COMP 382: Reasoning about Algorithms

More NP-Complete Reductions and Approximation Algorithms

Prof. Maryam Aliakbarpour

co-instructors: Prof. Anjum Chida & Prof. Konstantinos Mamouras

November 25, 2025

Today's Lecture

1. NP-Complete Problems

1.1 HAMILTONIAN-CYCLE Is NP-Complete

1.2 Traveling Salesman Problem (TSP) is NP-complete

2. Approximation Algorithms

2.1 2-Approximation Algorithm for Vertex Cover

2.2 (In)Approximability of TSP

2.3 Metric TSP and Its Approximation Algorithm

Reading:

- Chapter 12 of the *Algorithms* book [Erickson, 2019]

Content adapted from the same reference.

NP-Hard and NP-Complete Problems

NP-Hard Problem

A problem B is **NP-hard** if for every problem $A \in \text{NP}$, A reduces to B.

NP-Hard and NP-Complete Problems

NP-Hard Problem

A problem B is **NP-hard** if for every problem $A \in \text{NP}$, A reduces to B.

NP-Complete Problem

A problem B is **NP-complete** if:

1. $B \in \text{NP}$, and
2. For every problem $A \in \text{NP}$, A reduces to B.

The NP-Completeness Recipe

To show a new problem B is NP-complete, start from a known NP-complete problem A. Show a polynomial-time reduction from A to B.

$$\left.\begin{array}{ll} \text{A is NP-complete:} & \text{NP} \leq_p A \\ \text{A poly-time reduction from A to B:} & A \leq_p B \end{array}\right\} \implies \text{NP} \leq_p B$$

The NP-Completeness Recipe

To show a new problem B is NP-complete, start from a known NP-complete problem A. Show a polynomial-time reduction from A to B.

$$\left.\begin{array}{ll} \text{A is NP-complete:} & \text{NP} \leq_p A \\ \text{A poly-time reduction from A to B:} & A \leq_p B \end{array}\right\} \implies \text{NP} \leq_p B$$

The NP-Completeness Recipe

To show a new problem B is NP-complete, start from a known NP-complete problem A. Show a polynomial-time reduction from A to B.

A is NP-complete: $\text{NP} \leq_p A$

A poly-time reduction from A to B: $A \leq_p B$

$\Longrightarrow \text{NP} \leq_p B$

B is NP-hard.

The NP-Completeness Recipe

To show a new problem B is NP-complete, start from a known NP-complete problem A. Show a polynomial-time reduction from A to B.

A is NP-complete: $\text{NP} \leq_p A$

A poly-time reduction from A to B: $A \leq_p B$

$\Longrightarrow \text{NP} \leq_p B$

B is NP-hard.

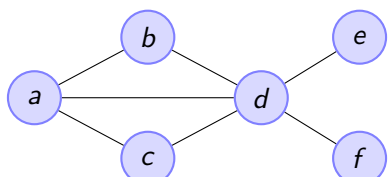
If we also show that B is in NP, then $\Longrightarrow$ B is NP-complete

Recap: Vertex Covers

Let $G = (V, E)$ be a simple undirected graph.

A vertex cover in G is a subset $C \subseteq V$ such that every edge of G has at least one endpoint in C.

Equivalently: every edge is "touched" by C.

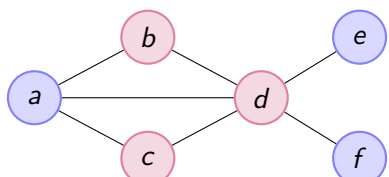


Recap: Vertex Covers

Let $G = (V, E)$ be a simple undirected graph.

A vertex cover in G is a subset $C \subseteq V$ such that every edge of G has at least one endpoint in C.

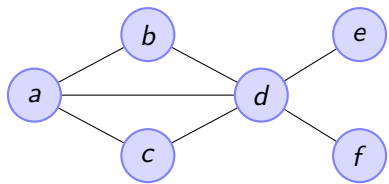
Equivalently: every edge is "touched" by C.



A vertex cover $C = \{b,\ c,\ d\}$ touches all edges.

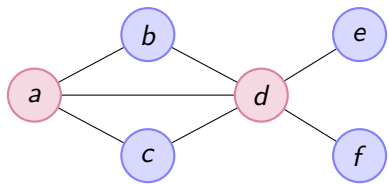
Minimum Vertex Cover Problem

VERTEX-COVER: Given a graph G and an integer k, does G contain a vertex cover of size at most k?



Minimum Vertex Cover Problem

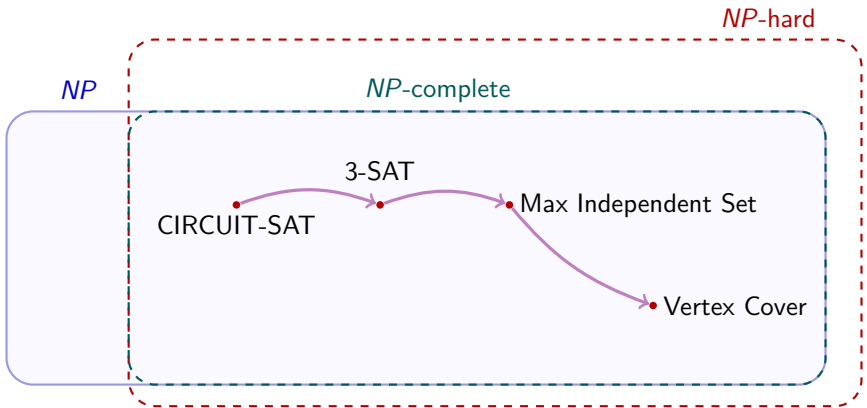
VERTEX-COVER: Given a graph G and an integer k, does G contain a vertex cover of size at most k?



Here $\{a,\ d\}$ is a vertex cover of size 2.

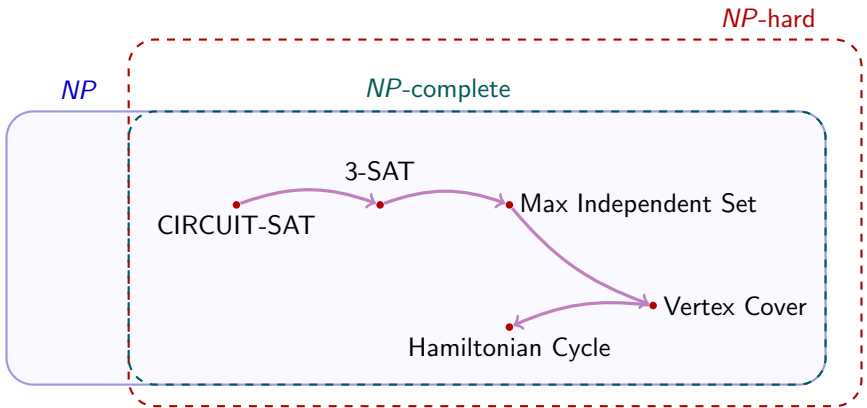
A Small NP-Completeness Family Portrait

Once one natural problem is shown NP-complete, the others follow by reductions.



A Small NP-Completeness Family Portrait

Once one natural problem is shown NP-complete, the others follow by reductions.

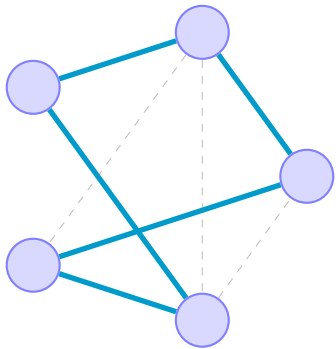


HAMILTONIAN-CYCLE Is NP-Complete

A reduction from VERTEX-COVER to HAMILTONIAN-CYCLE

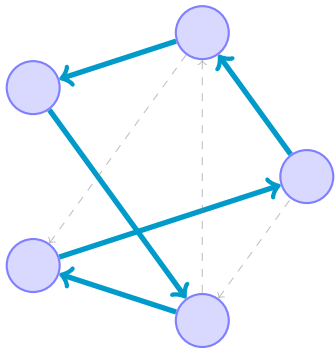
Hamiltonian Cycle

- A **Hamiltonian cycle** in a graph is a cycle that visits every vertex **exactly once**.
- Not the same as an Euler cycle (which uses every *edge* once).



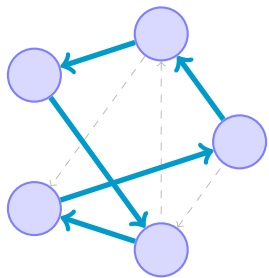
Directed Hamiltonian Cycle

- A **directed Hamiltonian cycle** in a graph is a cycle that visits every vertex **exactly once** and traverses the edges in their direction.



Directed Hamiltonian Cycle

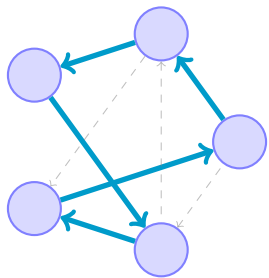
HAMILTONIAN-CYCLE: Given a directed graph H, does it contain a Hamiltonian cycle?



Directed Hamiltonian Cycle

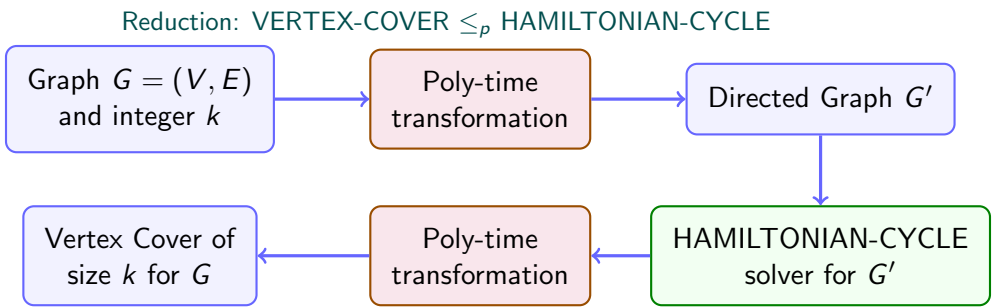
HAMILTONIAN-CYCLE: Given a directed graph H, does it contain a Hamiltonian cycle?

- We show directed HAMILTONIAN-CYCLE is NP-complete.
- The **undirected** version of this problem is also NP-complete. (Exercise)



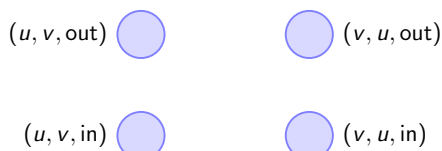
Plan: Reduce VERTEX-COVER to Directed HAMILTONIAN-CYCLE

- Input on the Vertex Cover side: An undirected graph $G = (V, E)$ and an integer k (target cover size).
- We will build a directed graph G' such that: G has a vertex cover of size k if and only if G' has a directed Hamiltonian cycle.



The Edge Gadget

For each edge $uv \in G$, we create gadget with four vertices.

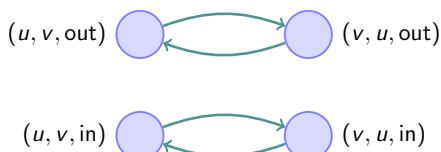


The Edge Gadget

For each edge $uv \in G$, we create gadget with four vertices.

We add the following edges:

- **Crossing edges:**
  $(u, v, \text{in}) \leftrightarrow (v, u, \text{in})$
  $(u, v, \text{in}) \leftrightarrow (v, u, \text{out})$

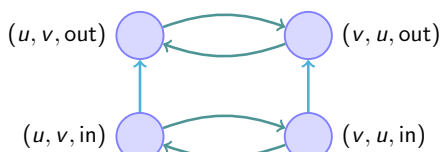


The Edge Gadget

For each edge $uv \in G$, we create gadget with four vertices.

We add the following edges:

- **Crossing edges:**
  $(u, v, \text{in}) \leftrightarrow (v, u, \text{in})$
  $(u, v, \text{in}) \leftrightarrow (v, u, \text{out})$
- **Internal chain edges:**
  $(u, v, \text{in}) \rightarrow (u, v, \text{out})$
  $(v, u, \text{in}) \rightarrow (v, u, \text{out})$

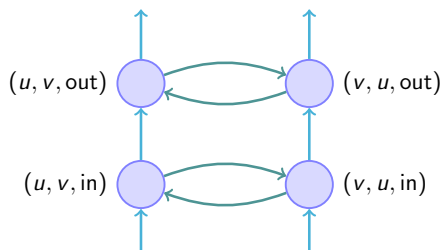


The Edge Gadget

For each edge $uv \in G$, we create gadget with four vertices.

We add the following edges:

- **Crossing edges:**
  $(u, v, \text{in}) \leftrightarrow (v, u, \text{in})$
  $(u, v, \text{in}) \leftrightarrow (v, u, \text{out})$
- **Internal chain edges:**
  $(u, v, \text{in}) \rightarrow (u, v, \text{out})$
  $(v, u, \text{in}) \rightarrow (v, u, \text{out})$
- **External chain edges:**
  connect this gadget to others.

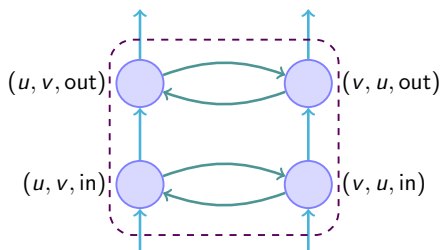


The Edge Gadget

For each edge $uv \in G$, we create gadget with four vertices.

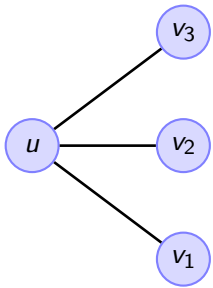
We add the following edges:

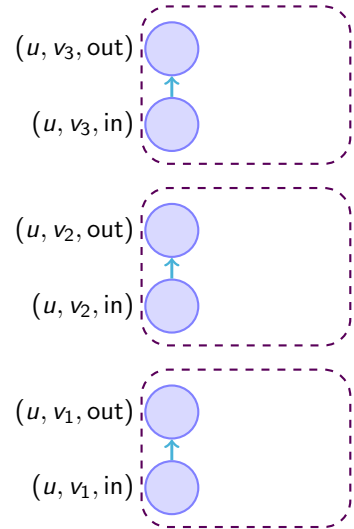
- **Crossing edges:**
  $(u, v, \text{in}) \leftrightarrow (v, u, \text{in})$
  $(u, v, \text{in}) \leftrightarrow (v, u, \text{out})$
- **Internal chain edges:**
  $(u, v, \text{in}) \rightarrow (u, v, \text{out})$
  $(v, u, \text{in}) \rightarrow (v, u, \text{out})$
- **External chain edges:**
  connect this gadget to others.



Vertex Chains

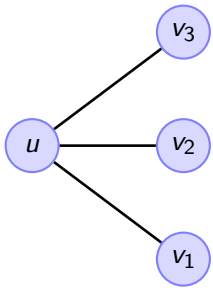
For each vertex u in G, connect the edge gadgets involving u into a directed chain.

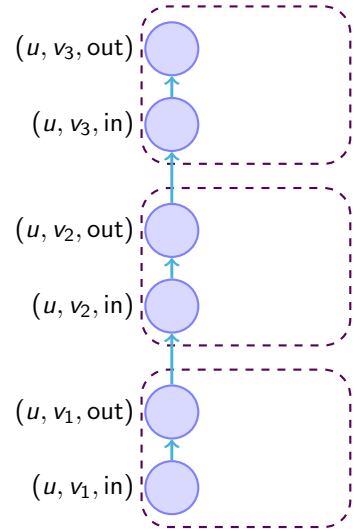




Vertex Chains

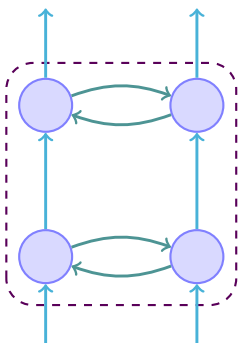
For each vertex u in G, connect the edge gadgets involving u into a directed chain.





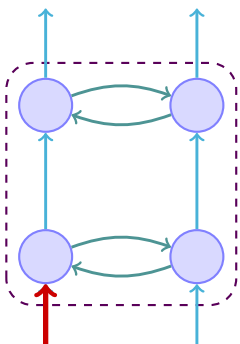
Possible Traversals

The Hamiltonian cycle must assume one of three states (straight or detour), corresponding to selecting u, v, or both.



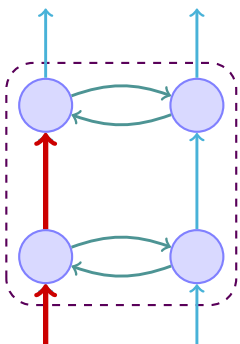
Possible Traversals

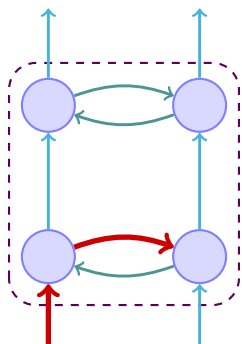
The Hamiltonian cycle must assume one of three states (straight or detour), corresponding to selecting u, v, or both.



Possible Traversals

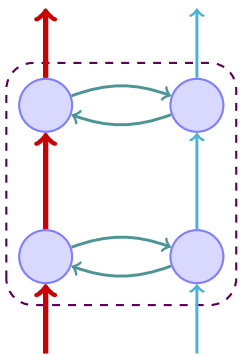
The Hamiltonian cycle must assume one of three states (straight or detour), corresponding to selecting u, v, or both.

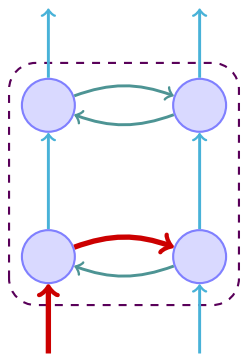




Possible Traversals

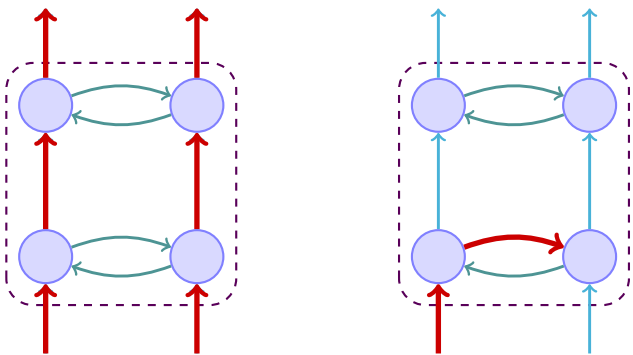
The Hamiltonian cycle must assume one of three states (straight or detour), corresponding to selecting u, v, or both.





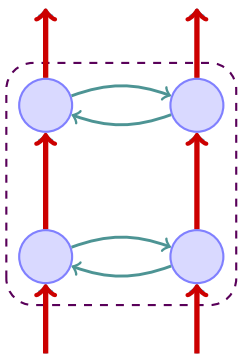
Possible Traversals

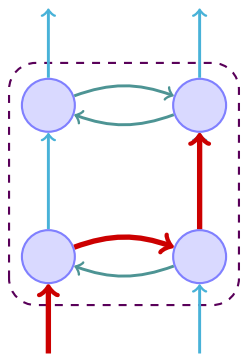
The Hamiltonian cycle must assume one of three states (straight or detour), corresponding to selecting u, v, or both.



Possible Traversals

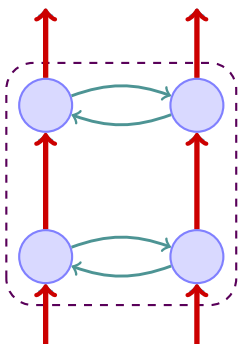
The Hamiltonian cycle must assume one of three states (straight or detour), corresponding to selecting u, v, or both.

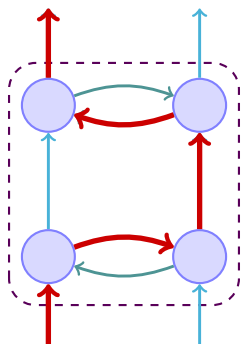




Possible Traversals

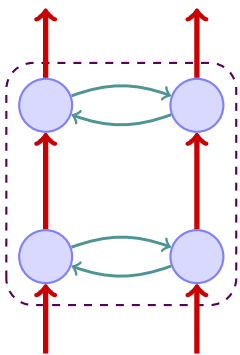
The Hamiltonian cycle must assume one of three states (straight or detour), corresponding to selecting u, v, or both.

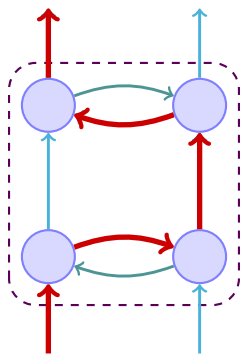


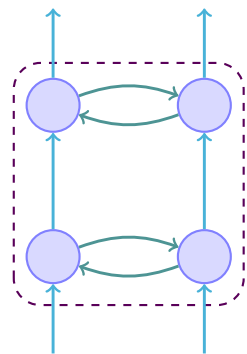


Possible Traversals

The Hamiltonian cycle must assume one of three states (straight or detour), corresponding to selecting u, v, or both.

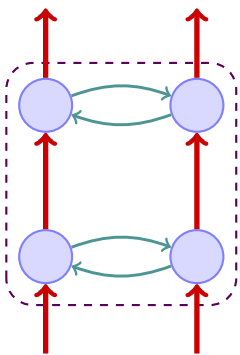


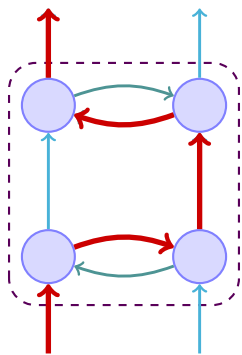


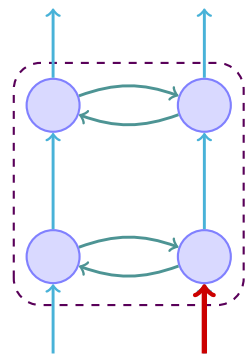


Possible Traversals

The Hamiltonian cycle must assume one of three states (straight or detour), corresponding to selecting u, v, or both.

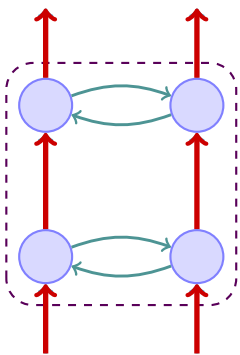


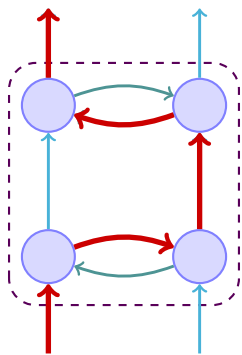


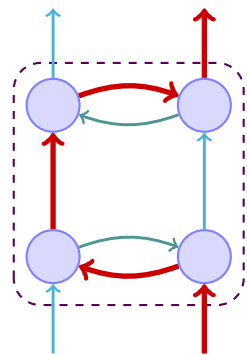


Possible Traversals

The Hamiltonian cycle must assume one of three states (straight or detour), corresponding to selecting u, v, or both.







Why Only Three Traversals?

If a Hamiltonian cycle ever crosses from the u-side to the v-side (or vice versa), it must also cross back. Otherwise, one vertex on the original side would remain unvisited, making a Hamiltonian cycle impossible.

Therefore, **every gadget admits exactly three traversals:**

1. Straight through on both sides,
2. A detour on the u-side,
3. A detour on the v-side.

These are the three configurations shown on the previous slide.

Interpreting the Gadget Behavior

Key idea: Once the Hamiltonian cycle enters u's chain, the structure of the gadgets forces it to traverse the *entire* chain.

- The cycle cannot skip a gadget in the chain without leaving a vertex unvisited.
- Thus, entering u's chain means "committing" to the vertex u.

Interpreting the Gadget Behavior

Key idea: Once the Hamiltonian cycle enters u's chain, the structure of the gadgets forces it to traverse the *entire* chain.

- The cycle cannot skip a gadget in the chain without leaving a vertex unvisited.
- Thus, entering u's chain means "committing" to the vertex u.

Interpretation: Passing through the full chain of u corresponds exactly to selecting the vertex u in the Vertex Cover instance.

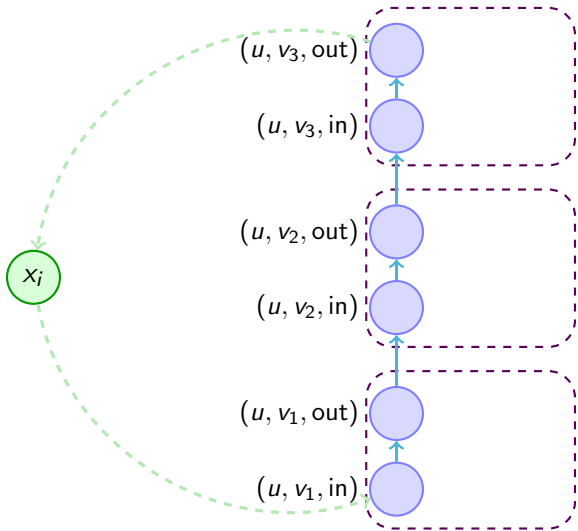
1. Straight through on both sides $\rightarrow$ selecting both u and v,
2. A detour on the u-side $\rightarrow$ selecting u,
3. A detour on the v-side $\rightarrow$ selecting v.

Cover Vertices

We are looking for a mechanism **to enforce the selection of k chains**.

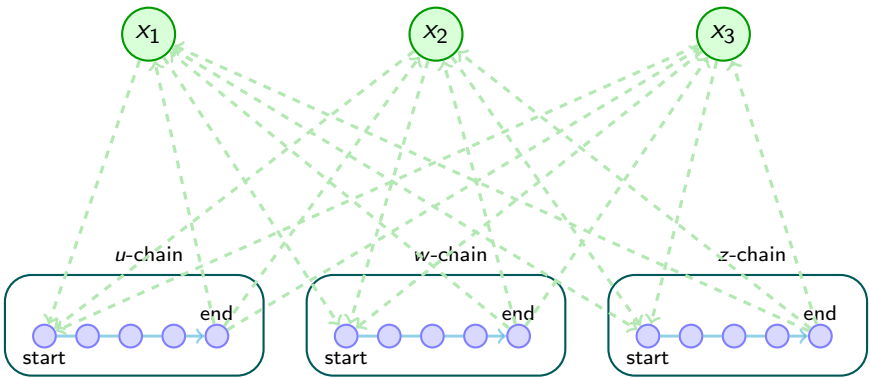
- Introduce k special vertices $x_1, \ldots, x_k$.
- Each x_i has:
  - edges to the **start** of every vertex chain,
  - edges from the **end** of every vertex chain.

Cover Vertices



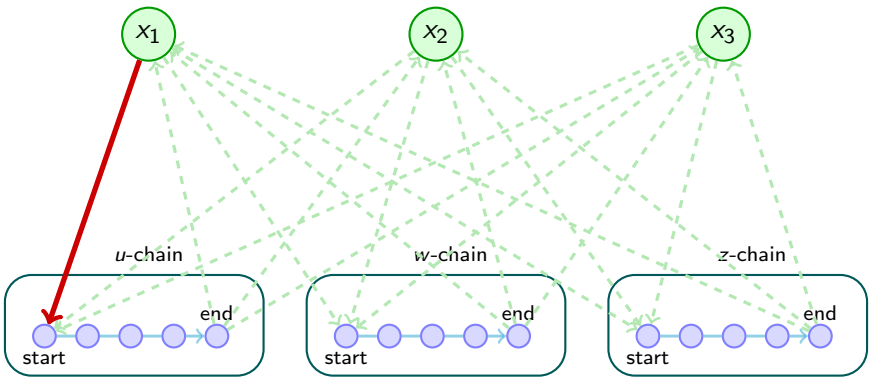
Cover Vertices Select Chains

Any Hamiltonian cycle must enter exactly k vertex chains—via the x_i nodes. Those k chains correspond to chosen vertices of the vertex cover.



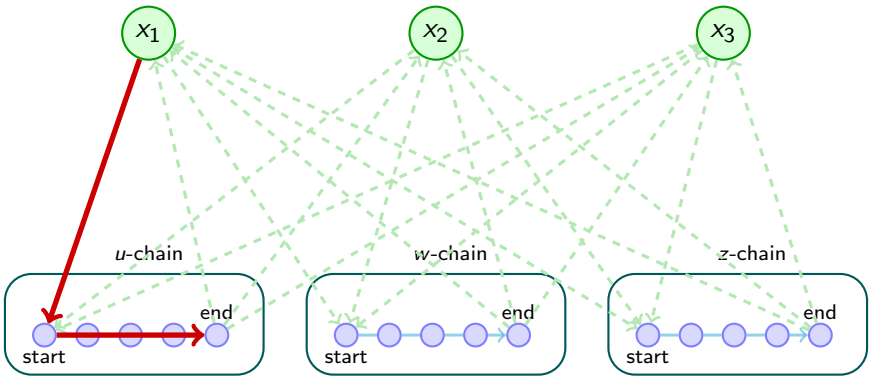
Cover Vertices Select Chains

Any Hamiltonian cycle must enter exactly k vertex chains—via the x_i nodes. Those k chains correspond to chosen vertices of the vertex cover.



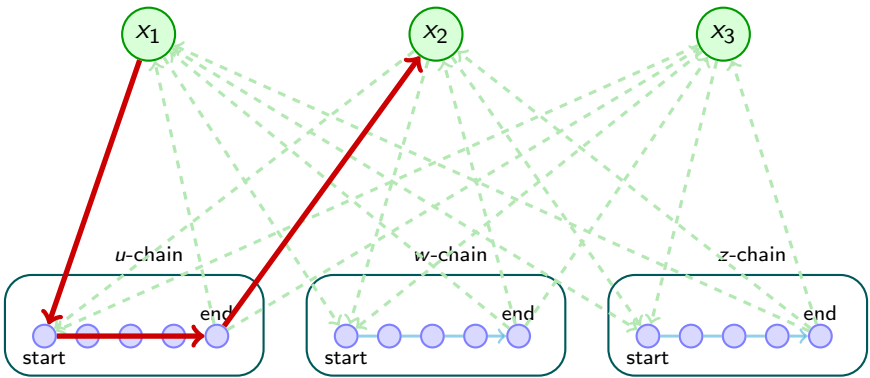
Cover Vertices Select Chains

Any Hamiltonian cycle must enter exactly k vertex chains—via the x_i nodes. Those k chains correspond to chosen vertices of the vertex cover.



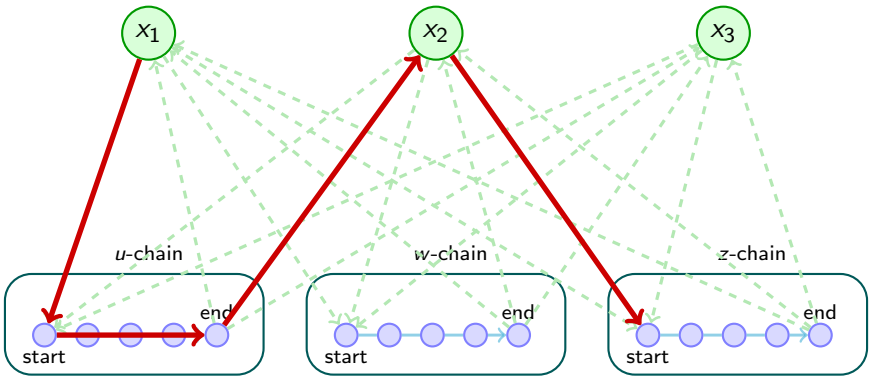
Cover Vertices Select Chains

Any Hamiltonian cycle must enter exactly k vertex chains—via the x_i nodes. Those k chains correspond to chosen vertices of the vertex cover.



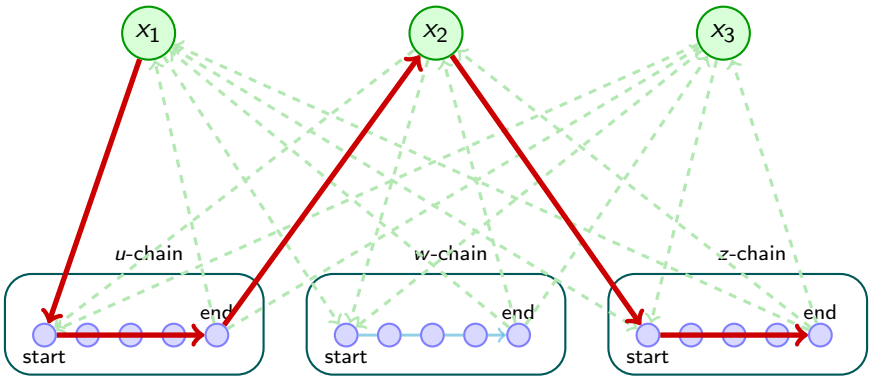
Cover Vertices Select Chains

Any Hamiltonian cycle must enter exactly k vertex chains—via the x_i nodes. Those k chains correspond to chosen vertices of the vertex cover.



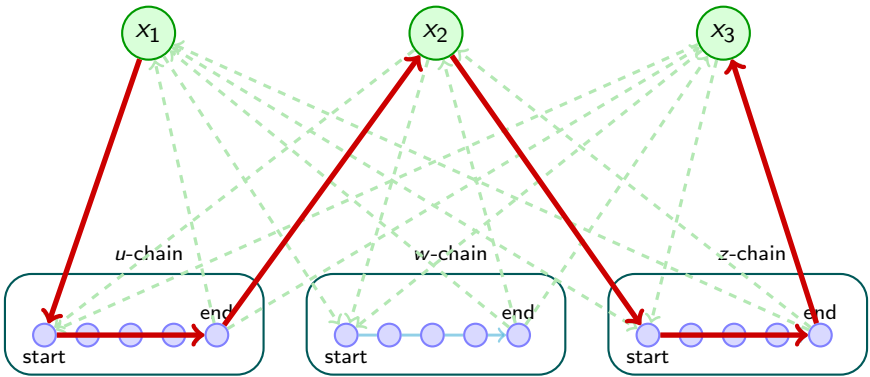
Cover Vertices Select Chains

Any Hamiltonian cycle must enter exactly k vertex chains—via the x_i nodes. Those k chains correspond to chosen vertices of the vertex cover.



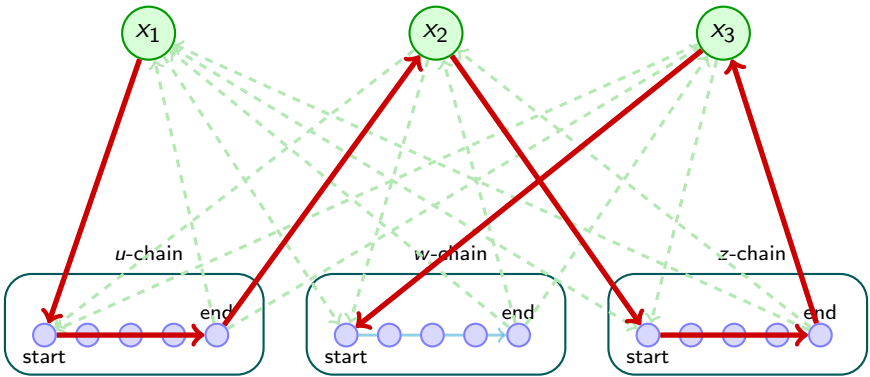
Cover Vertices Select Chains

Any Hamiltonian cycle must enter exactly k vertex chains—via the x_i nodes. Those k chains correspond to chosen vertices of the vertex cover.



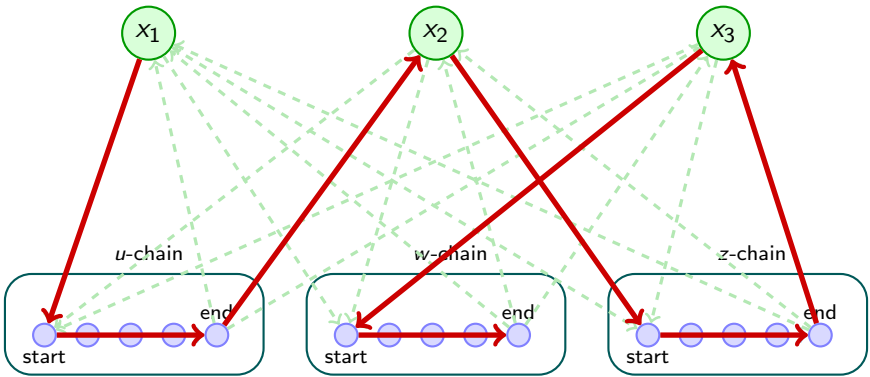
Cover Vertices Select Chains

Any Hamiltonian cycle must enter exactly k vertex chains—via the x_i nodes. Those k chains correspond to chosen vertices of the vertex cover.



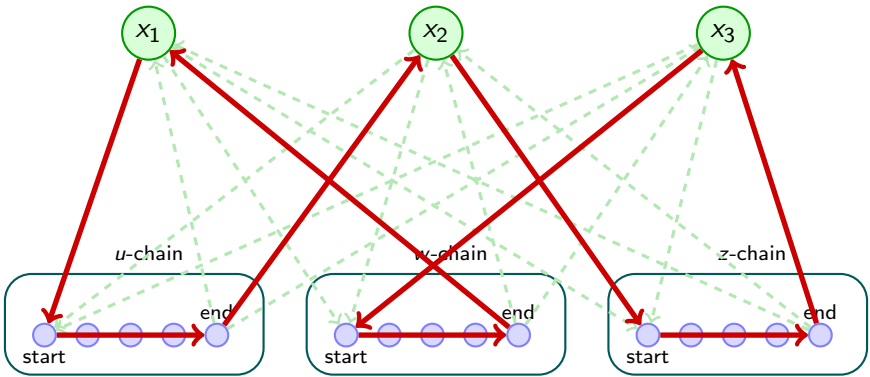
Cover Vertices Select Chains

Any Hamiltonian cycle must enter exactly k vertex chains—via the x_i nodes. Those k chains correspond to chosen vertices of the vertex cover.

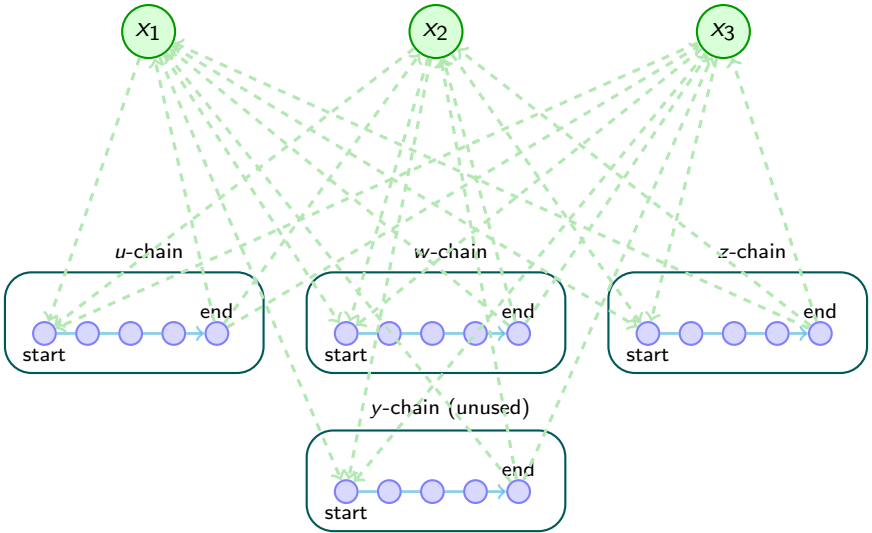


Cover Vertices Select Chains

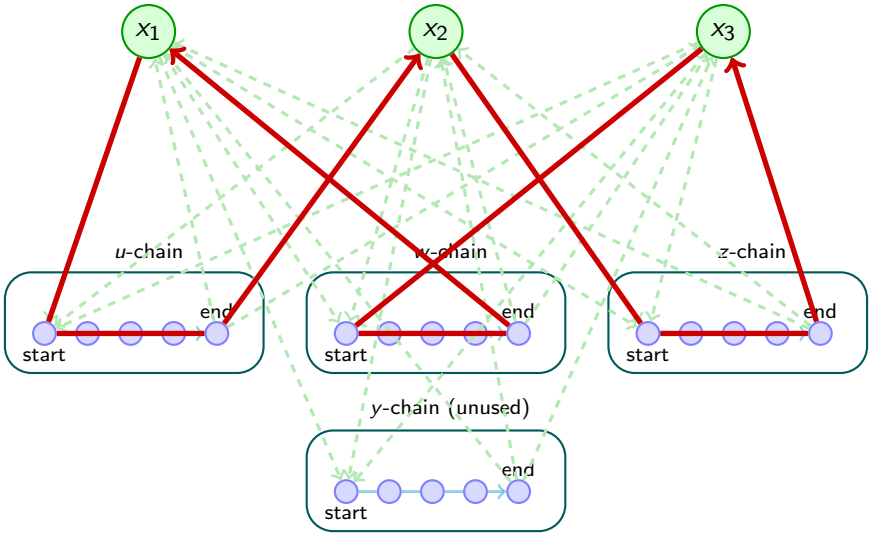
Any Hamiltonian cycle must enter exactly k vertex chains—via the x_i nodes. Those k chains correspond to chosen vertices of the vertex cover.



Can We Have Unused Chains?

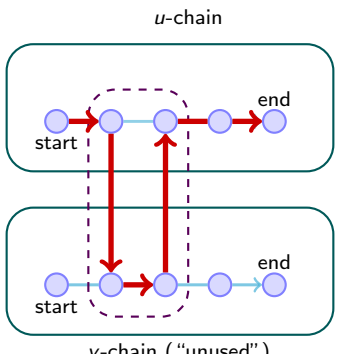


Can We Have Unused Chains?



Zoom: Detours cover an “unused” chain

As we traverse other vertices’ chains, each gadget’s detour dips to the opposite side and back. Thus an “unused” chain still has all its vertices visited via these detours.



Correctness: Vertex Cover $\Longrightarrow$ Hamiltonian cycle

Suppose $C = \{u_1, \ldots, u_k\}$ is a vertex cover.

- We traverse vertex chains in order $u_1,\ u_2,\ \ldots,\ u_k$.

Correctness: Vertex Cover $\implies$ Hamiltonian cycle

Suppose $C = \{u_1, \ldots, u_k\}$ is a vertex cover.

- We traverse vertex chains in order $u_1,\ u_2,\ \ldots,\ u_k$.
- Enter each chain from cover vertex x_i and exit to x_{i+1}, assuming $x_1 = x_{k+1}$:

$$x_i \to u_i \to x_{i+1}$$

Correctness: Vertex Cover $\implies$ Hamiltonian cycle

Suppose $C = \{u_1, \ldots, u_k\}$ is a vertex cover.

- We traverse vertex chains in order $u_1,\ u_2,\ \ldots,\ u_k$.
- Enter each chain from cover vertex x_i and exit to x_{i+1}, assuming $x_1 = x_{k+1}$:

$$x_i \to u_i \to x_{i+1}$$

- For each edge gadget (u, v):

Correctness: Vertex Cover $\Longrightarrow$ Hamiltonian cycle

Suppose $C = \{u_1, \ldots, u_k\}$ is a vertex cover.

- We traverse vertex chains in order $u_1,\ u_2,\ \ldots,\ u_k$.
- Enter each chain from cover vertex x_i and exit to x_{i+1}, assuming $x_1 = x_{k+1}$:

$$x_i \to u_i \to x_{i+1}$$

- For each edge gadget (u, v):
  - If $v \in C$, go straight through $(u, v, \text{in}) \to (u, v, \text{out})$.

Correctness: Vertex Cover $\implies$ Hamiltonian cycle

Suppose $C = \{u_1, \ldots, u_k\}$ is a vertex cover.

- We traverse vertex chains in order $u_1,\ u_2,\ \ldots,\ u_k$.
- Enter each chain from cover vertex x_i and exit to x_{i+1}, assuming $x_1 = x_{k+1}$:

$$x_i \to u_i \to x_{i+1}$$

- For each edge gadget (u, v):
  - If $v \in C$, go straight through $(u, v, \text{in}) \to (u, v, \text{out})$.
  - If $v \notin C$, take the detour through the v-side to ensure the gadget is covered.

Correctness: Vertex Cover $\Longrightarrow$ Hamiltonian cycle

Suppose $C = \{u_1, \ldots, u_k\}$ is a vertex cover.

- We traverse vertex chains in order $u_1,\ u_2,\ \ldots,\ u_k$.
- Enter each chain from cover vertex x_i and exit to x_{i+1}, assuming $x_1 = x_{k+1}$:

$$x_i \to u_i \to x_{i+1}$$

- For each edge gadget (u, v):
  - If $v \in C$, go straight through $(u, v, \text{in}) \to (u, v, \text{out})$.
  - If $v \notin C$, take the detour through the v-side to ensure the gadget is covered.
- This builds a Hamiltonian cycle in H.

Correctness: Hamiltonian cycle $\implies$ Vertex Cover

Given Hamiltonian cycle C in H:

- C must use an outgoing edge from each x_i into some vertex chain.
- Once C enters chain of u, it must traverse **all** gadgets of u in order.
- Thus exactly k chains are fully traversed.
- Let S be the set of vertices whose chains are chosen.
- We show that S is a vertex cover:
  - For any edge $uv \in G$, the cycle must visit (u, v, in).
  - Therefore either u or v's chain must be selected.
- Hence $|S| = k$ and S covers all edges.

Conclusion

- Reduction transforms (G, k) into H in $O(|V| + |E|)$ time.
- H has a Hamiltonian cycle $\iff$ G has a vertex cover of size k.
- Therefore **Directed HAMILTONIAN-CYCLE** is NP-hard.

Conclusion

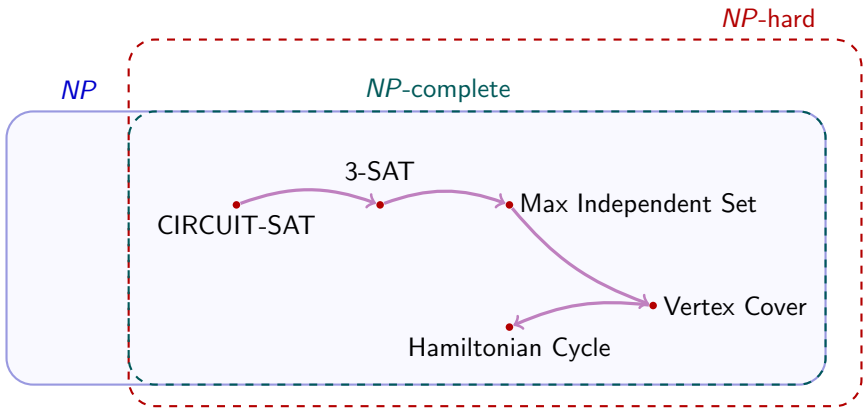
- Reduction transforms (G, k) into H in $O(|V| + |E|)$ time.
- H has a Hamiltonian cycle $\iff$ G has a vertex cover of size k.
- Therefore **Directed HAMILTONIAN-CYCLE** is NP-hard.
- Since the problem is in NP and our reduction proves NP-hardness, **Directed HAMILTONIAN-CYCLE is NP-complete**.

Conclusion

- Reduction transforms (G, k) into H in $O(|V| + |E|)$ time.
- H has a Hamiltonian cycle $\iff$ G has a vertex cover of size k.
- Therefore **Directed HAMILTONIAN-CYCLE** is NP-hard.
- Since the problem is in NP and our reduction proves NP-hardness, **Directed HAMILTONIAN-CYCLE is NP-complete**.
- Also implies related problems:
  - Hamiltonian path is NP-hard.
  - Undirected variants also NP-hard via simple modifications. (See homework)

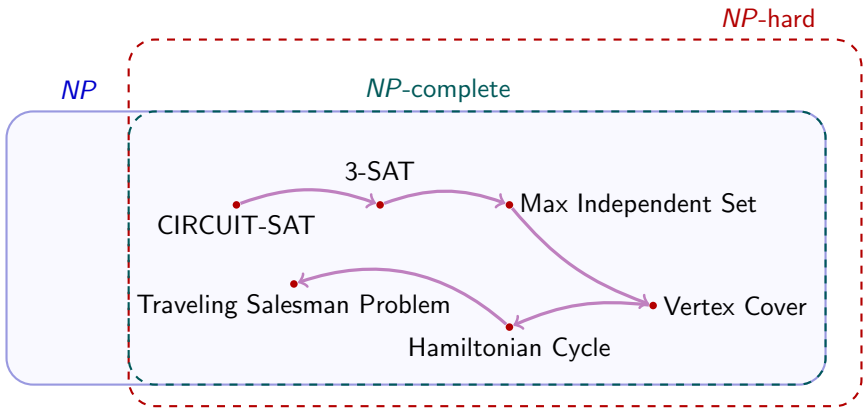
A Small NP-Completeness Family Portrait

Once one natural problem is shown NP-complete, the others follow by reductions.



A Small NP-Completeness Family Portrait

Once one natural problem is shown NP-complete, the others follow by reductions.



Traveling Salesman Problem (TSP) is NP-complete

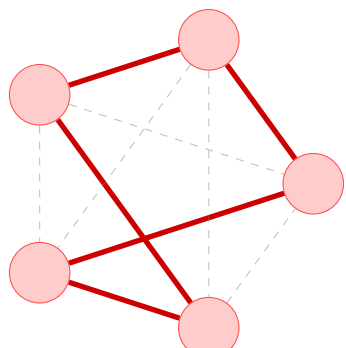
A reduction from Hamiltonian cycle to TSP

Traveling Salesman Problem (TSP)

Problem: Find a Hamiltonian cycle (visiting every vertex exactly once) of minimum total edge cost.

TSP: Given a weighted **complete** graph, is there a tour visiting every vertex exactly once with total cost $\leq B$?

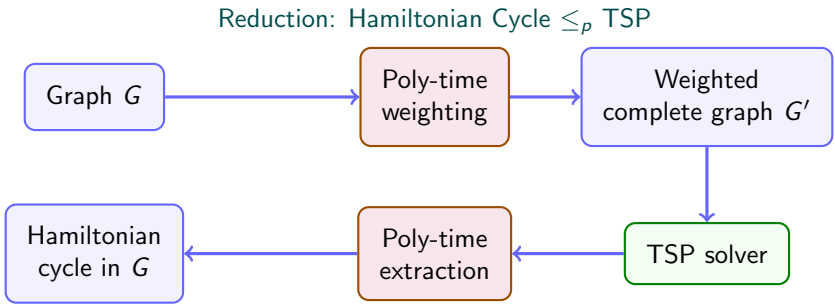
You can "complete" a non-complete graph by assigning weight ∞ (or a very large weight $> B$) to every missing edge.



Plan: Reduce Hamiltonian Cycle to TSP

- Input on the Hamiltonian Cycle side: a graph $G = (V, E)$.
- We will build a weighted complete graph G' and threshold B such that:

$$G \text{ has a Hamiltonian cycle} \iff G' \text{ has a TSP tour of total cost } \leq B.$$



Reduction: Hamiltonian Cycle $\Leftrightarrow$ TSP

Given an unweighted graph $G = (V, E)$, build a complete weighted graph G' by

$$w(u, v) = \begin{cases} 1 & \text{if } \{u, v\} \in E, \\ M & \text{otherwise}, \end{cases}$$

for some $M > 1$. Let $B = |V|$.

Claim. G has a Hamiltonian cycle $\Leftrightarrow$ G' has a TSP tour of total cost $\leq B$.

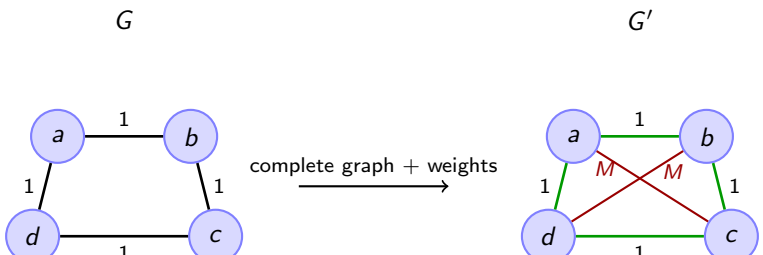
- If G has a Hamiltonian cycle, the corresponding tour in G' uses only weight-1 edges $\Rightarrow$ total cost $= |V| = B$.
- If G' has a tour of cost $\leq B$, it cannot use any weight-M edge (that would raise the cost to at least $B + M - 1 > B$). Thus the tour corresponds to a Hamiltonian cycle in G.

Thus

$$\text{HAMILTONIAN-CYCLE} \leq_p \text{TSP} \quad \Longrightarrow \quad \text{TSP is NP-complete.}$$

Reduction: Hamiltonian Cycle $\Leftrightarrow$ TSP

Any Hamiltonian Cycle in G' needs to take $\leq |V| = 4$ edges. To obtain total weight of at most four, it must avoid all red (weight-M) edges, since including even one such edge would push the cost above 4. Therefore only green edges can appear in a valid low-cost tour.



Approximation Algorithms

If we relax optimality, does it get easier?

Approximation Algorithms

NP-complete problems are hard to solve *exactly*. A natural question is:

If we stop insisting on the optimal solution, can we solve the problem efficiently?

Approximation Algorithms

NP-complete problems are hard to solve *exactly*. A natural question is:

If we stop insisting on the optimal solution, can we solve the problem efficiently?

Approximation Algorithms:

- Run in polynomial time.
- Always produce a *feasible* solution.
- Guarantee that the solution is within a factor α of the optimum.

Approximation Algorithms

NP-complete problems are hard to solve *exactly*. A natural question is:

If we stop insisting on the optimal solution, can we solve the problem efficiently?

Approximation Algorithms:

- Run in polynomial time.
- Always produce a *feasible* solution.
- Guarantee that the solution is within a factor α of the optimum.
- For minimization problems $\alpha > 1$, and smaller α means better approximation.

$$\mathsf{OPT} \leq \mathsf{cost}(\mathsf{ALG}) \leq \alpha \cdot \mathsf{OPT}.$$

- For maximization problems $\alpha < 1$, and larger α means better approximation.

$$\mathsf{OPT} \geq \mathsf{cost}(\mathsf{ALG}) \geq \alpha \cdot \mathsf{OPT}.$$

Approximability of Problems

Often we seek a solution that is *close* to optimal: a constant-factor approximation, or an α that grows slowly with n (e.g., $\log n$).

A natural question:

> Does every NP-hard optimization problem admit a polynomial-time approximation algorithm?

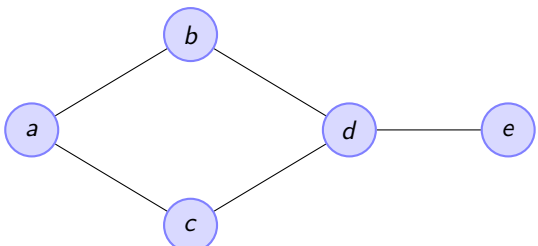
- Some problems *do* admit good approximations.
- Others remain hard even to approximate within any reasonable factor.

This leads to an entire area of study: **approximability theory**.

2-Approximation Algorithm for Vertex Cover

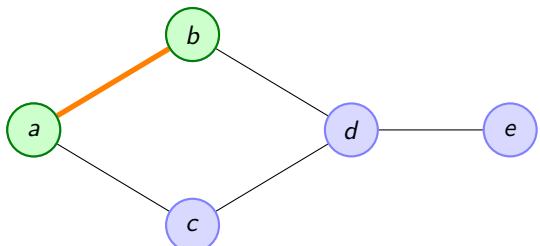
2-Approximation for Vertex Cover

The algorithm repeatedly picks an uncovered edge and adds *both* endpoints to the cover.



2-Approximation for Vertex Cover

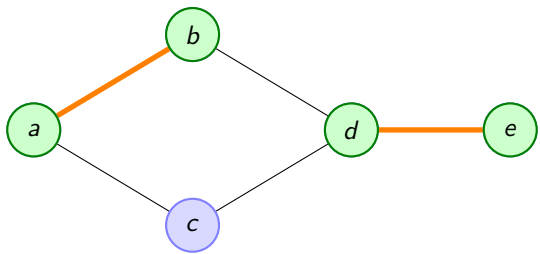
The algorithm repeatedly picks an uncovered edge and adds *both* endpoints to the cover.



Step 1: pick an uncovered edge (here (a, b)), add a and b.

2-Approximation for Vertex Cover

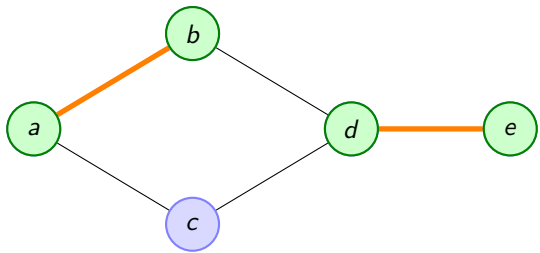
The algorithm repeatedly picks an uncovered edge and adds *both* endpoints to the cover.



Step 1: pick an uncovered edge (here (a, b)), add a and b.
Step 2: pick another uncovered edge (here (d, e)), add d and e.

2-Approximation for Vertex Cover

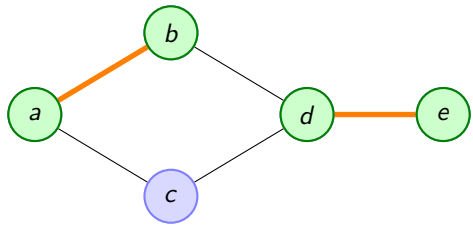
The algorithm repeatedly picks an uncovered edge and adds *both* endpoints to the cover.



Step 1: pick an uncovered edge (here (a, b)), add a and b.
Step 2: pick another uncovered edge (here (d, e)), add d and e.
All edges are covered. $\Rightarrow$ Final cover $= \{a, b, c, d\}$.

Why the Greedy Algorithm Is a 2-Approximation

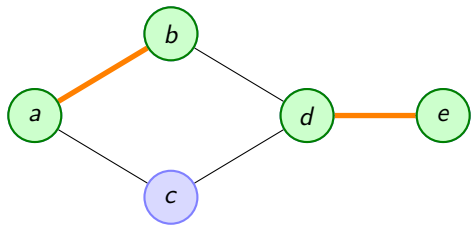
- Let M be the set of edges the algorithm picks. They are pairwise disjoint $\Rightarrow$ forming a matching.



Picked edges form a matching M

Why the Greedy Algorithm Is a 2-Approximation

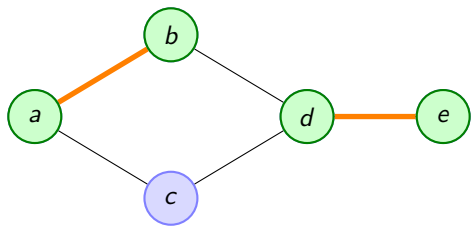
- Let M be the set of edges the algorithm picks. They are pairwise disjoint $\Rightarrow$ forming a matching.
- Any vertex cover must hit each edge of M. $\Rightarrow |C^*| \geq |M|$.



Picked edges form a matching M

Why the Greedy Algorithm Is a 2-Approximation

- Let M be the set of edges the algorithm picks. They are pairwise disjoint $\Rightarrow$ forming a matching.
- Any vertex cover must hit each edge of M. $\Rightarrow |C^*| \geq |M|$.
- The algorithm adds *both* endpoints of each edge in M. $\Rightarrow |C_{\text{ALG}}| = 2|M|$.

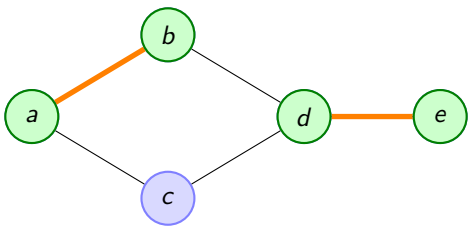


Picked edges form a matching M

Why the Greedy Algorithm Is a 2-Approximation

- Let M be the set of edges the algorithm picks. They are pairwise disjoint $\Rightarrow$ forming a matching.
- Any vertex cover must hit each edge of M. $\Rightarrow |C^*| \geq |M|$.
- The algorithm adds *both* endpoints of each edge in M. $\Rightarrow |C_{\text{ALG}}| = 2|M|$.
- Therefore

$$|C_{\text{ALG}}| = 2|M| \leq 2|C^*|.$$



Picked edges form a matching M

(In)Approximability of TSP

Approximating TSP

APPROX-TSP Given a complete graph G and a constant $\alpha > 1$, output a tour in G with cost at most $\alpha \cdot \mathrm{OPT}$.

Intuitively, this requirement is **weaker** than exact TSP.

- Any algorithm solving exact TSP automatically solves APPROX-TSP.

Approximating TSP

APPROX-TSP Given a complete graph G and a constant $\alpha > 1$, output a tour in G with cost at most $\alpha \cdot \mathrm{OPT}$.

Intuitively, this requirement is **weaker** than exact TSP.

- Any algorithm solving exact TSP automatically solves APPROX-TSP.

However, APPROX-TSP and TSP have essentially the same difficulty.

Inapproximability of TSP

Theorem

TSP admits *no* polynomial-time α-approximation algorithm for any constant $\alpha \geq 1$, unless P $=$ NP.

Inapproximability of TSP

Theorem

TSP admits *no* polynomial-time α-approximation algorithm for any constant $\alpha \geq 1$, unless $\mathsf{P} = \mathsf{NP}$.

How do we prove this? We return to our standard reduction recipe:

Reduce a known NP-complete problem (Hamiltonian Cycle)
to APPROX-TSP.

The Strategy: Distinguishing by Gap

Goal: Reduce Hamiltonian Cycle to α-Approx TSP.

To do this, we construct an instance where the optimal cost falls into two disjoint ranges separated by a factor of α.

The Distinguishing Condition

If an algorithm guarantees an α-approximation, it returns a tour of cost $C \leq \alpha \cdot \text{OPT}$. We need a gap such that:

- **Case Yes (Hamiltonian):** $\text{OPT} = n \implies C \leq \alpha n$.
- **Case No (Not Hamiltonian):** $\text{OPT} > \alpha n$.

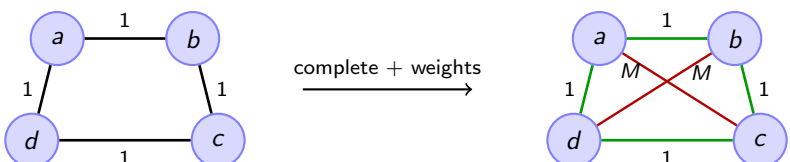
If we create this gap, checking if $C \leq \alpha n$ decides the Hamiltonian Cycle problem.

Creating the Gap

Construction: Build complete graph G' from G with weights:

$$w(u,v) = \begin{cases} 1 & \text{if } (u,v) \in E(G) \quad \text{(Original)} \\ M & \text{if } (u,v) \notin E(G) \quad \text{(Non-edge)} \end{cases}$$

where M is a large number (chosen later).



Creating the Gap

The Costs:

- **If G is Hamiltonian:** We use n edges of weight 1.
  $\Rightarrow \mathrm{OPT}(G') = n$.
- **If G is NOT Hamiltonian:** We must use at least one weight M.
  $\Rightarrow \mathrm{OPT}(G') \geq M + (n-1)$.

Forcing the Gap: Set $M = \alpha n$.

$$\text{No Hamiltonian Cycle} \implies \mathrm{OPT}(G') \geq \alpha n + (n-1) > \alpha n.$$

The Reduction Algorithm

The Procedure

Input: Graph G, Approximation factor α.

1. **Set the Penalty:** Let $n = |V|$ and choose a large weight $M = \alpha n + 1$.
2. **Construct G':** Create a complete graph where:
   - Existing edges in G get weight **1**.
   - Missing edges (non-edges) get weight M.
3. **Run the Solver:** Let C be the cost returned by $\textsc{Approx-TSP}(G')$.
4. **The Decision:**
   - If $C \leq \alpha n \longrightarrow$ Return **YES**.
   - If $C > \alpha n \longrightarrow$ Return **NO**.

Proof of Correctness

We analyze the output based on the structure of G:

Case 1: G has a Hamiltonian Cycle

- The optimal tour uses only original edges: $\mathrm{OPT}(G') = n$.
- The α-approx algorithm returns cost $C \leq \alpha \cdot \mathrm{OPT}$.
- Therefore, $C \leq \alpha n$.
- **Result:** Procedure correctly returns **YES**.

Proof of Correctness

We analyze the output based on the structure of G:

Case 1: G has a Hamiltonian Cycle

- The optimal tour uses only original edges: $\mathrm{OPT}(G') = n$.
- The α-approx algorithm returns cost $C \leq \alpha \cdot \mathrm{OPT}$.
- Therefore, $C \leq \alpha n$.
- **Result:** Procedure correctly returns **YES**.

Case 2: G has NO Hamiltonian Cycle

- Any tour must use at least one non-edge (weight M).
- Therefore, $\mathrm{OPT}(G') \geq M + (n-1) > \alpha n$.
- Since any tour cost $C \geq \mathrm{OPT}(G')$, we have $C > \alpha n$.
- **Result:** Procedure correctly returns **NO**.

Metric TSP and Its Approximation Algorithm

Turns out TSP is not so hopeless...

Metric TSP

Metric TSP An input instance to Metric TSP is a complete graph where edge weights are a metric.

- **Identity of indiscernibles:** $d(u,v) = 0 \Leftrightarrow u = v\,.$
- **Non-negativity:** $\forall u \neq v: \quad d(u,v) > 0.$
- **Symmetric:** $d(u,v) = d(v,u)\,.$
- **Triangle inequality:** $\forall u,\ v,\ w: \quad d(u,w) \leq d(u,v) + d(v,w)\,.$

Why do we care?

- Many natural settings (FedEx, road networks, Euclidean distances) satisfy the triangle inequality.
- Triangle inequality enables good approximation algorithms.

2-Approximation for Metric TSP: MST Doubling

Algorithm ("double-tree"):

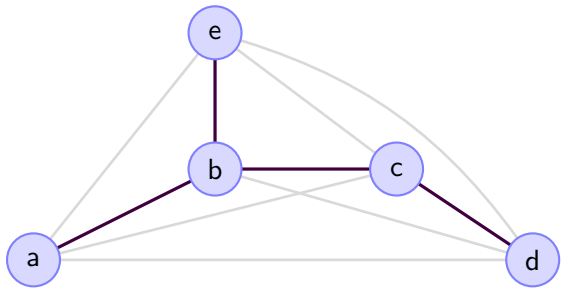
1. Compute a Minimum Spanning Tree (MST) T of the metric.
2. Double every edge of T to get an Eulerian multigraph.
3. Take an Euler tour and *shortcut* repeated vertices to obtain a TSP tour.

Visualizing the Double-Tree Algorithm

Step 1: Metric MST

We compute the MST of the metric graph (shown in **purple**).

Current Cost $\leq \mathrm{OPT}$.



Why?

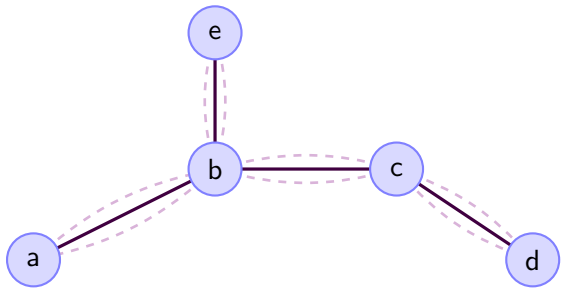
$$W(MST) \leq W(\text{Hamiltonian Path} \leq W(\text{Hamiltonian Cycle})$$

Visualizing the Double-Tree Algorithm

Step 2: Doubling

We double every edge in the MST.
This creates an **Eulerian Multigraph** (every node has even degree).

Current Cost $\leq 2 \cdot \mathrm{OPT}$.

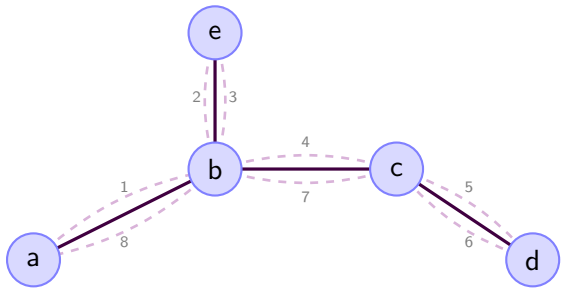


Visualizing the Double-Tree Algorithm

Step 3: Euler Tour

We traverse the doubled edges in a continuous loop (DFS order).

`a` $\rightarrow$ `b` $\rightarrow$ `e` $\rightarrow$ `b` $\rightarrow$ `c` $\rightarrow$ `d` $\rightarrow$ `c` $\rightarrow$ `b` $\rightarrow$ `a`

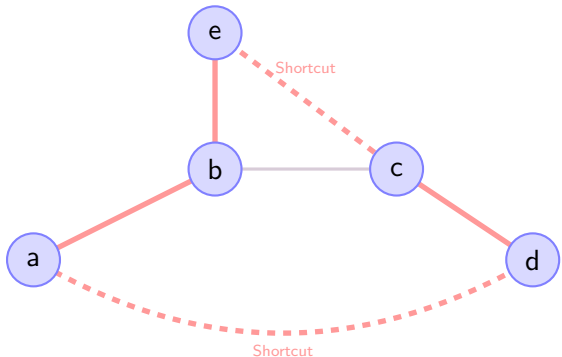


Visualizing the Double-Tree Algorithm

Step 4: Shortcutting

We delete repeated vertices from the Euler Tour. Thanks to the triangle inequality, taking a shortcut (red) is always cheaper.

Final Tour: `a` $\rightarrow$ `b` $\rightarrow$ `e` $\rightarrow$ `c` $\rightarrow$ `d` $\rightarrow$ `a`



References


Erickson, J. (2019).
Algorithms.
Self-published.