Algorithms & Models of Computation

CS/ECE 374, Fall 2020

NP and NP Completeness

Lecture 23

Tuesday, December 1, 2020

LaTeXed: October 27, 2020 13:58

23.1
NP-Completeness: Cook-Levin Theorem

23.1.1
Completeness

NP: Non-deterministic polynomial

Definition 23.1.

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

Definition 23.2.

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

Example 23.3.

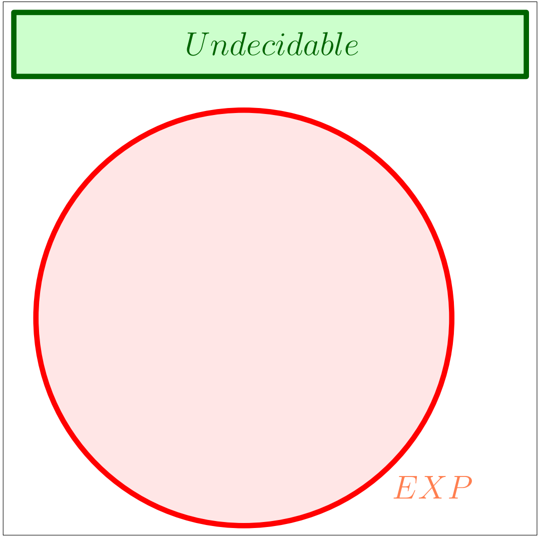
1. **3SAT** is in **NP**.
2. But **Not3SAT** is in **co-NP**.

In the beginning...

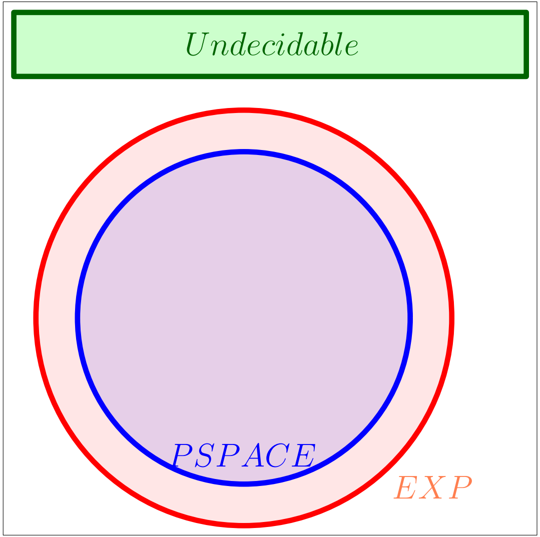
In the beginning...

Undecidable

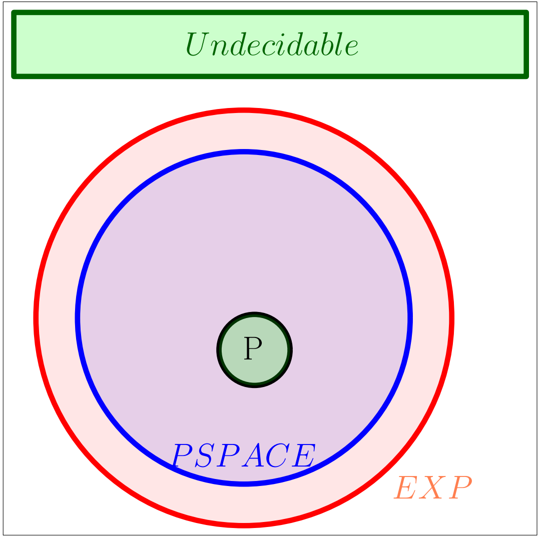
In the beginning...



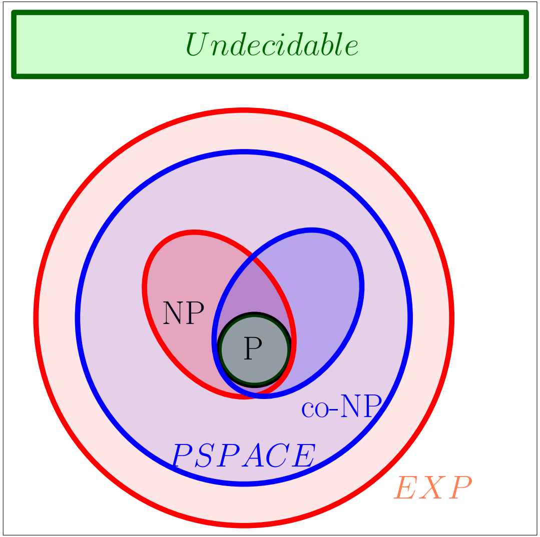
In the beginning...



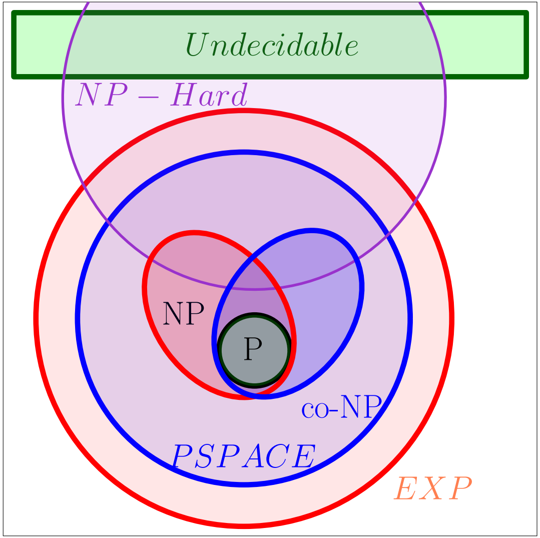
In the beginning...



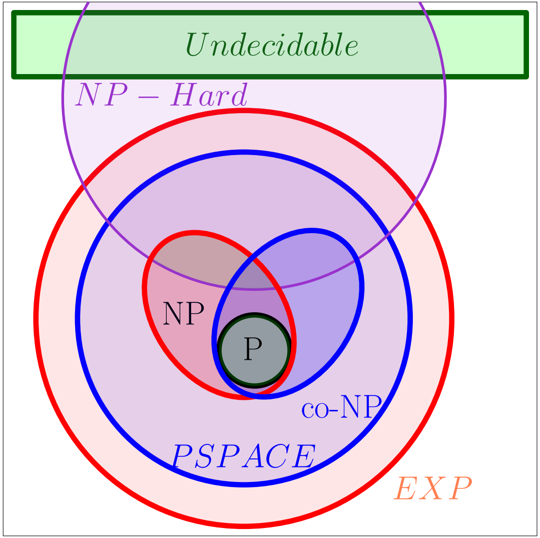
In the beginning...



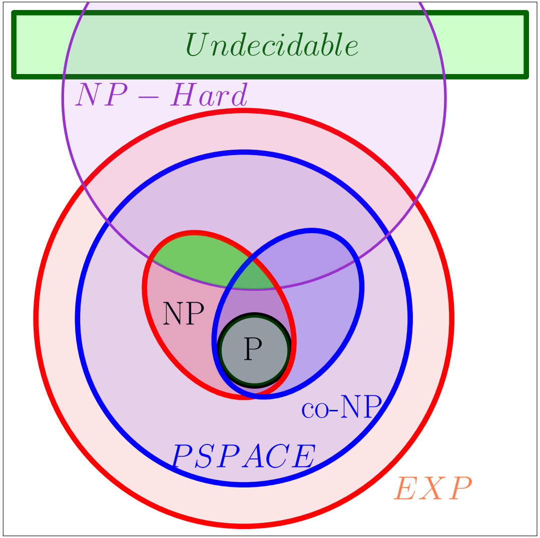
In the beginning...



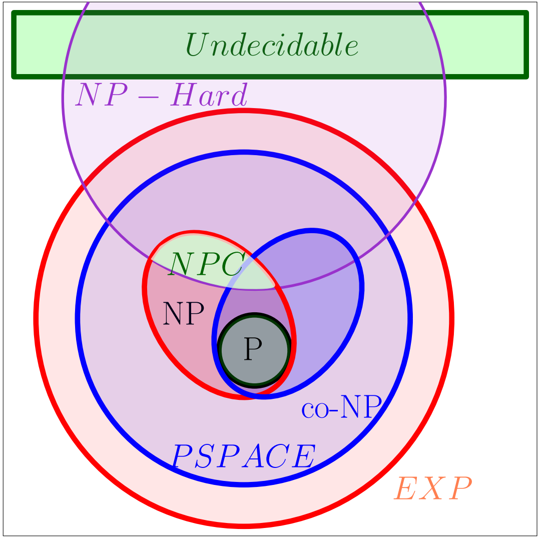
In the beginning...



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 23.4.

A problem $\boldsymbol{X}$ is said to be **NP-Complete** if

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

Solving **NP-Complete** Problems

Proposition 23.5.

Suppose X is **NP-Complete**. *Then X can be solved in polynomial time* $\iff$ **P** $=$ **NP**.

Proof.

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

0.1 Let $Y \in$ **NP**. We know $Y \leq_P X$.

0.2 We showed that if $Y \leq_P X$ and X can be solved in polynomial time, then Y can be solved in polynomial time.

0.3 Thus, every problem $Y \in$ **NP** is such that $Y \in P$.

0.4 $\implies NP \subseteq P$.

0.5 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 23.6.

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 $\mathbf{P} \neq \mathbf{NP}$,
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.)

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.
(This is proof by mob opinion — take with a grain of salt.)

Consequences of proving **NP-Complete**ness

If $\boldsymbol{X}$ is **NP-Complete**

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

$\boldsymbol{X}$ is unlikely to be efficiently solvable.

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

Consequences of proving **NP-Complete**ness

If $\boldsymbol{X}$ is **NP-Complete**

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

$\boldsymbol{X}$ is unlikely to be efficiently solvable.

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

THE END

…

(for now)

23.1.2
SAT is NP-Complete

NP-Complete Problems

Question

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

Answer

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

Cook-Levin Theorem

Theorem 23.7 (Cook-Levin).

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

Need to show

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

Might see proof later...

Steve Cook won the Turing award for his theorem.

Cook-Levin Theorem

Theorem 23.7 (Cook-Levin).

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

Need to show

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

Might see proof later...

Steve Cook won the Turing award for his theorem.

THE END

…

(for now)

23.1.3
Other NP Complete Problems

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 $Y \leq_P X$.

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 $Y \leq_P X$.

Proving that a problem **X** is **NP-Complete**

To prove $\boldsymbol{X}$ is **NP-Complete**, show

1. Show that $\boldsymbol{X}$ is in **NP**.
2. Give a polynomial-time reduction <u>from</u> a known **NP-Complete** problem such as **SAT** <u>to</u> $\boldsymbol{X}$

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

$\boldsymbol{Y} \leq_P \boldsymbol{SAT}$ and $\boldsymbol{SAT} \leq_P \boldsymbol{X}$ and hence $\boldsymbol{Y} \leq_P \boldsymbol{X}$.

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**

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

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!

THE END

…

(for now)

23.2
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?

Lemma 23.1.

Independent set is in **NP**.

Independent Set

Problem: Independent Set

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

Lemma 23.1.

Independent set is in **NP**.

3SAT $\leq_P$ Independent Set

The reduction 3SAT $\leq_P$ Independent Set

Input: Given a 3CNF 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 3CNF formulas – reduction would not work for other kinds of boolean formulas.

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.

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**

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.

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.

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.

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
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

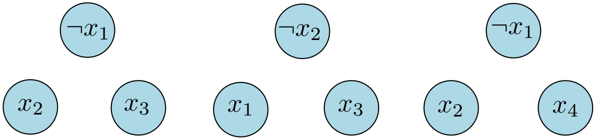


Figure: Graph for $\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

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

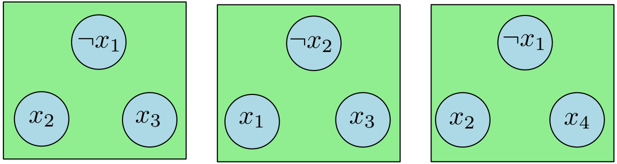


Figure: Graph for $\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

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

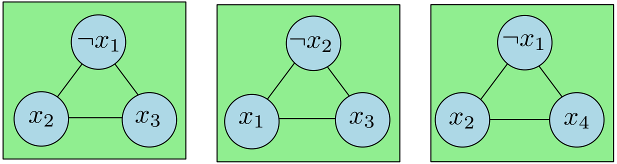


Figure: Graph for $\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

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

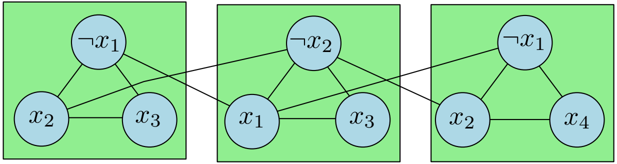


Figure: Graph for $\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

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

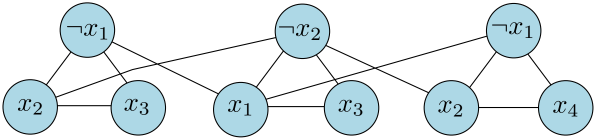


Figure: Graph for $\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

Correctness

Proposition 23.2.

φ is satisfiable iff $\mathbf{G}_\varphi$ has an independent set of size $\mathbf{k}$ ($=$ number of clauses in φ).

Proof.

$\Rightarrow$ Let $\mathbf{a}$ be the truth assignment satisfying φ

- Pick one of the vertices, corresponding to true literals under $\mathbf{a}$, from each triangle. This is an independent set of the appropriate size. Why?

□

Correctness

Proposition 23.2.

φ is satisfiable iff $\boldsymbol{G}_\varphi$ has an independent set of size $\boldsymbol{k}$ ($=$ number of clauses in φ).

Proof.

$\Rightarrow$ Let $\boldsymbol{a}$ be the truth assignment satisfying φ

- Pick one of the vertices, corresponding to true literals under $\boldsymbol{a}$, from each triangle. This is an independent set of the appropriate size. Why?

□

Correctness

Proposition 23.2.

φ is satisfiable iff $\mathbf{G}_\varphi$ has an independent set of size $\mathbf{k}$ ($=$ number of clauses in φ).

Proof.

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

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

Summary

Theorem 23.3.

Independent set *is* **NP-Complete** *(i.e.,* **NPC***).*

THE END

...

(for now)

23.3
NP-Completeness of Hamiltonian Cycle

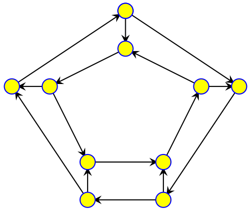
23.3.1
Reduction from 3SAT to Hamiltonian Cycle: Basic idea

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

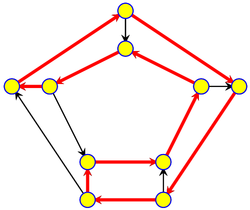


Directed Hamiltonian Cycle

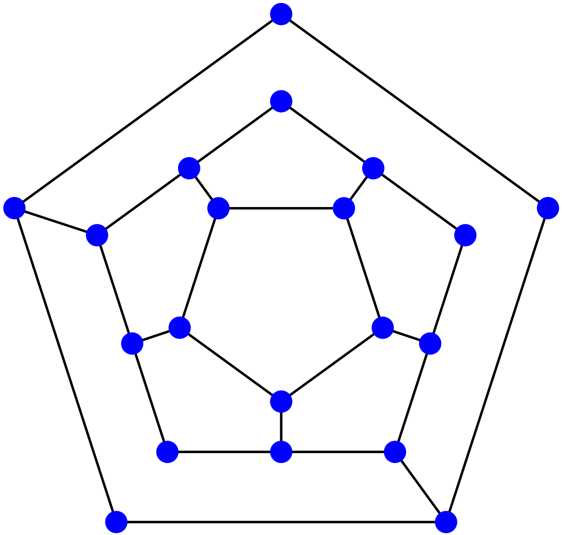
Input Given a directed graph $G = (V, E)$ with n vertices

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

Reduction construction

From 3SAT to Hamiltonian cycle in directed graph

1. To show reduction, we next describe an algorithm:
   - Input: **3SAT** formula φ
   - Output: A graph G_φ.
   - Running time is polynomial.
   - Requirement: φ is satisfiable $\iff$ G_φ is Hamiltonian.
2. Given **3SAT** 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 $\mathcal{A}$
3. Notation: φ has n variables $x_1, x_2, \ldots, x_n$ and m clauses $C_1, C_2, \ldots, C_m$.

Reduction construction

From 3SAT to Hamiltonian cycle in directed graph

1. To show reduction, we next describe an algorithm:
   - Input: **3SAT** formula φ
   - Output: A graph G_φ.
   - Running time is polynomial.
   - Requirement: φ is satisfiable $\iff$ G_φ is Hamiltonian.
2. Given **3SAT** 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 $\mathcal{A}$
3. Notation: φ has n variables $x_1, x_2, \ldots, x_n$ and m clauses $C_1, C_2, \ldots, C_m$.

Reduction construction

From 3SAT to Hamiltonian cycle in directed graph

1. To show reduction, we next describe an algorithm:
   - Input: **3SAT** formula φ
   - Output: A graph G_φ.
   - Running time is polynomial.
   - Requirement: φ is satisfiable $\iff$ G_φ is Hamiltonian.
2. Given **3SAT** 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 $\mathcal{A}$
3. Notation: φ has n variables $x_1, x_2, \ldots, x_n$ and m clauses $C_1, C_2, \ldots, C_m$.

Reduction construction

From 3SAT to Hamiltonian cycle in directed graph

1. To show reduction, we next describe an algorithm:
   - Input: **3SAT** formula φ
   - Output: A graph G_φ.
   - Running time is polynomial.
   - Requirement: φ is satisfiable $\iff$ G_φ is Hamiltonian.
2. Given **3SAT** 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 $\mathcal{A}$
3. Notation: φ has n variables $x_1, x_2, \ldots, x_n$ and m clauses $C_1, C_2, \ldots, C_m$.

Reduction construction

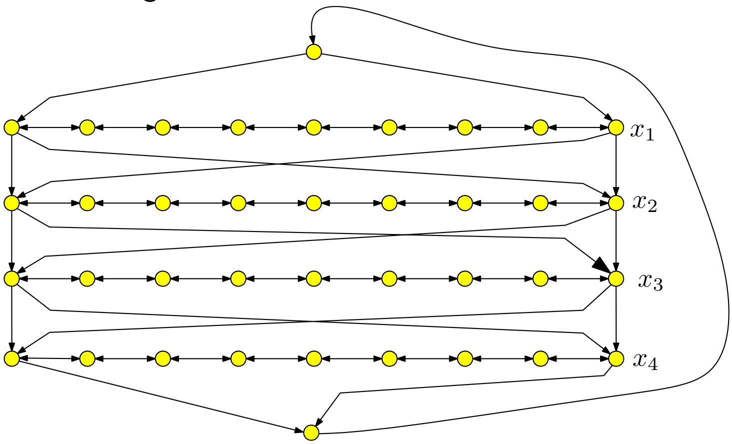
From 3SAT to Hamiltonian cycle in directed graph

1. To show reduction, we next describe an algorithm:
   - Input: **3SAT** formula φ
   - Output: A graph G_φ.
   - Running time is polynomial.
   - Requirement: φ is satisfiable $\iff$ G_φ is Hamiltonian.
2. Given **3SAT** 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 $\mathcal{A}$
3. Notation: φ has n variables $x_1, x_2, \ldots, x_n$ and m clauses $C_1, C_2, \ldots, C_m$.

Encoding assignments

Converting φ to a graph

Given a formula with n variables, we need a graph with $\mathbf{2^n}$ different Hamiltonian paths, that can encode their assignments.



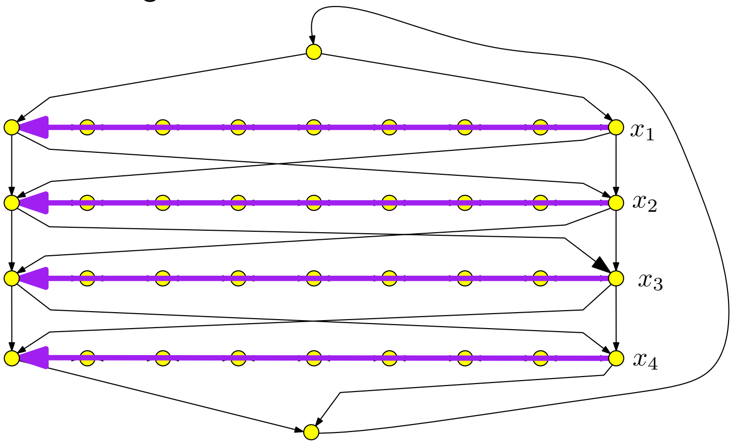
1

5

Encoding assignments

Converting φ to a graph

Given a formula with n variables, we need a graph with 2^n different Hamiltonian paths, that can encode their assignments.



2

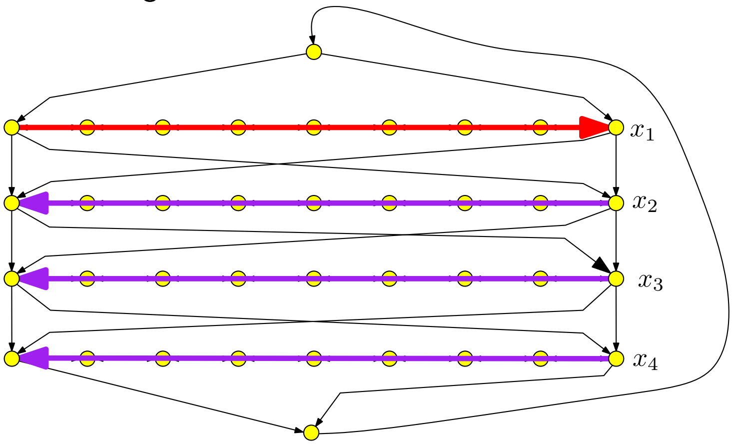
5

$$x_1 = 0, x_2 = 0, x_3 = 0, x_4 = 0$$

Encoding assignments

Converting φ to a graph

Given a formula with n variables, we need a graph with 2^n different Hamiltonian paths, that can encode their assignments.



3

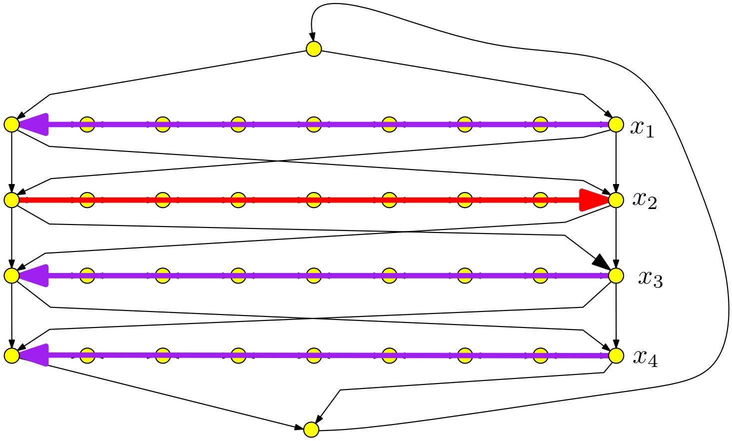
5

$$x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 0$$

Encoding assignments

Converting φ to a graph

Given a formula with $\boldsymbol{n}$ variables, we need a graph with $\mathbf{2^n}$ different Hamiltonian paths, that can encode their assignments.



4

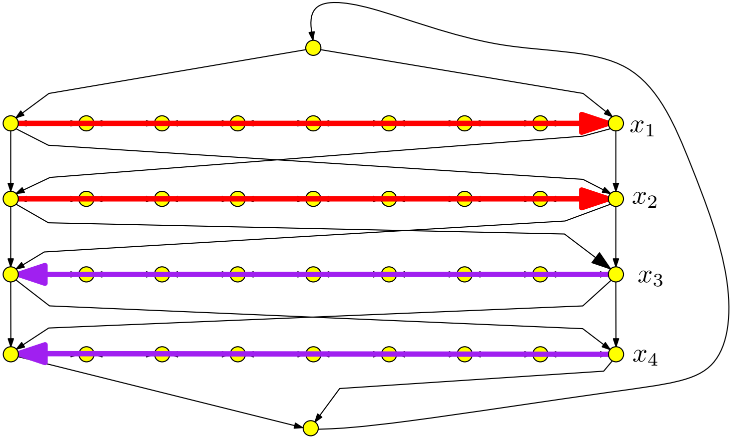
5

$$x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 0$$

Encoding assignments

Converting φ to a graph

Given a formula with n variables, we need a graph with 2^n different Hamiltonian paths, that can encode their assignments.



5

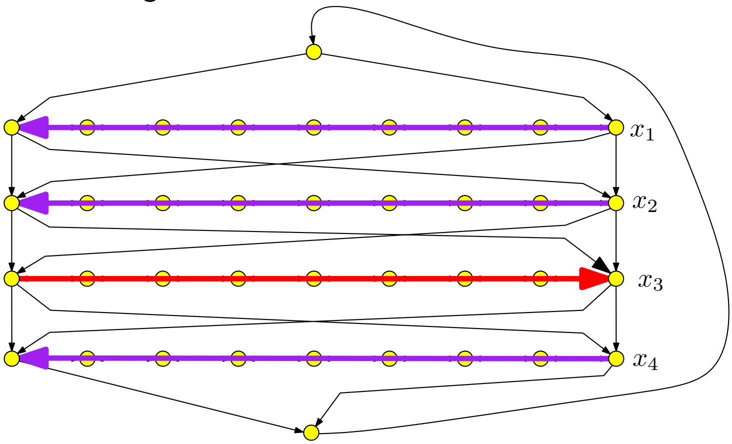
5

$$x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0$$

Encoding assignments

Converting φ to a graph

Given a formula with n variables, we need a graph with 2^n different Hamiltonian paths, that can encode their assignments.



6

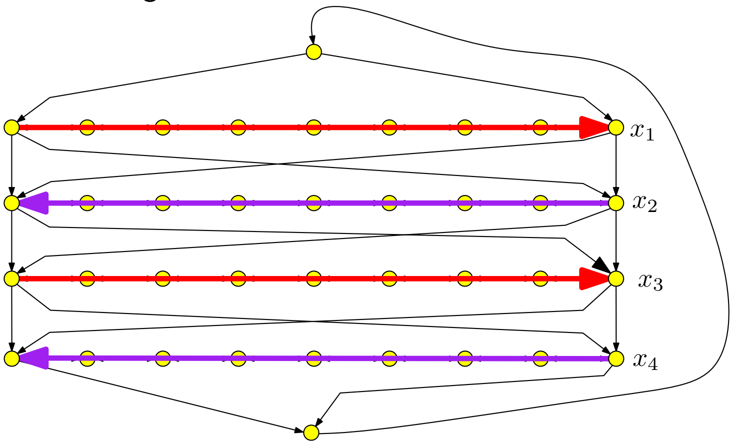
5

$$x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 0$$

Encoding assignments

Converting φ to a graph

Given a formula with n variables, we need a graph with 2^n different Hamiltonian paths, that can encode their assignments.



7

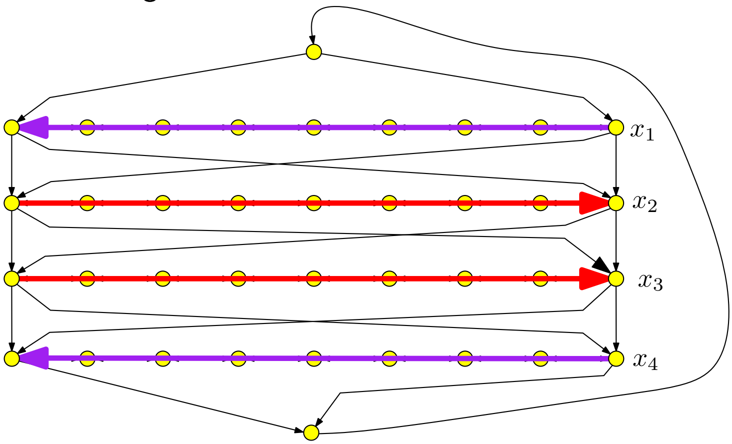
5

$$x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 0$$

Encoding assignments

Converting φ to a graph

Given a formula with n variables, we need a graph with 2^n different Hamiltonian paths, that can encode their assignments.



8

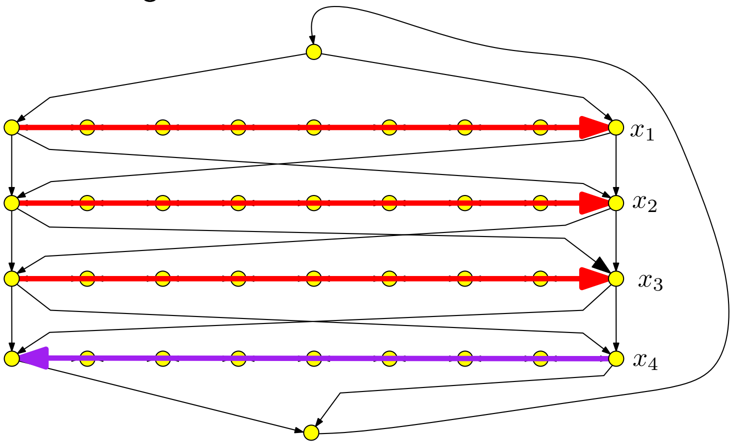
5

$$x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0$$

Encoding assignments

Converting φ to a graph

Given a formula with n variables, we need a graph with 2^n different Hamiltonian paths, that can encode their assignments.



9

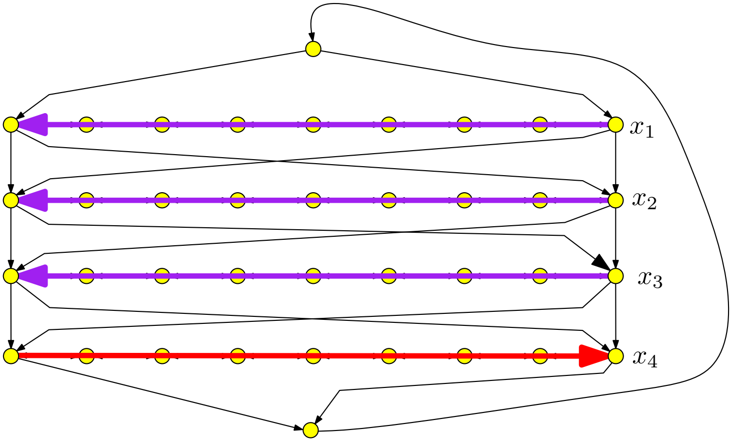
5

$$x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 01$$

Encoding assignments

Converting φ to a graph

Given a formula with n variables, we need a graph with 2^n different Hamiltonian paths, that can encode their assignments.



10

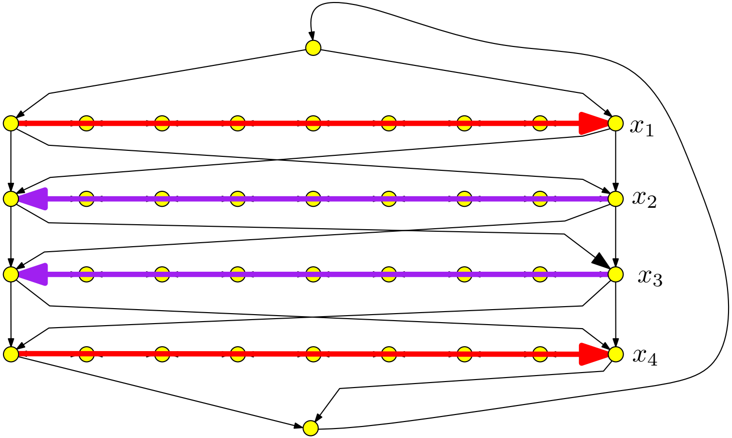
5

$$x_1 = 0, x_2 = 0, x_3 = 0, x_4 = 1$$

Encoding assignments

Converting φ to a graph

Given a formula with n variables, we need a graph with 2^n different Hamiltonian paths, that can encode their assignments.



11

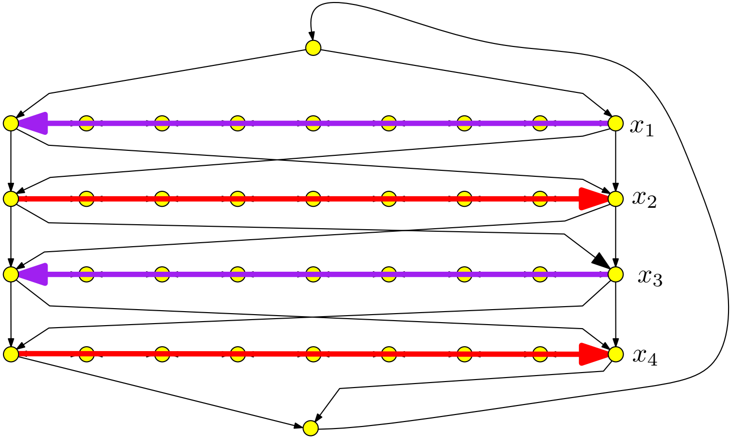
5

$$x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1$$

Encoding assignments

Converting φ to a graph

Given a formula with n variables, we need a graph with 2^n different Hamiltonian paths, that can encode their assignments.



12

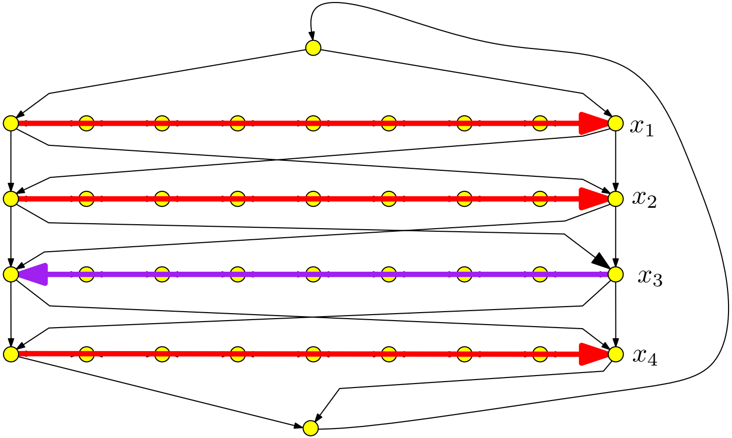
5

$x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 1$

Encoding assignments

Converting φ to a graph

Given a formula with n variables, we need a graph with 2^n different Hamiltonian paths, that can encode their assignments.



13

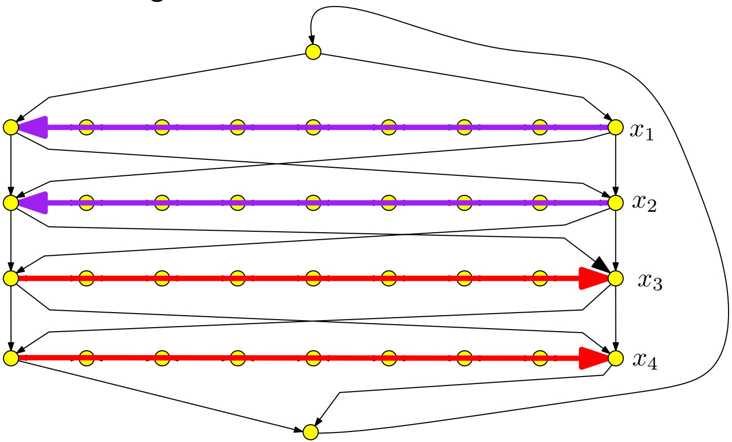
5

$$x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 1$$

Encoding assignments

Converting φ to a graph

Given a formula with n variables, we need a graph with 2^n different Hamiltonian paths, that can encode their assignments.



14

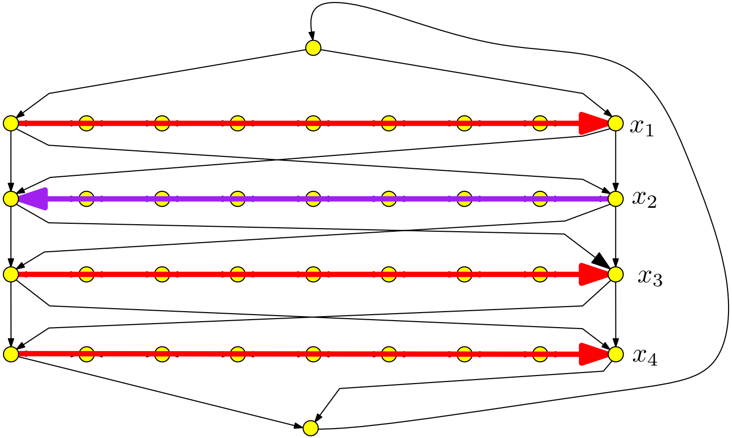
5

$$x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 1$$

Encoding assignments

Converting φ to a graph

Given a formula with n variables, we need a graph with 2^n different Hamiltonian paths, that can encode their assignments.



15

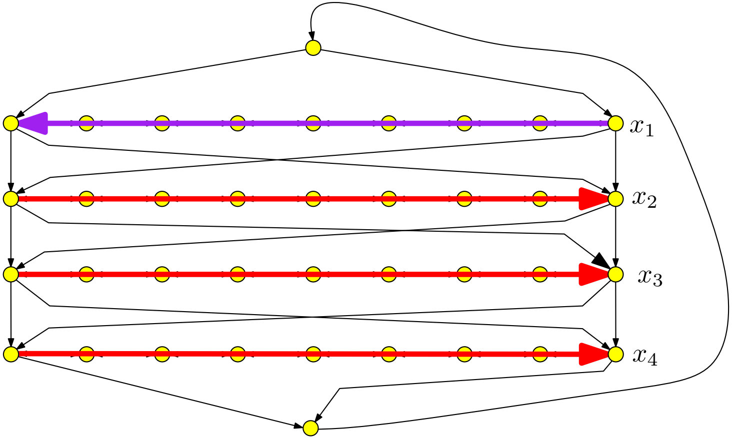
5

$$x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 1$$

Encoding assignments

Converting φ to a graph

Given a formula with n variables, we need a graph with 2^n different Hamiltonian paths, that can encode their assignments.



16

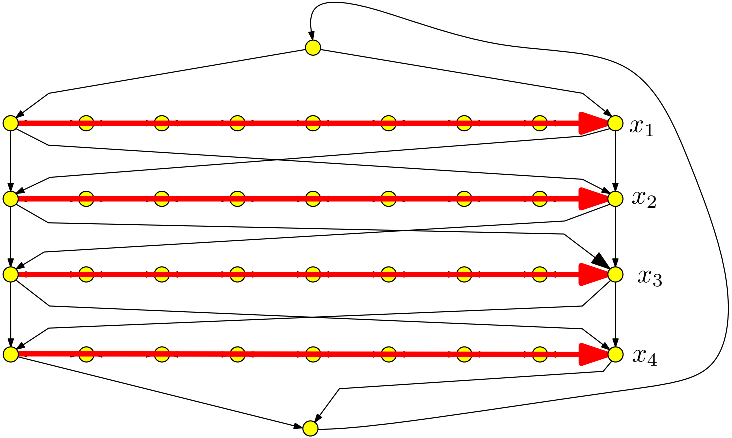
5

$$x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 1$$

Encoding assignments

Converting φ to a graph

Given a formula with n variables, we need a graph with 2^n different Hamiltonian paths, that can encode their assignments.



17

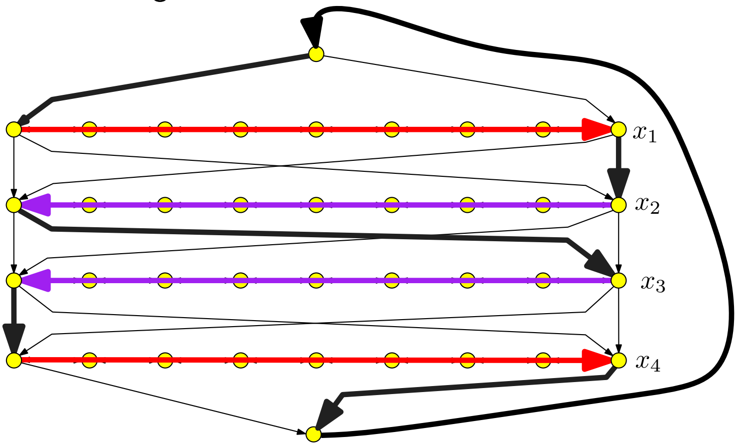
5

$$x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 1$$

Encoding assignments

Converting φ to a graph

Given a formula with n variables, we need a graph with 2^n different Hamiltonian paths, that can encode their assignments.



18

5

THE END

...

(for now)

23.3.2
The reduction: Encoding the formula constraints

3SAT $\leq_P$ Directed Hamiltonian Cycle

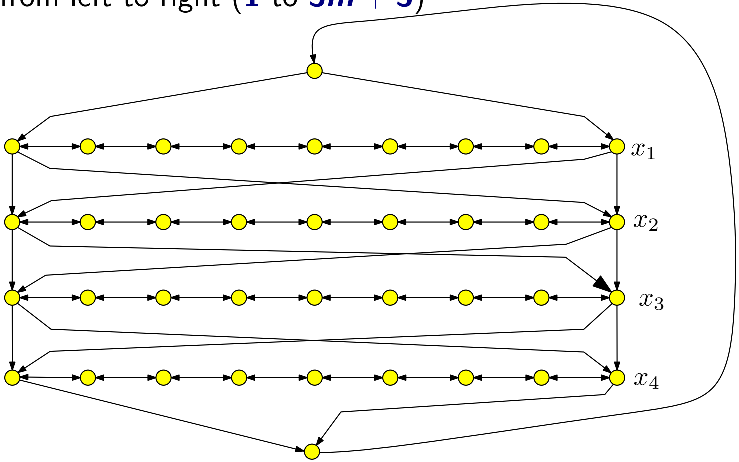
Input: φ formula.
Output: Graph G_φ.

Saw: How to encode assignments...
Now need to encode constraints of φ.

The reduction algorithm: Phase I

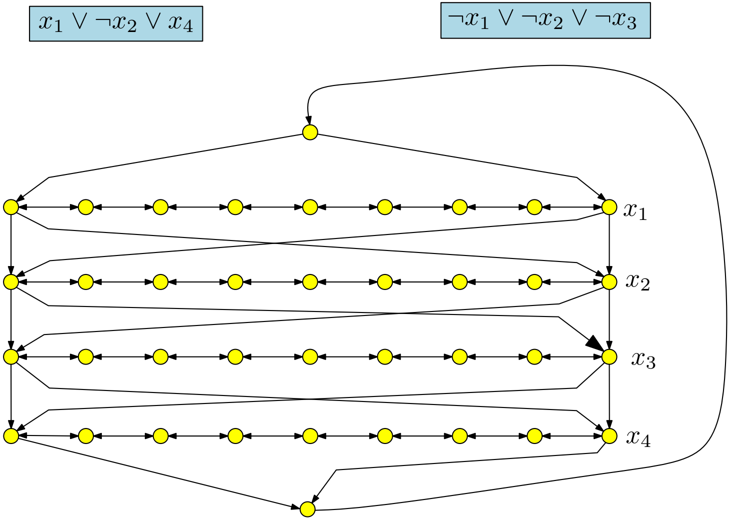
Converting φ to a graph

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



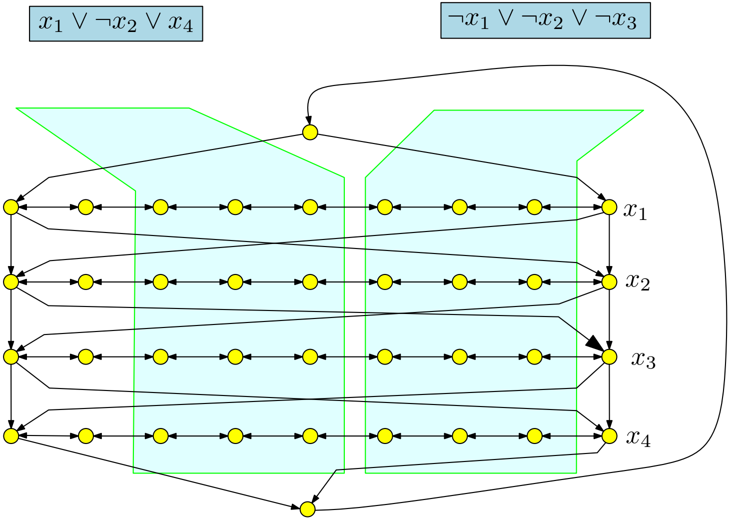
The Reduction algorithm: Phase II

- 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 algorithm: 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.

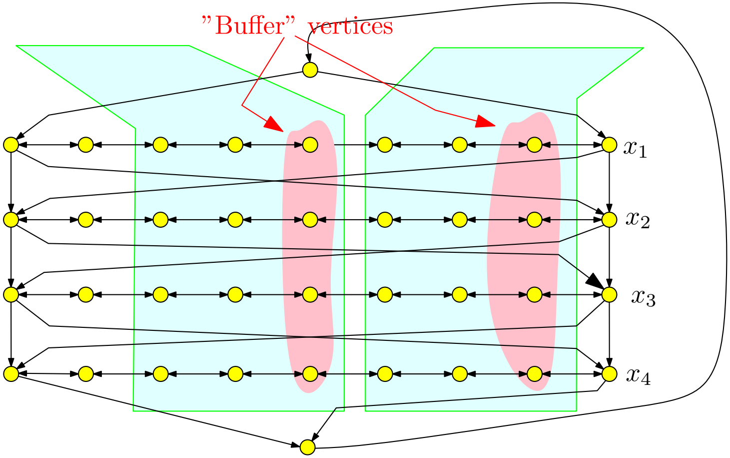


The Reduction algorithm: Phase II

- 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.

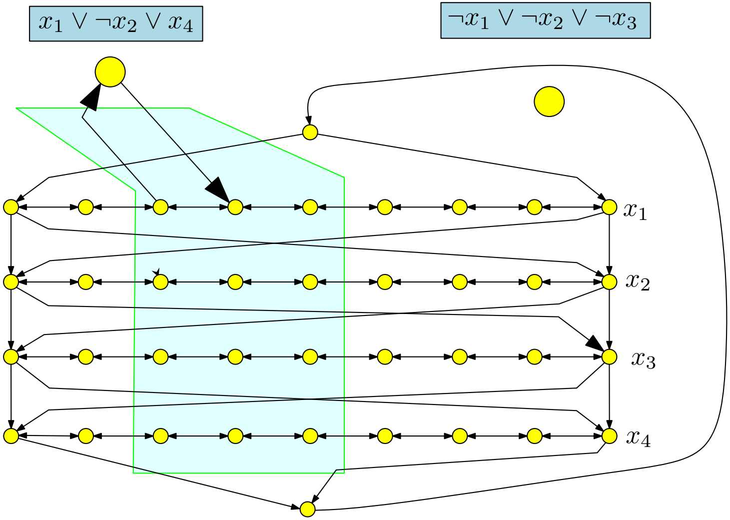
$x_1 \vee \neg x_2 \vee x_4$

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



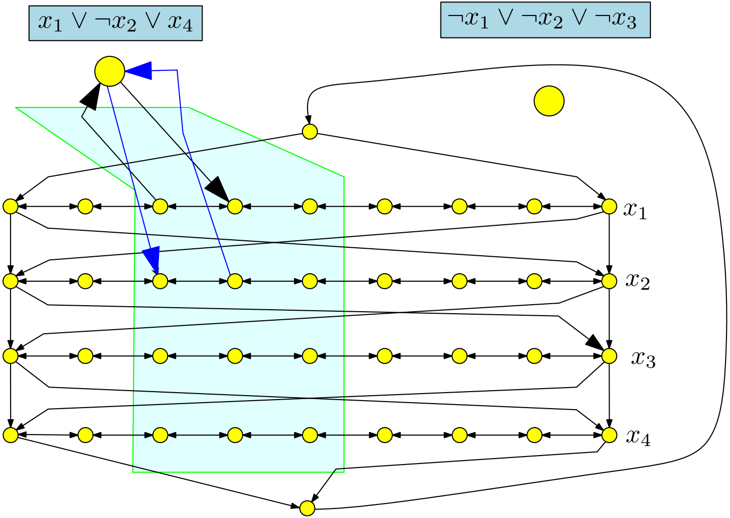
The Reduction algorithm: Phase II

- 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.



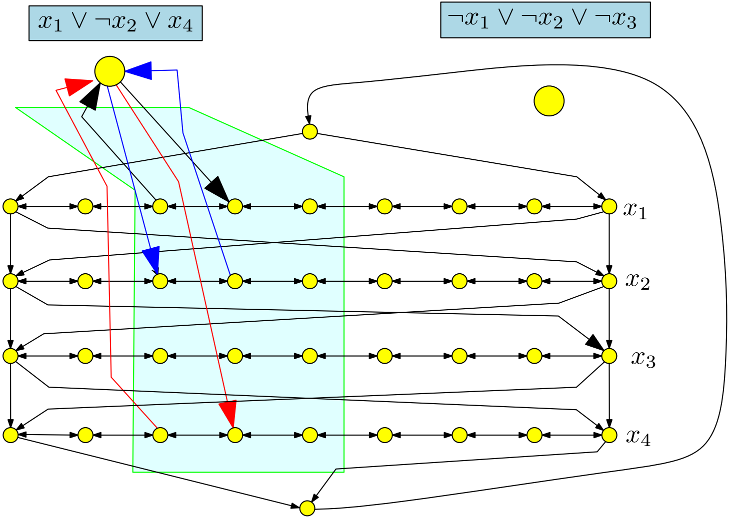
The Reduction algorithm: Phase II

- 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.



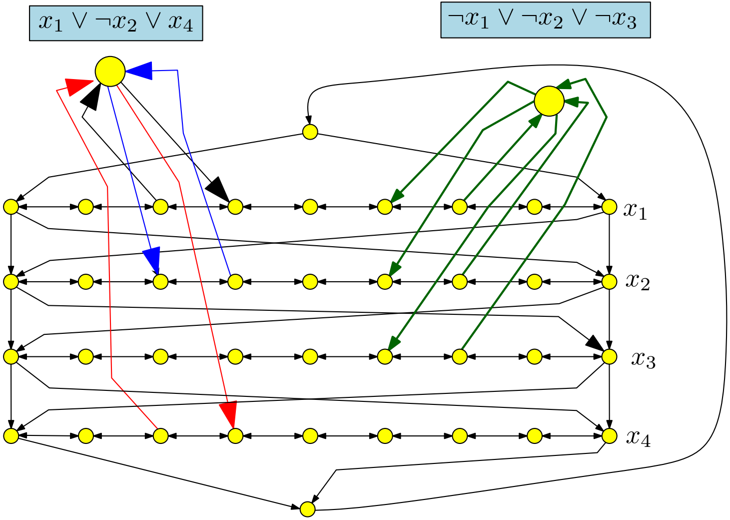
The Reduction algorithm: Phase II

- 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 algorithm: 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 END

...

(for now)

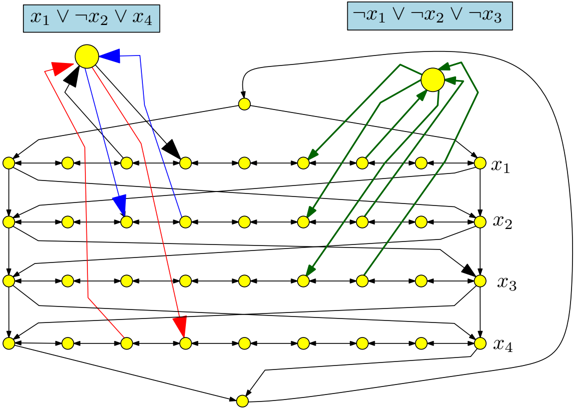
23.3.3

If there is a satisfying assignment, then there is a Hamiltonian cycle

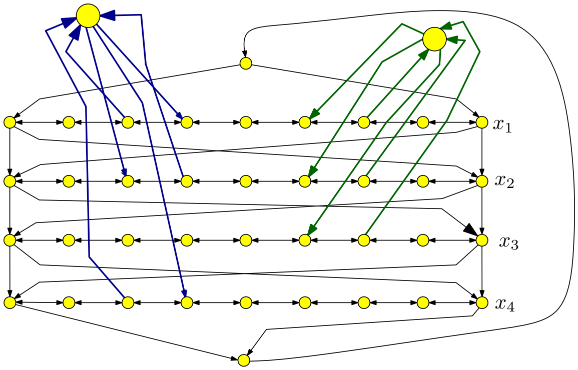
From satisfying assignment to Hamiltonian cycle: By figure

3SAT formula φ:

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



From satisfying assignment to Hamiltonian cycle: By figure



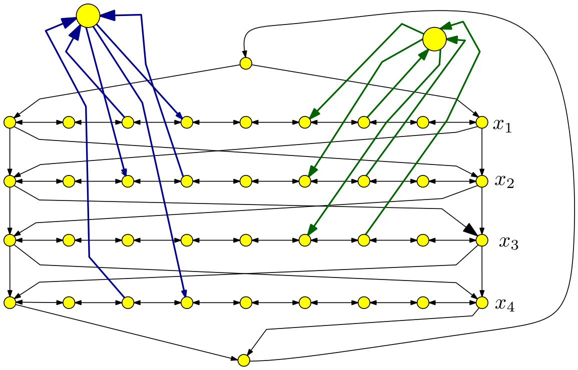
3SAT formula φ:

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

From satisfying assignment to Hamiltonian cycle: By figure

3SAT formula φ:

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



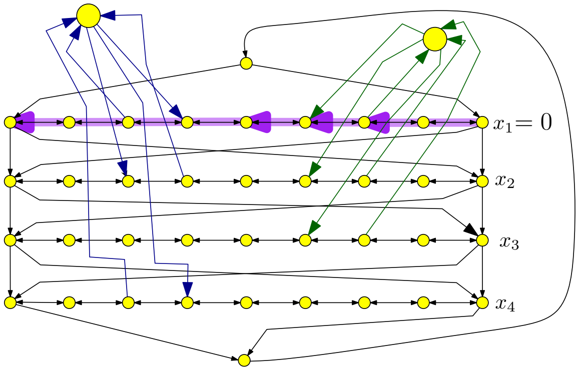
A satisfying assignment:
$x_1 = \mathbf{0}$, $x_2 = \mathbf{1}$, $x_3 = \mathbf{0}$, $x_4 = \mathbf{1}$

From satisfying assignment to Hamiltonian cycle: By figure

3SAT formula φ:

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

A satisfying assignment:
$x_1 = 0$, $x_2 = 1$, $x_3 = 0$, $x_4 = 1$

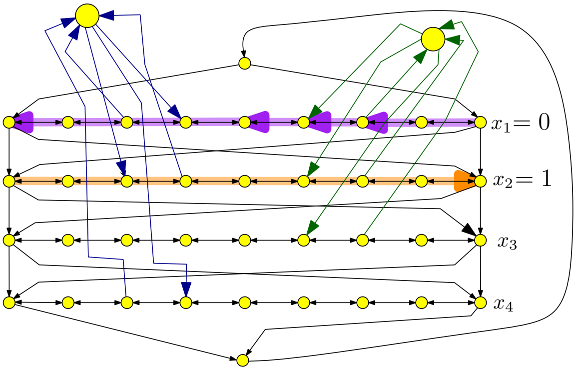


From satisfying assignment to Hamiltonian cycle: By figure

3SAT formula φ:

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

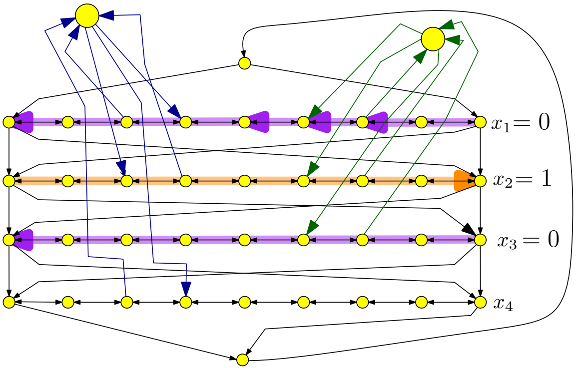
A satisfying assignment:
$x_1 = \mathbf{0}$, $x_2 = \mathbf{1}$, $x_3 = \mathbf{0}$, $x_4 = \mathbf{1}$



From satisfying assignment to Hamiltonian cycle: By figure

3SAT formula φ:

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



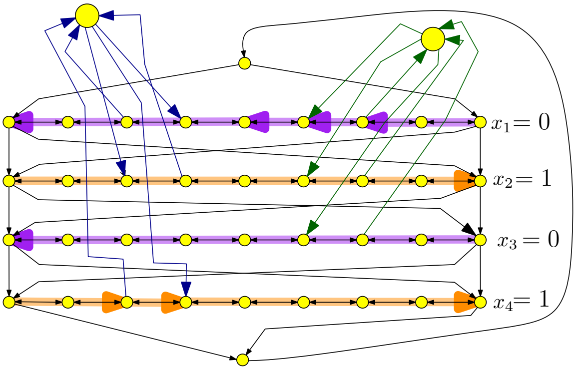
A satisfying assignment:
$x_1 = 0$, $x_2 = 1$, $x_3 = 0$, $x_4 = 1$

From satisfying assignment to Hamiltonian cycle: By figure

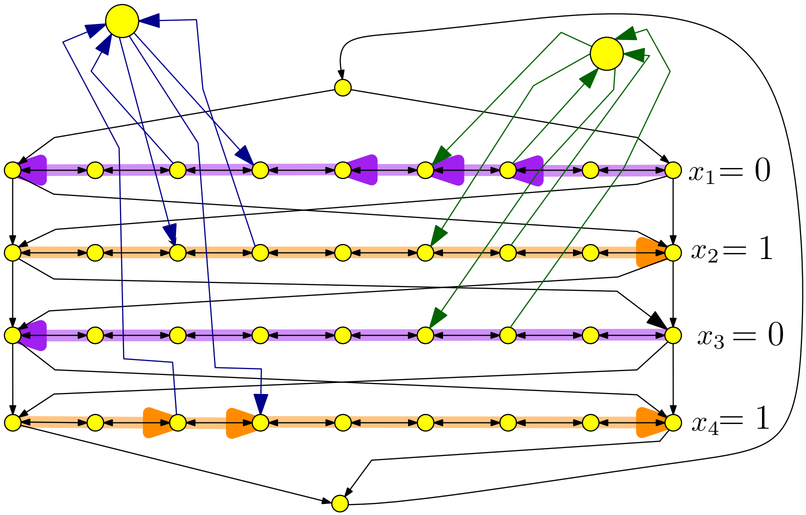
3SAT formula φ:

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

A satisfying assignment:
$x_1 = 0$, $x_2 = 1$, $x_3 = 0$, $x_4 = 1$

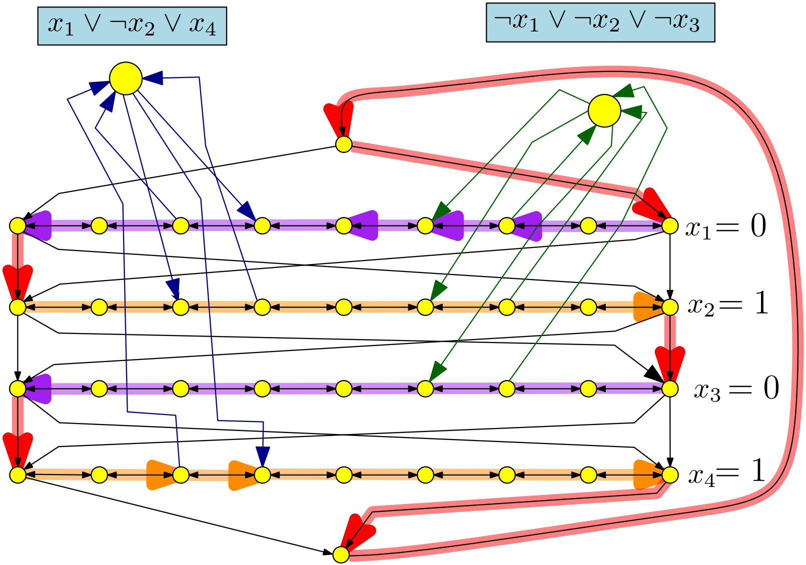


Reduction: Satisfying assignment ⇒ Hamiltonian cycle



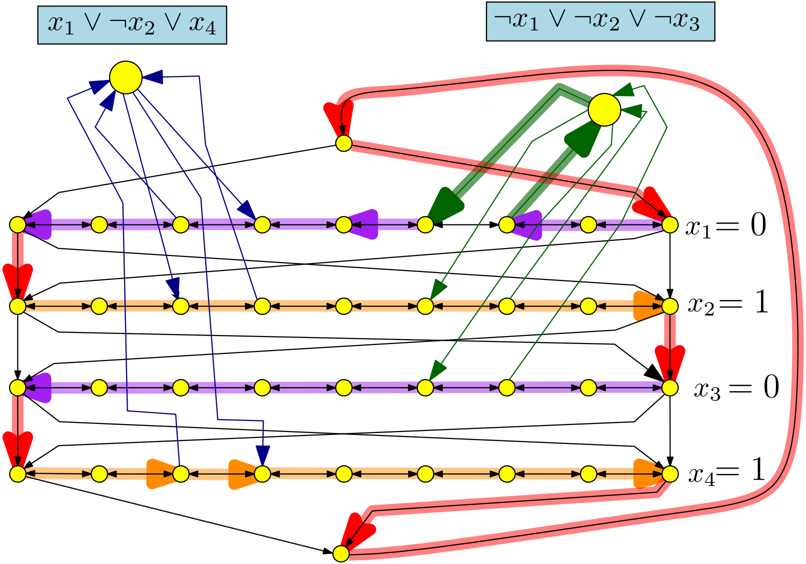
Satisfying assignment: $\boldsymbol{x_1 = 0}$, $\boldsymbol{x_2 = 1}$, $\boldsymbol{x_3 = 0}$, $\boldsymbol{x_4 = 1}$

Reduction: Satisfying assignment ⇒ Hamiltonian cycle



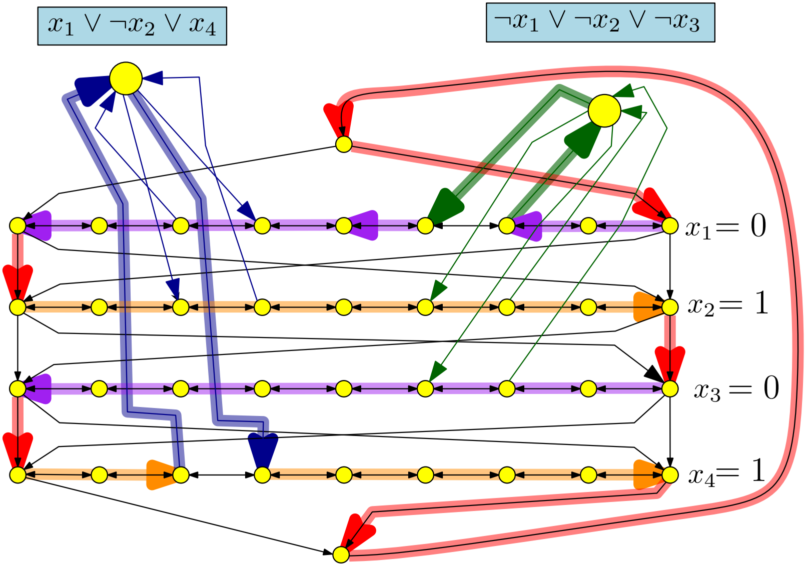
Satisfying assignment: $\boldsymbol{x_1 = 0}$, $\boldsymbol{x_2 = 1}$, $\boldsymbol{x_3 = 0}$, $\boldsymbol{x_4 = 1}$

Reduction: Satisfying assignment ⇒ Hamiltonian cycle



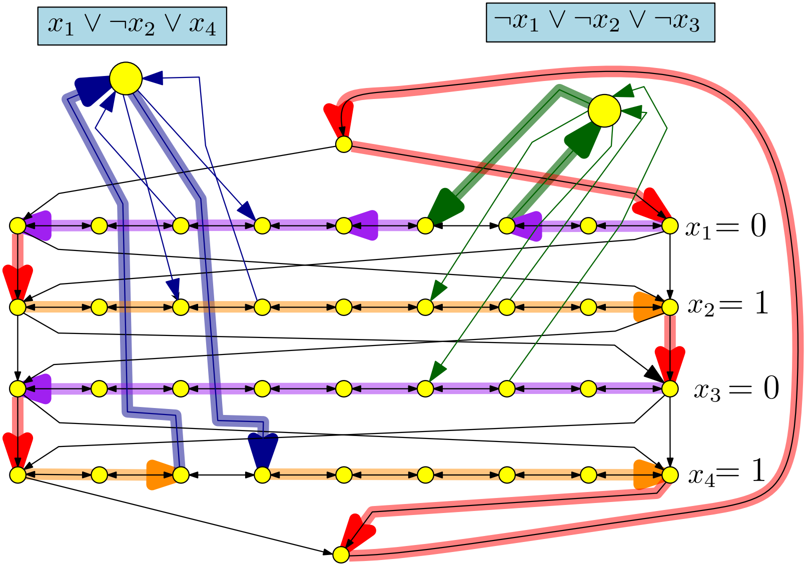
Satisfying assignment: $\boldsymbol{x_1 = 0}$, $\boldsymbol{x_2 = 1}$, $\boldsymbol{x_3 = 0}$, $\boldsymbol{x_4 = 1}$

Reduction: Satisfying assignment ⇒ Hamiltonian cycle



Satisfying assignment: $\boldsymbol{x_1 = 0}$, $\boldsymbol{x_2 = 1}$, $\boldsymbol{x_3 = 0}$, $\boldsymbol{x_4 = 1}$

Reduction: Satisfying assignment ⇒ Hamiltonian cycle



Satisfying assignment: $\boldsymbol{x_1 = 0}$, $\boldsymbol{x_2 = 1}$, $\boldsymbol{x_3 = 0}$, $\boldsymbol{x_4 = 1}$

Conclude: If φ has a satisfying assignment then there is an Hamiltonian cycle in G_φ.

Correctness Proof

Lemma 23.1.

φ has a satisfying assignment $\alpha \implies G_\varphi$ has a Hamiltonian cycle.

Proof.

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

- If $\alpha(x_i) = 1$ then traverse path i from left to right
- If $\alpha(x_i) = 0$ 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
- Clearly, resulting cycle is Hamiltonian. □

THE END

...

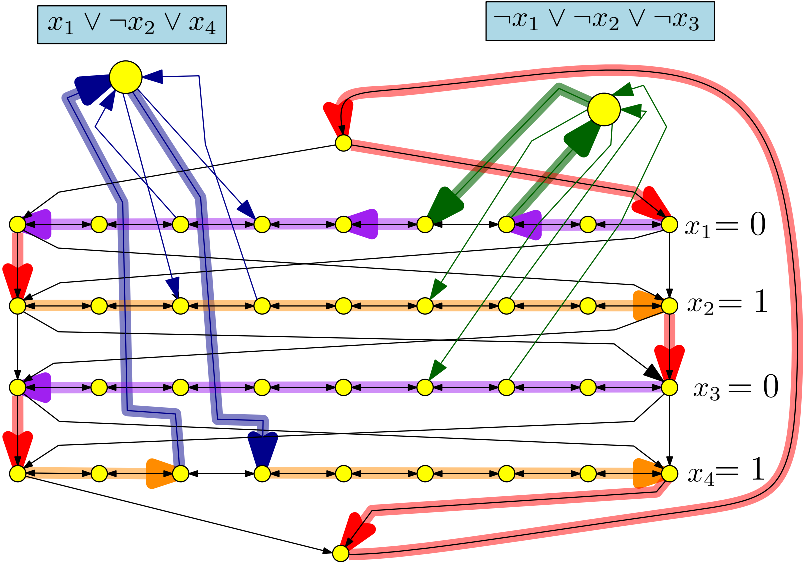
(for now)

23.3.4

If there is a Hamiltonian cycle $\implies$ $\exists$satisfying assignment

Reduction: Hamiltonian cycle $\implies \exists$ satisfying assignment

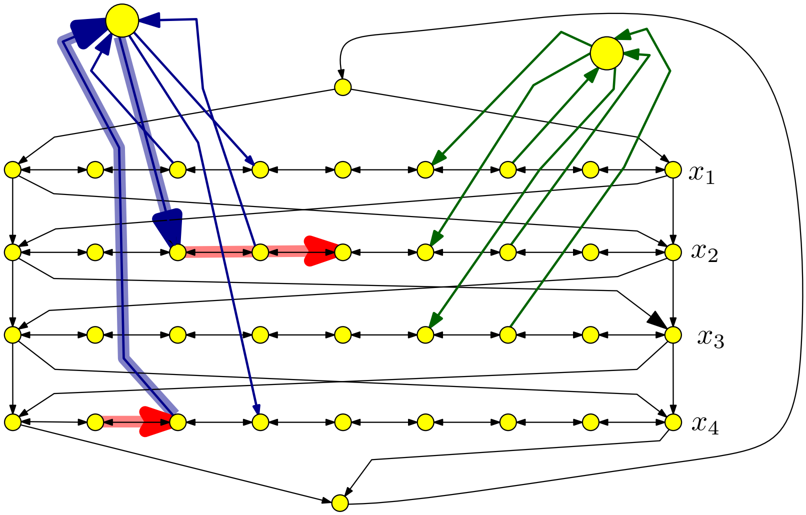
We are given a Hamiltonian cycle in $\boldsymbol{G_\varphi}$:



Want to extract satisfying assignment...

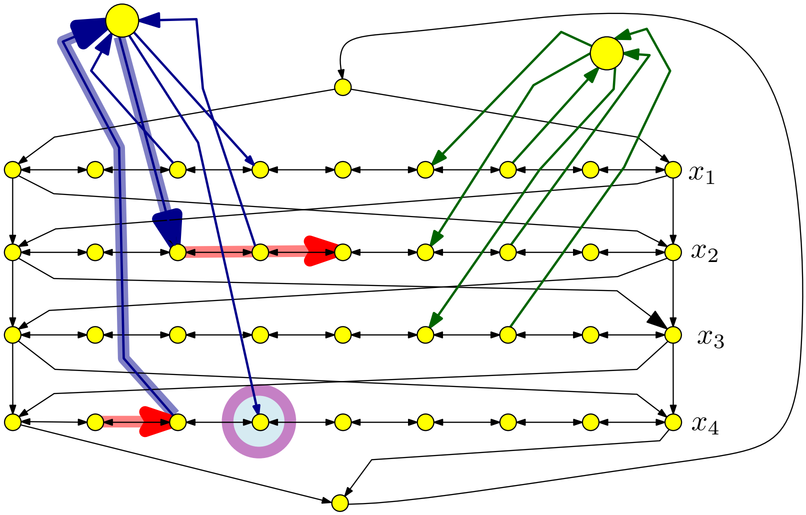
Reduction: Hamiltonian cycle $\implies \exists$ satisfying assignment

No shenanigan: Hamiltonian cycle can not leave a row in the middle



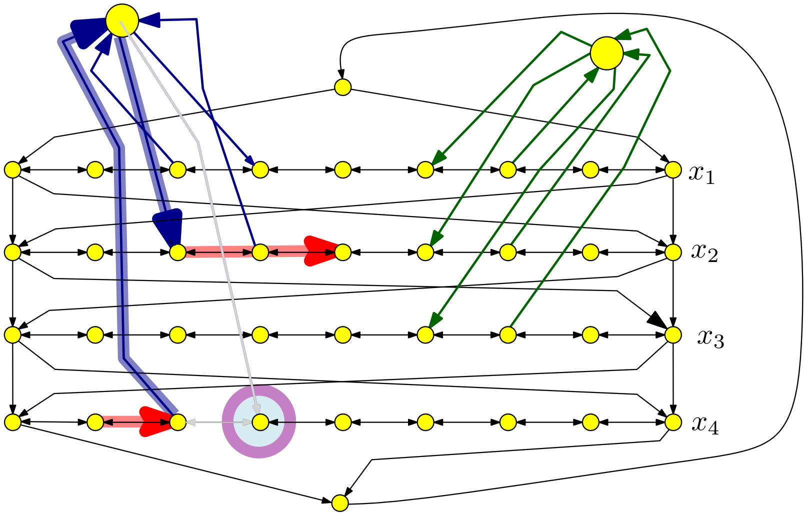
Reduction: Hamiltonian cycle $\Longrightarrow$ $\exists$ satisfying assignment

No shenanigan: Hamiltonian cycle can not leave a row in the middle



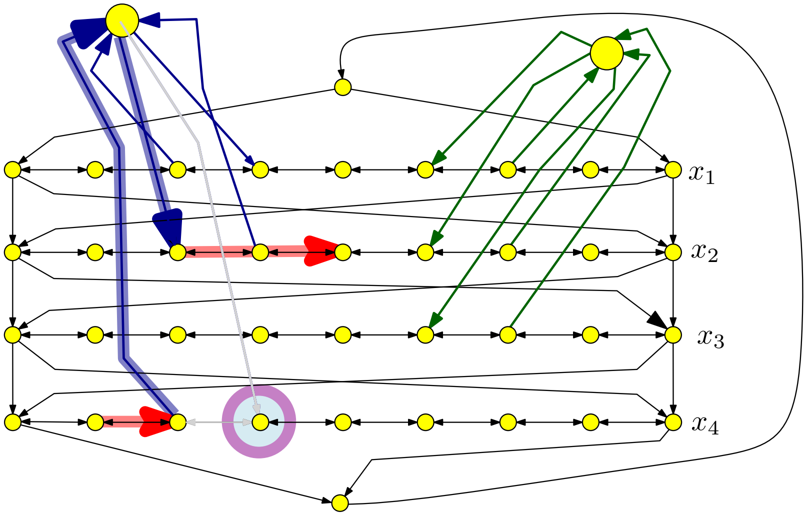
Reduction: Hamiltonian cycle $\implies$ $\exists$ satisfying assignment

No shenanigan: Hamiltonian cycle can not leave a row in the middle



Reduction: Hamiltonian cycle $\implies$ $\exists$ satisfying assignment

No shenanigan: Hamiltonian cycle can not leave a row in the middle



Conclude: Hamiltonian cycle must go through each row completely from left to right, or right to left. As such, can be interpreted as a valid assignment.

Hamiltonian Cycle $\Rightarrow$ Satisfying assignment

Suppose Π is a Hamiltonian cycle in G_φ

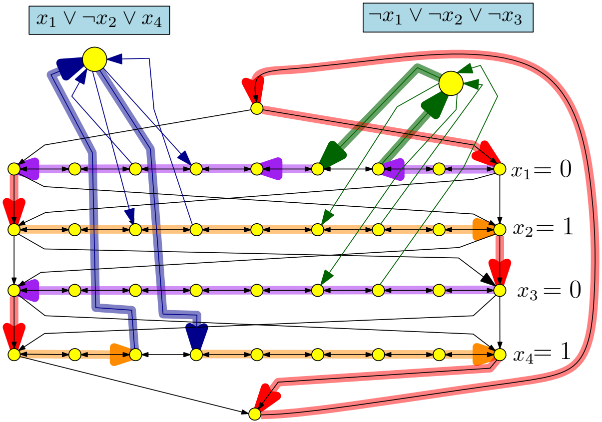
- If Π enters c_j (vertex for clause C_j) from vertex $3j$ on path i then it must leave the clause vertex on edge to $3j+1$ on the <u>same path i</u>
  - If not, then only unvisited neighbor of $3j+1$ on path i is $3j+2$
  - Thus, we don't have two unvisited neighbors (one to enter from, and the other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

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

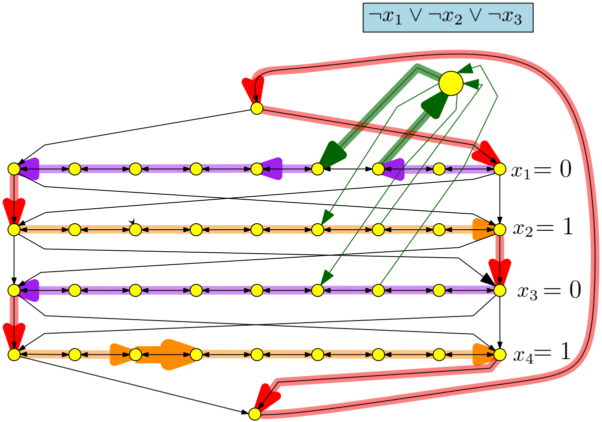
Hamiltonian Cycle $\implies$ Satisfying assignment (contd)

- 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



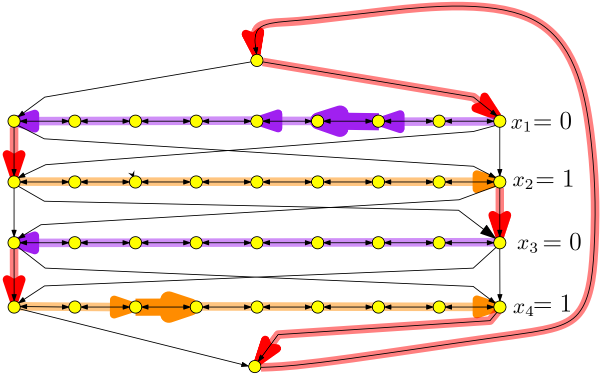
Hamiltonian Cycle $\implies$ Satisfying assignment (contd)

- 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



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



Correctness Proof

We just proved:

Lemma 23.2.

G_φ has a Hamiltonian cycle $\implies$ φ has a satisfying assignment α.

Lemma 23.3.

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

Proof.

Follows from **Lemma 23.1** and **Lemma 23.2**. □

Correctness Proof

We just proved:

Lemma 23.2.

G_φ has a Hamiltonian cycle $\implies$ φ has a satisfying assignment α.

Lemma 23.3.

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

Proof.

Follows from **Lemma 23.1** and **Lemma 23.2** . □

Summary

What we did:

1. Showed that **Directed Hamiltonian Cycle** is in **NP**.
2. Provided a polynomial time reduction from **3SAT** to **Directed Hamiltonian Cycle**.
3. Proved that φ satisfiable $\iff$ G_φ is Hamiltonian.

Theorem 23.4.

*The problem **Hamiltonian Cycle** in directed graphs is **NP-Complete**.*

Summary

What we did:

1. Showed that **Directed Hamiltonian Cycle** is in **NP**.
2. Provided a polynomial time reduction from **3SAT** to **Directed Hamiltonian Cycle**.
3. Proved that φ satisfiable $\iff$ G_φ is Hamiltonian.

Theorem 23.4.

*The problem **Hamiltonian Cycle** in directed graphs is **NP-Complete**.*

THE END

...

(for now)

23.4
Hamiltonian cycle in undirected graph

Hamiltonian Cycle

Problem 23.1.

Input *Given undirected graph* $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$

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

NP-Completeness

Theorem 23.2.

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

Proof.

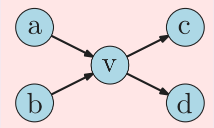
- The problem is in **NP**; proof left as exercise.
- Hardness proved by reducing Directed Hamiltonian Cycle to this problem □

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

- 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})

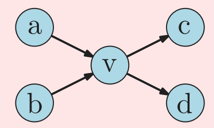


Reduction Sketch

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})

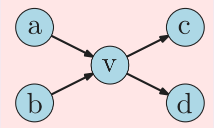


Reduction Sketch

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})

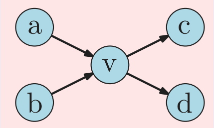


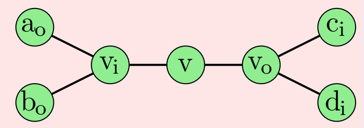
Reduction Sketch

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

Reduction

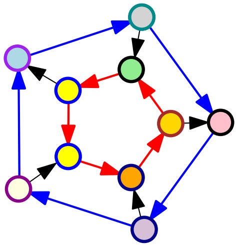
- Replace each vertex v by 3 vertices: v_{in}, v, and v_{out}
- A directed edge (a, b) is replaced by edge (a_{out}, b_{in})



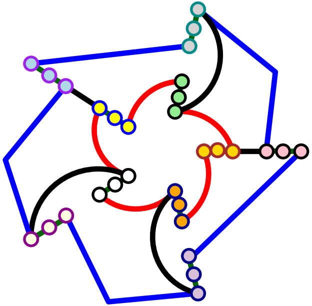


Hamiltonian cycle reduction

Undirected to directed case

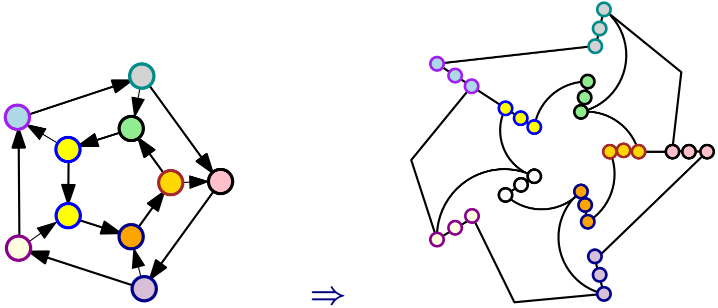


$\Rightarrow$



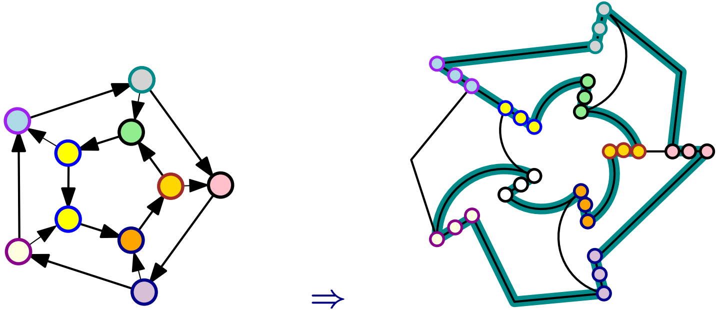
Hamiltonian cycle reduction

Undirected to directed case



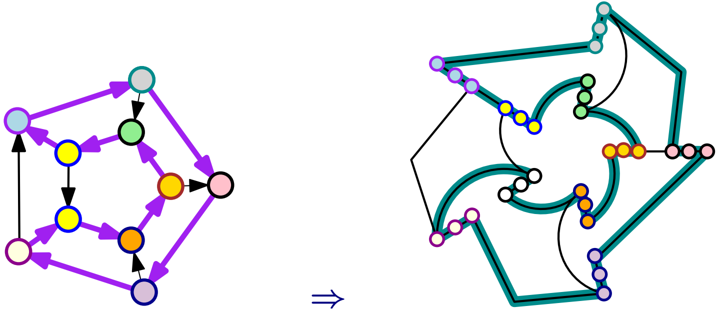
Hamiltonian cycle reduction

Undirected to directed case



Hamiltonian cycle reduction

Undirected to directed case



Reduction: Wrap-up

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

THE END

…

(for now)