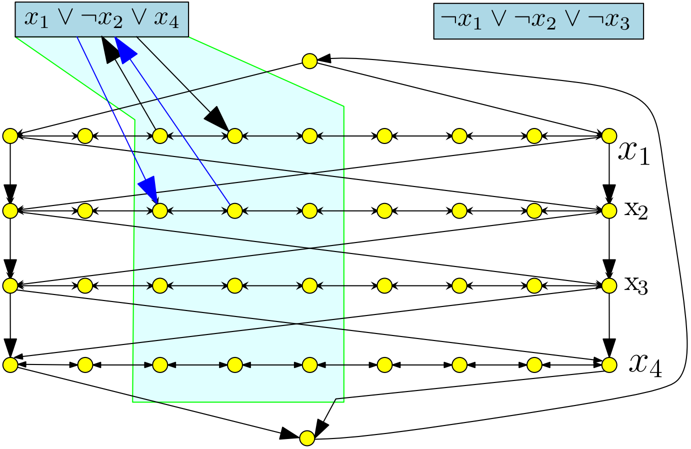
The Reduction: Phase II

- 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.



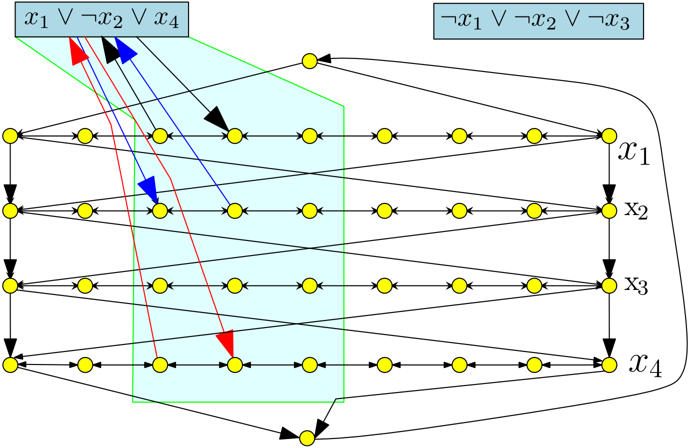
The Reduction: Phase II

- 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.



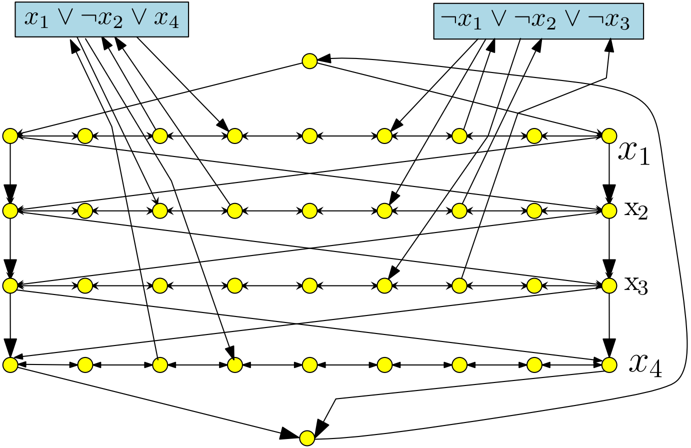
The Reduction: Phase II

- 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

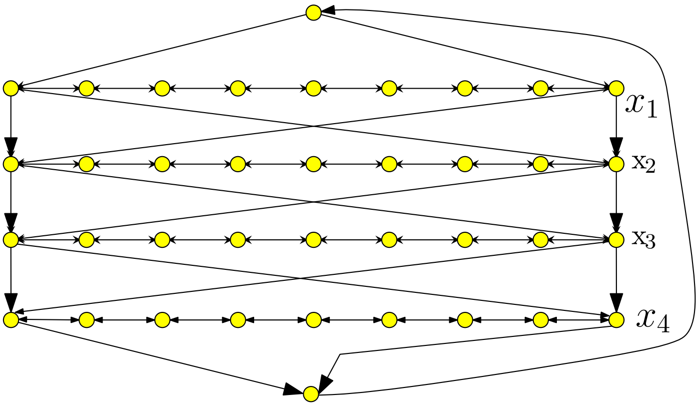
- If $a(x_i) = 1$ then traverse path i from left to right
- If $a(x_i) = 0$ then traverse path i from right to left
- For each clause, path of at least one variable is in the "right" direction to splice in the node corresponding to clause □

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

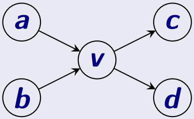
Goal: Given directed graph G, need to construct undirected graph G' such that G has Hamiltonian cycle iff G' has Hamiltonian cycle

Reduction Sketch

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

Reduction

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

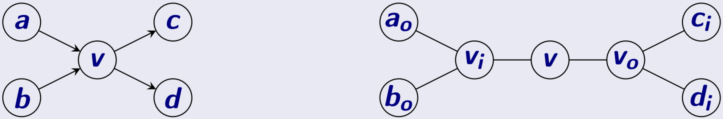


Reduction Sketch

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

Reduction

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

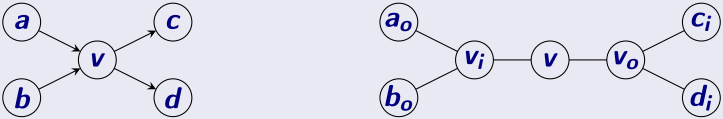


Reduction Sketch

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

Reduction

- Replace each vertex v by 3 vertices: v_{in}, v, and v_{out}
- A directed edge (x, y) is replaced by edge (x_{out}, y_{in})



Reduction: Wrapup

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