CS11921 - Fall 2023

Design & Analysis of Algorithms

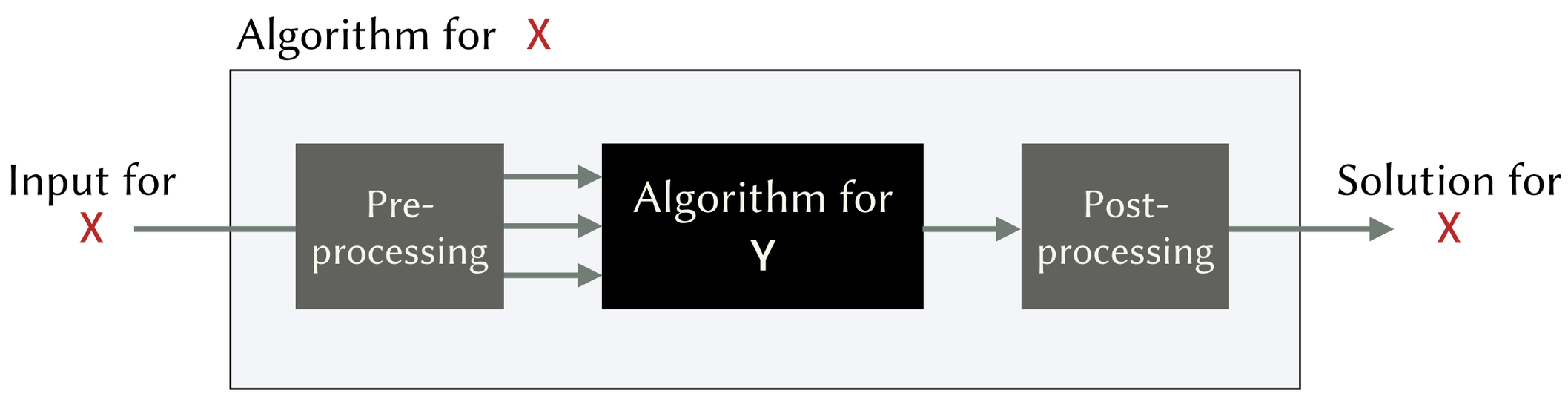
P, NP, NP-Complete and NP-Hard Problems

Ibrahim Albluwi

Reductions (Recap + Finetuning!)

A **reduction** from problem X to problem Y:
An algorithm for solving problem X that includes a solver of problem Y as a subroutine.

X **polytime-reduces** Y (denoted as $X \leqslant_p Y$) :
X can be solved using a solver for Y in addition to polytime amount of work.



Total cost for solving X = Cost of solving Y + Cost of reduction

Y might be called multiple time (typically 1 call)

Typically less than the cost of solving Y

Reductions (Recap + Finetuning!)

A **reduction** from problem X to problem Y:
An algorithm for solving problem X that includes a solver of problem Y as a subroutine.

X **polytime-reduces** Y (denoted as $X \leqslant_p Y$) :
X can be solved using a solver for Y in addition to polytime amount of work.

A **Turing Reduction** from X to Y:

- Allows calling Y's solver multiple times.
- Allows post-processing the output of Y's solver.

A **Karp Reduction** from X to Y:

- Allows calling Y's solver only once.
- Does not allow post-processing the output of Y 's solver.

Reductions (Recap + Finetuning!)

A **reduction** from problem X to problem Y:
An algorithm for solving problem X that includes a solver of problem Y as a subroutine.

X **polytime-reduces** Y (denoted as $X \leqslant_p Y$) :
X can be solved using a solver for Y in addition to polytime amount of work.

A **Turing Reduction** from X to Y:

- Allows calling Y's solver multiple times.
- Allows post-processing the output of Y's solver.

A **Karp Reduction** from X to Y:

- Allows calling Y's solver only once.
- Does not allow post-processing the output of Y's solver.

Example.

SELECT Given a list of elements, find the k^{th} largest element.

SORT Given a list of elements, order the elements in non-decreasing order.

SORT reduces to **SELECT**

Sort the elements by repeatedly using SELECT to find the next largest element.

Reductions (Recap + Finetuning!)

A **reduction** from problem X to problem Y:
An algorithm for solving problem X that includes a solver of problem Y as a subroutine.

X **polytime-reduces** Y (denoted as $X \leqslant_p Y$) :
X can be solved using a solver for Y in addition to polytime amount of work.

A **Turing Reduction** from X to Y:

- Allows calling Y's solver multiple times.
- Allows post-processing the output of Y's solver.

Example.

SELECT Given a list of elements, find the k^{th} largest element.

SORT Given a list of elements, order the elements in non-decreasing order.

SORT reduces to **SELECT**

Sort the elements by repeatedly using SELECT to find the next largest element.

A **Karp Reduction** from X to Y:

- Allows calling Y's solver only once.
- Does not allow post-processing the output of Y's solver.

Also called a **many-to-one** reduction:
Input to X is preprocessed such that every `YES` instance of X maps to a `YES` answer in Y and every `NO` instance of X maps to a `NO` answer in Y.

Example.

`TOTALITY` reduces to `EQUIVALENCE`

See the slides on Reductions discussed before.

Quiz # 1 (déjà vu!)

Suppose there is a proof that problem X is *difficult* to solve.

How can we prove that a problem Y is also *difficult* to solve?

A. Show that X reduces *easily* to Y.

B. Show that Y reduces *easily* to X.

Quiz # 1 (déjà vu!)

Suppose there is a proof that problem X is *difficult* to solve.

How can we prove that a problem Y is also *difficult* to solve?

A. Show that X reduces *easily* to Y.

B. Show that Y reduces *easily* to X.

Quiz # 1 (déjà vu!)

Suppose there is a proof that problem X is *difficult* to solve.

How can we prove that a problem Y is also *difficult* to solve?

A. Show that X reduces *easily* to Y.

B. Show that Y reduces *easily* to X.

Explanation.

We know that $X \notin \mathsf{EASY}$ and we want to use this information to show that $Y \notin \mathsf{EASY}$. I.e. we want to show that $X \notin \mathsf{EASY} \implies Y \notin \mathsf{EASY}$). We can achieve this by showing that the the following contrapositive statement holds:

$$Y \in \mathsf{EASY} \implies X \in \mathsf{EASY}$$

- If X reduces easily to Y and $Y \in \mathsf{EASY}$ then $X \in \mathsf{EASY}$.
- Since it is known that $X \notin \mathsf{EASY}$, then $Y \in \mathsf{EASY}$ **must be false!** (otherwise, there will be a contradiction)

NEVER FORGET

If A is hard to solve and

A easily reduces to B $(A \leqslant_p B)$,

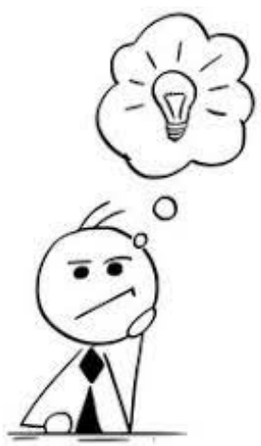
Then B is also hard to solve!

NEVER FORGET

If A is hard to solve and

A easily reduces to B ($A \leqslant_p B$),

Then B is also hard to solve!



What does it mean for a problem to be hard anyway?

A fine line **Between *Hard* and *Easy* Problems**

- Shortest Paths on unweighted graphs —— `O(E+V) using BFS`
- Shortest Paths on weighted DAGs —— `O(E+V) using Topological sort`
- Longest Paths on weighted DAGs —— `O(E+V) using Topological Sort`
- Shortest Paths on weighted graphs (no negative weights) `O(ELogV) using Dijkstra's`

A fine line **Between *Hard* and *Easy* Problems**

- Shortest Paths on unweighted graphs — `O(E+V) using BFS`
- Shortest Paths on weighted DAGs — `O(E+V) using Topological sort`
- Longest Paths on weighted DAGs — `O(E+V) using Topological Sort`
- Shortest Paths on weighted graphs (no negative weights) `O(ELogV) using Dijkstra's`
- Longest Paths on weighted graphs `NO KNOWN POLYNOMIAL TIME ALGORITHM!`

A fine line **Between *Hard* and *Easy* Problems**


Shortest Paths on unweighted graphs — `O(E+V) using BFS`


Shortest Paths on weighted DAGs — `O(E+V) using Topological sort`


Longest Paths on weighted DAGs — `O(E+V) using Topological Sort`


Shortest Paths on weighted graphs (no negative weights) `O(ELogV) using Dijkstra's`

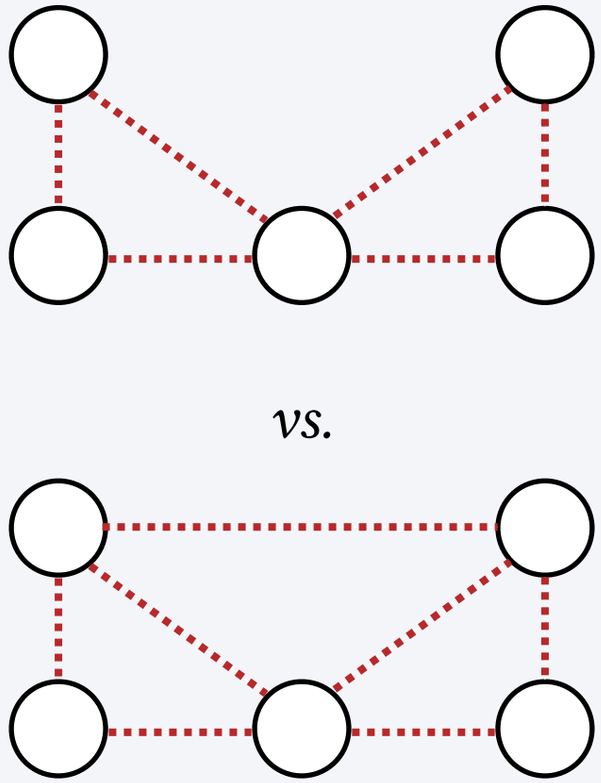

Longest Paths on weighted graphs `NO KNOWN POLYNOMIAL TIME ALGORITHM!`


Fractional Knapsack Problem — `has an efficient greedy algorithm`


0-1 Knapsack Problem — `NO KNOWN POLYNOMIAL TIME ALGORITHM!`

A fine line Between *Hard* and *Easy* Problems

Does a graph G contain an Eulerian Cycle?
(a cycle that visits all the *edges* in G exactly once)

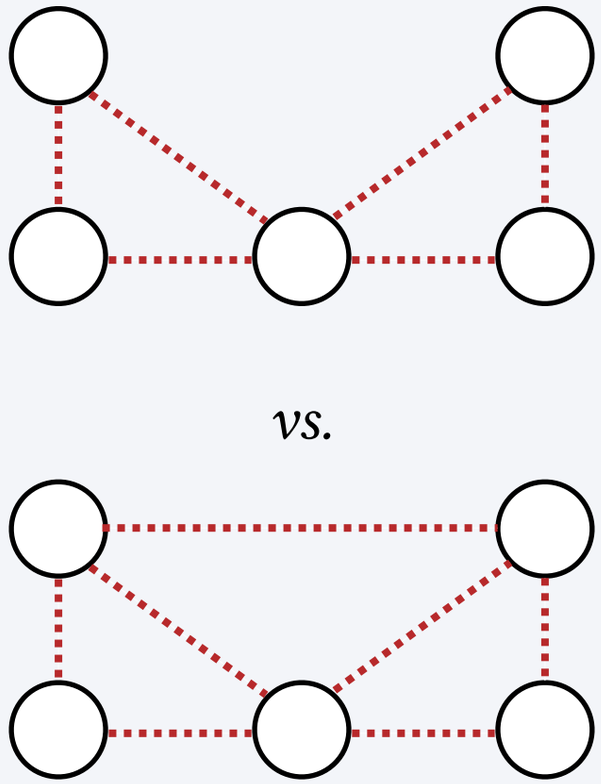


vs.

A fine line **Between *Hard* and *Easy* Problems**

Does a graph G contain an Eulerian Cycle?
(a cycle that visits all the *edges* in G exactly once)

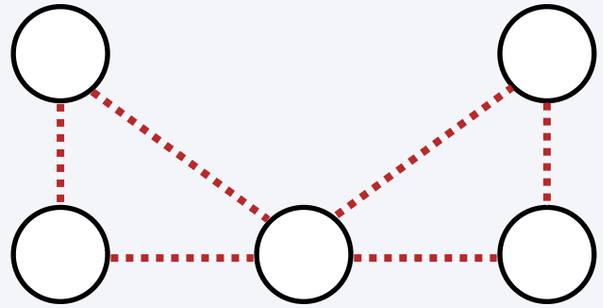
Direct solution: True if and only if each vertex has an even degree!



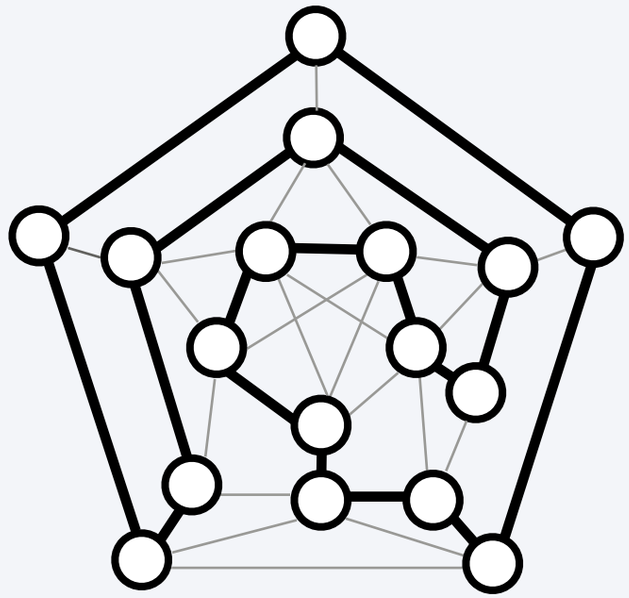
A fine line Between *Hard* and *Easy* Problems

Does a graph G contain an Eulerian Cycle?
(a cycle that visits all the *edges* in G exactly once)

Direct solution: True if and only if each vertex has an even degree!



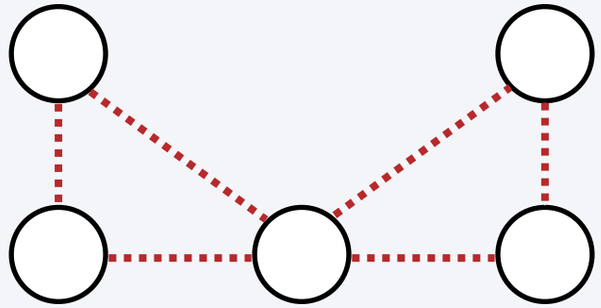
Does a graph G contain a Hamiltonian Cycle?
(a cycle that visits all the *vertices* in G exactly once)



A fine line Between *Hard* and *Easy* Problems

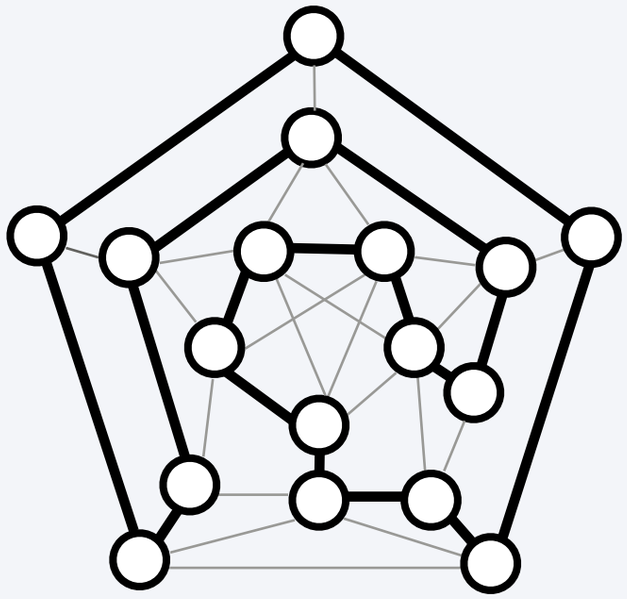
Does a graph G contain an Eulerian Cycle?
(a cycle that visits all the *edges* in G exactly once)

Direct solution: True if and only if each vertex has an even degree!



Does a graph G contain a Hamiltonian Cycle?
(a cycle that visits all the *vertices* in G exactly once)

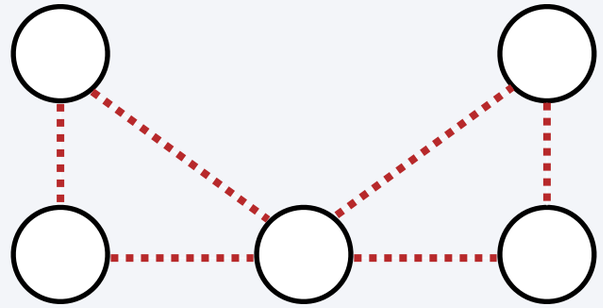
```
NO KNOWN POLYNOMIAL TIME ALGORITHM!
```



A fine line Between *Hard* and *Easy* Problems

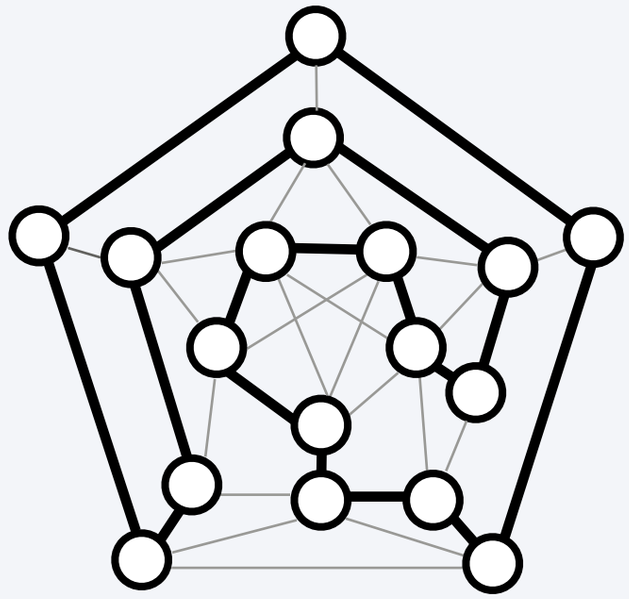
Does a graph G contain an Eulerian Cycle?
(a cycle that visits all the *edges* in G exactly once)

Direct solution: True if and only if each vertex has an even degree!



Does a graph G contain a Hamiltonian Cycle?
(a cycle that visits all the *vertices* in G exactly once)

NO KNOWN POLYNOMIAL TIME ALGORITHM!



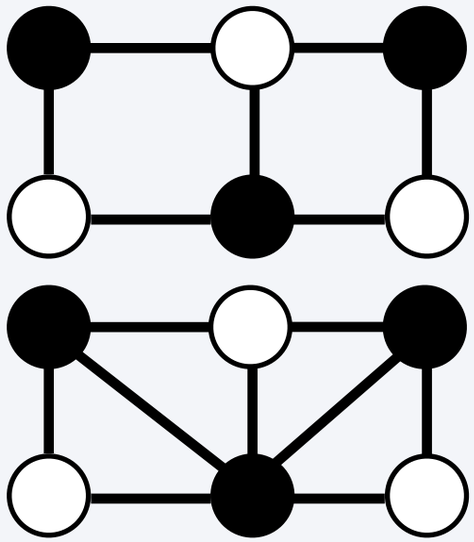
Traveling Salesman Problem (TSP)
Given a complete weighted graph, what is the *shortest Hamiltonian Cycle*?

NO KNOWN POLYNOMIAL TIME ALGORITHM!

A fine line **Between *Hard* and *Easy* Problems**

Is a graph 2-Colorable?
(can the vertices be colored using 2 colors, such that no two adjacent vertices have the same color?)

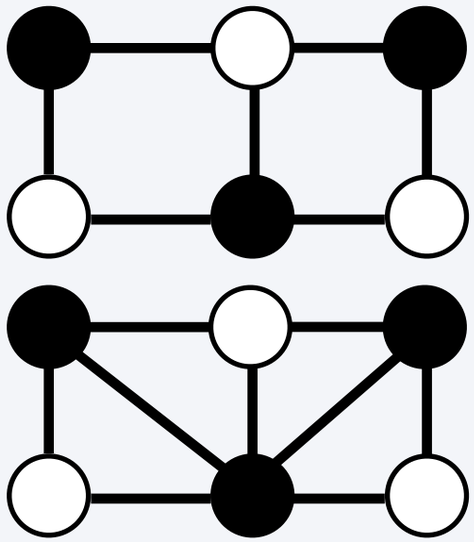
Direct solution: True if there is no cycle of odd length (can be checked using BFT)



A fine line **Between *Hard* and *Easy* Problems**

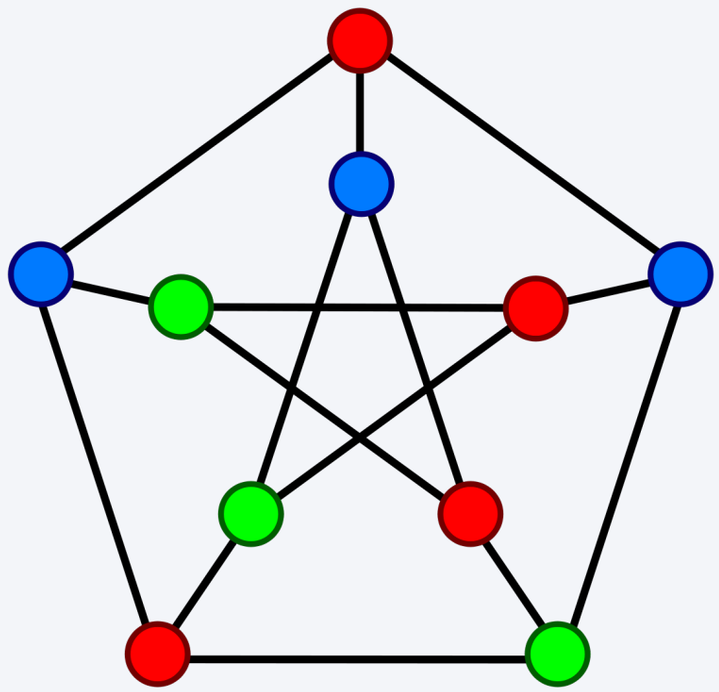
Is a graph 2-Colorable?
(can the vertices be colored using 2 colors, such that no two adjacent vertices have the same color?)

Direct solution: True if there is no cycle of odd length (can be checked using BFT)



Is a graph k-Colorable?
(can the vertices be colored using k colors or less, such that no two adjacent vertices have the same color?)

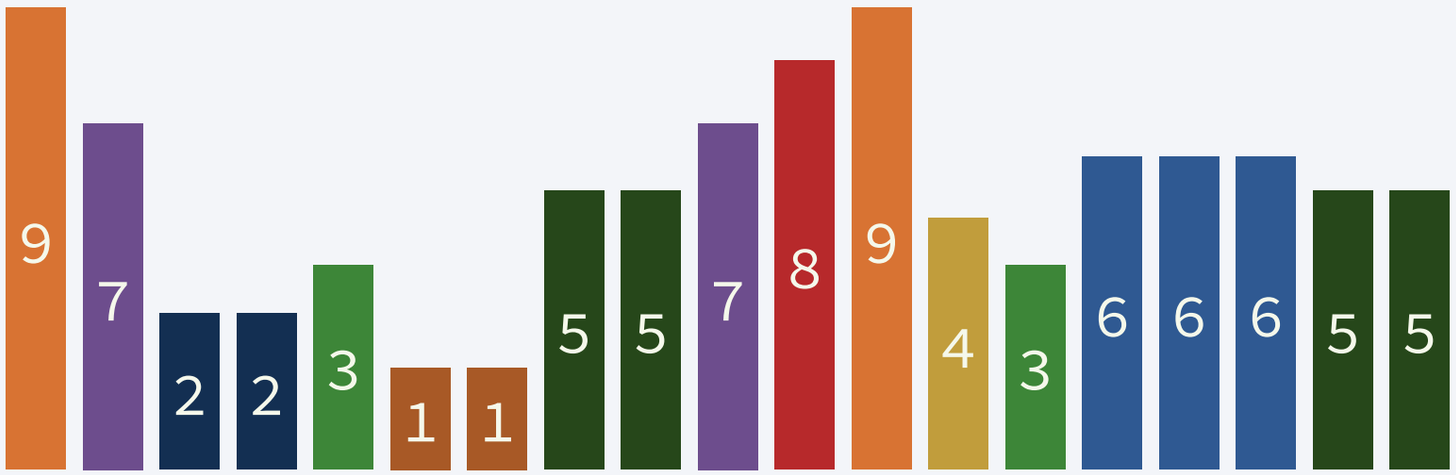
```
NO KNOWN POLYNOMIAL TIME ALGORITHM!
```

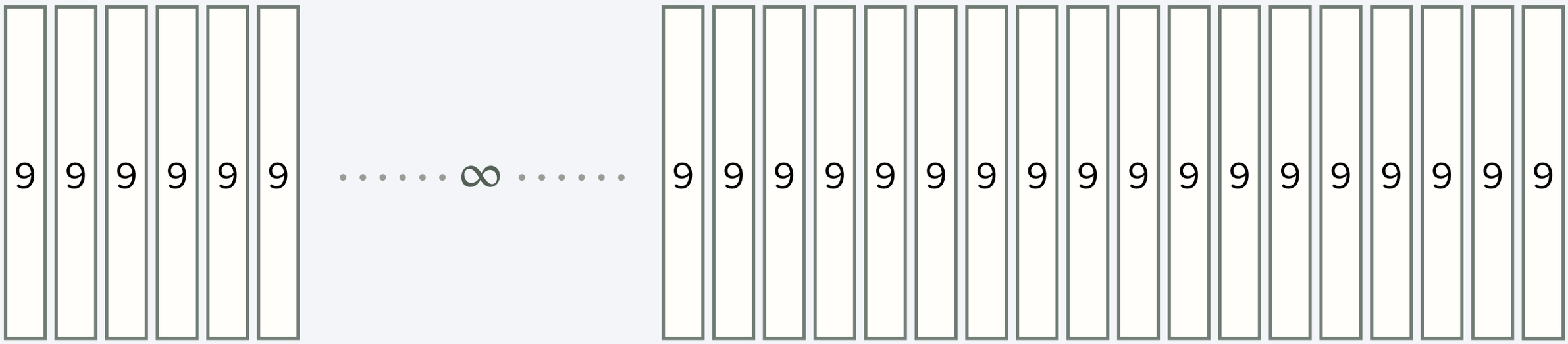


More Hard Problems

Bin Packing

Given an unlimited number of bins (each with capacity C), and n objects with sizes $s_1, \ldots, s_n$ where $0 < s_i \leq C$, find the *minimum* number of bins needed to pack all objects.



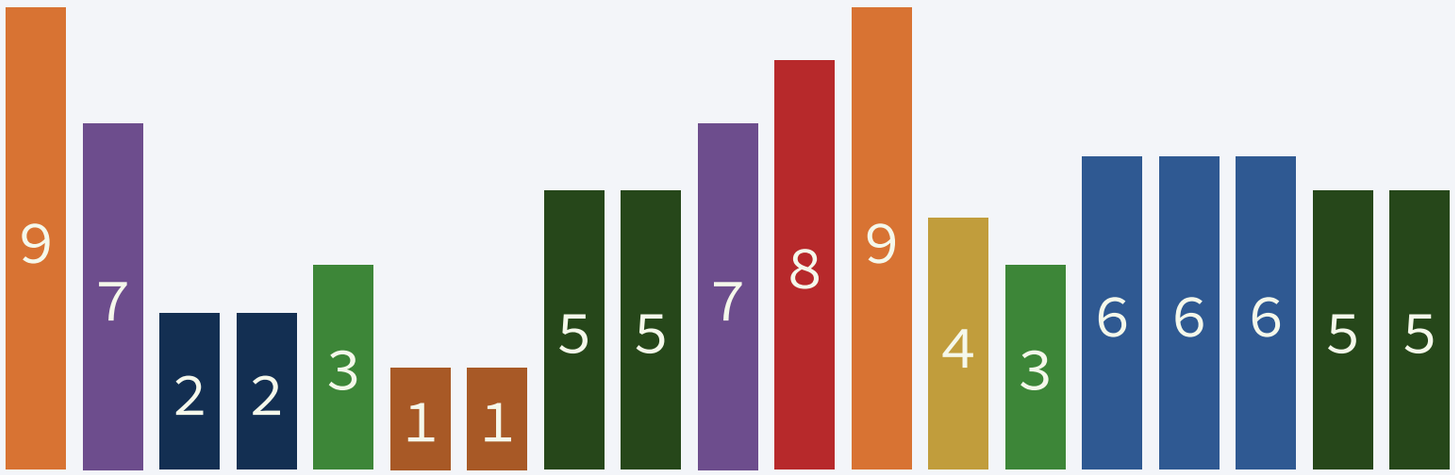


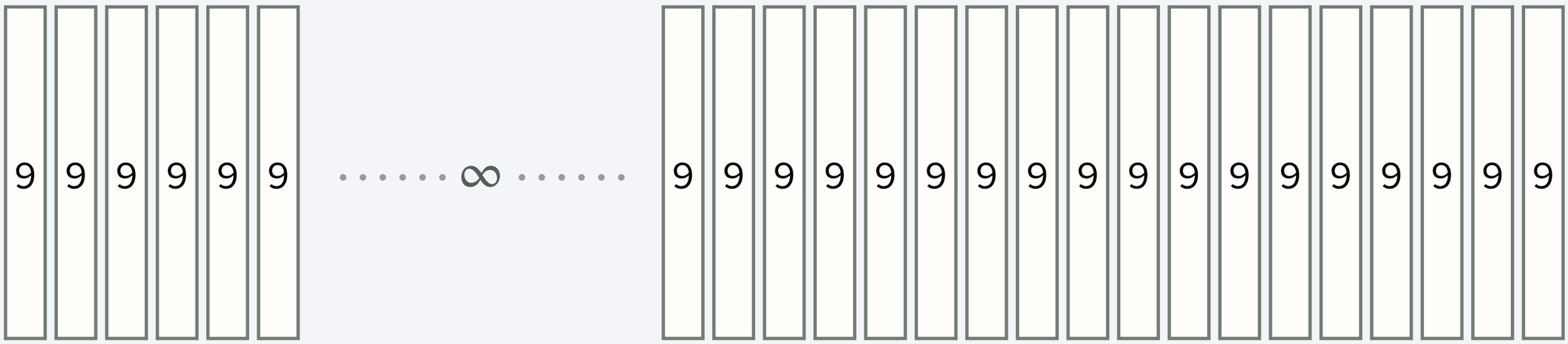
More Hard Problems

Bin Packing

Given an unlimited number of bins (each with capacity C), and n objects with sizes $s_1, \ldots, s_n$ where $0 < s_i \leq C$, find the *minimum* number of bins needed to pack all objects.

NO KNOWN POLYNOMIAL TIME ALGORITHM!





More Hard Problems

Subset Sum

Given a multiset S of integers and an integer k, find a *minimum* subset of S whose elements sum up to exactly k.

Example. S = `{1, 1, 1, 4, 4, 5, 6}`, k = `8`

Possible Subsets: `{1, 1, 1, 5}`, `{1, 1, 6}`, `{4, 4}` ← min subset

More Hard Problems

Subset Sum

Given a multiset S of integers and an integer k, find a *minimum* subset of S whose elements sum up to exactly k.

Example. S = {1, 1, 1, 4, 4, 5, 6}, k = 8

Possible Subsets: {1, 1, 1, 5}, {1, 1, 6}, {4, 4} ← min subset

NO KNOWN POLYNOMIAL TIME ALGORITHM!

More Hard Problems

Subset Sum

Given a multiset S of integers and an integer k, find a *minimum* subset of S whose elements sum up to exactly k.

Example. S = {1, 1, 1, 4, 4, 5, 6}, k = 8

Possible Subsets: {1, 1, 1, 5}, {1, 1, 6}, {4, 4} ← min subset

NO KNOWN POLYNOMIAL TIME ALGORITHM!

Subset Partition

Given a multiset S of integers, can S be partitioned into 2 subsets of the same sum?

Example. S = {1, 2, 3, 4}

YES: {1, 4} and {2, 3}

S = {1, 2, 3, 4, 5}

No

More Hard Problems

Subset Sum

Given a multiset S of integers and an integer k, find a *minimum* subset of S whose elements sum up to exactly k.

Example. S = {1, 1, 1, 4, 4, 5, 6}, k = 8

Possible Subsets: {1, 1, 1, 5}, {1, 1, 6}, {4, 4} ← min subset

NO KNOWN POLYNOMIAL TIME ALGORITHM!

Subset Partition

Given a multiset S of integers, can S be partitioned into 2 subsets of the same sum?

Example. S = {1, 2, 3, 4}

YES: {1, 4} and {2, 3}

S = {1, 2, 3, 4, 5}

No

NO KNOWN POLYNOMIAL TIME ALGORITHM!

A Hard Problem?

What is common between finding longest paths in cyclic graphs, 0-1 Knapsack, Subset Sum, Subset Partition, Bin-Packing, TSP and Checking if a Hamiltonian cycle exists?
(+ many others ...)

A Hard Problem?

What is common between finding longest paths in cyclic graphs, 0-1 Knapsack, Subset Sum, Subset Partition, Bin-Packing, TSP and Checking if a Hamiltonian cycle exists?
(+ many others ...)

(1) No one until now found a polynomial time algorithm to solve any of them.

A Hard Problem?

What is common between finding longest paths in cyclic graphs, 0-1 Knapsack, Subset Sum, Subset Partition, Bin-Packing, TSP and Checking if a Hamiltonian cycle exists?
(+ many others ...)

(1) No one until now found a polynomial time algorithm to solve any of them.

(2) No one proved that no polynomial time algorithm can be found for any of them.

A Hard Problem?

What is common between finding longest paths in cyclic graphs, 0-1 Knapsack, Subset Sum, Subset Partition, Bin-Packing, TSP and Checking if a Hamiltonian cycle exists?
(+ many others ...)

(1) No one until now found a polynomial time algorithm to solve any of them.

(2) No one proved that no polynomial time algorithm can be found for any of them.

(3) Each of them poly-time reduces to all the other problems!
I.e. Finding a polynomial time solution to any of them means that all of them have polynomial time solutions!

A Hard Problem?

What is common between finding longest paths in cyclic graphs, 0-1 Knapsack, Subset Sum, Subset Partition, Bin-Packing, TSP and Checking if a Hamiltonian cycle exists?
(+ many others ...)

(1) No one until now found a polynomial time algorithm to solve any of them.

(2) No one proved that no polynomial time algorithm can be found for any of them.

(3) Each of them poly-time reduces to all the other problems!
I.e. Finding a polynomial time solution to any of them means that all of them have polynomial time solutions!

(4) You will get $1,000,000 from the Clay Mathematics Institute if you find a polynomial time solution for any of them or prove that any of them can't have a polynomial time solution!

Welcome to the

P *vs* NP

Problem

Definitions

Optimization problem:
Find the ***best*** solution among a set of feasible solutions.

Decision problem:
Requires a **yes/no** answer.

Definitions

Optimization problem:
Find the ***best*** solution among a set of feasible solutions.

Decision problem:
Requires a **yes/no** answer.

Examples

Traveling Salesman Problem

Optimization problem:
Given a complete weighted graph G, find a simple circuit C that visits each node in G exactly once such that the total cost of the edges in C is *minimum*.

Definitions

Optimization problem:
Find the ***best*** solution among a set of feasible solutions.

Decision problem:
Requires a **yes/no** answer.

Examples

Traveling Salesman Problem

Optimization problem:
Given a complete weighted graph G, find a simple circuit C that visits each node in G exactly once such that the total cost of the edges in C is *minimum*.

Decision problem:
Given a complete weighted graph G, does G contain a simple circuit C that visits each node exactly once such that the total cost of the edges in C is *less than or equal to some threshold T*?

Definitions

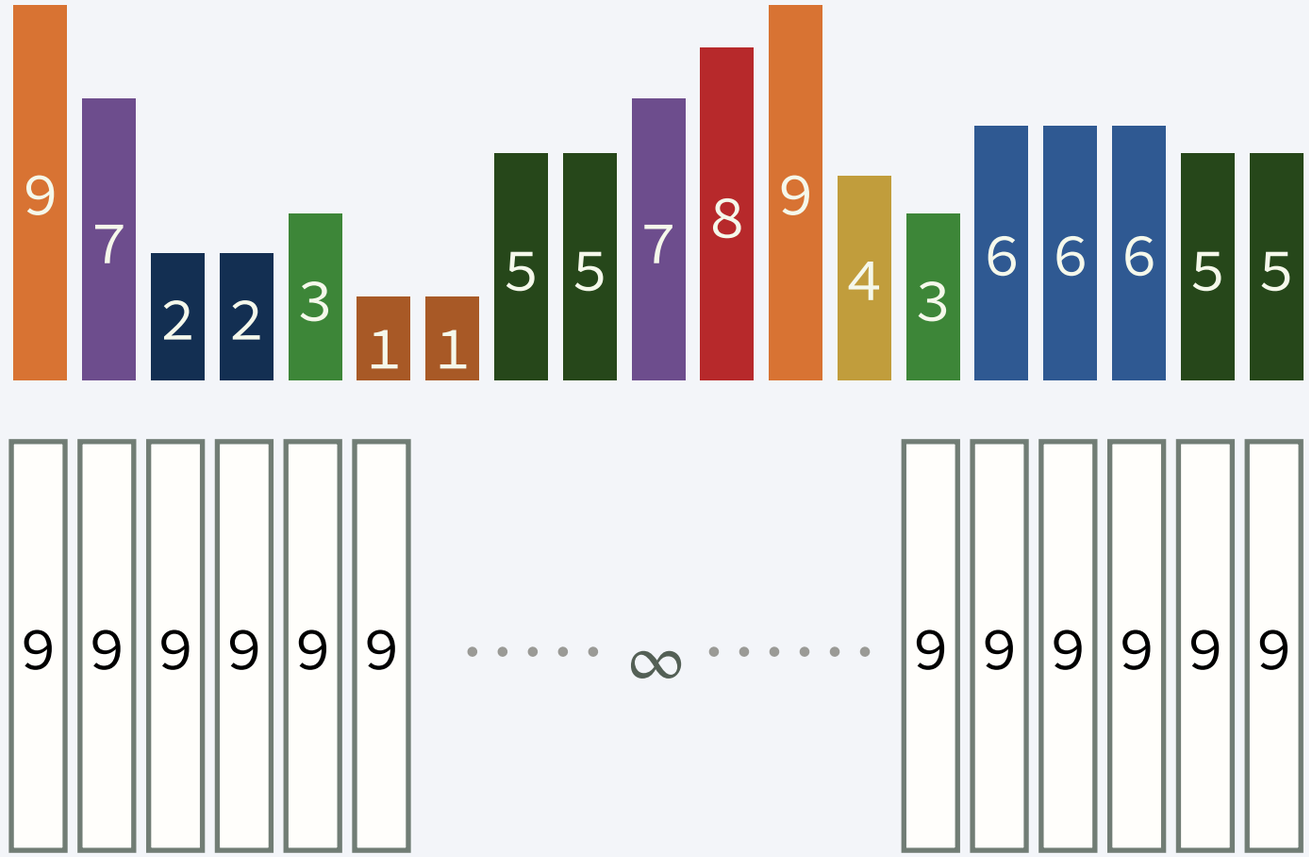
Optimization problem:
Find the ***best*** solution among a set of feasible solutions.

Decision problem:
Requires a **yes/no** answer.

Examples

Bin-Packing

Optimization problem:
Given an unlimited number of bins (each with capacity C), and n objects with sizes $s_1, \ldots, s_n$ where $0 < s_i \leq C$, find the *minimum* number of bins needed to pack all objects



Definitions

Optimization problem:
Find the ***best*** solution among a set of feasible solutions.

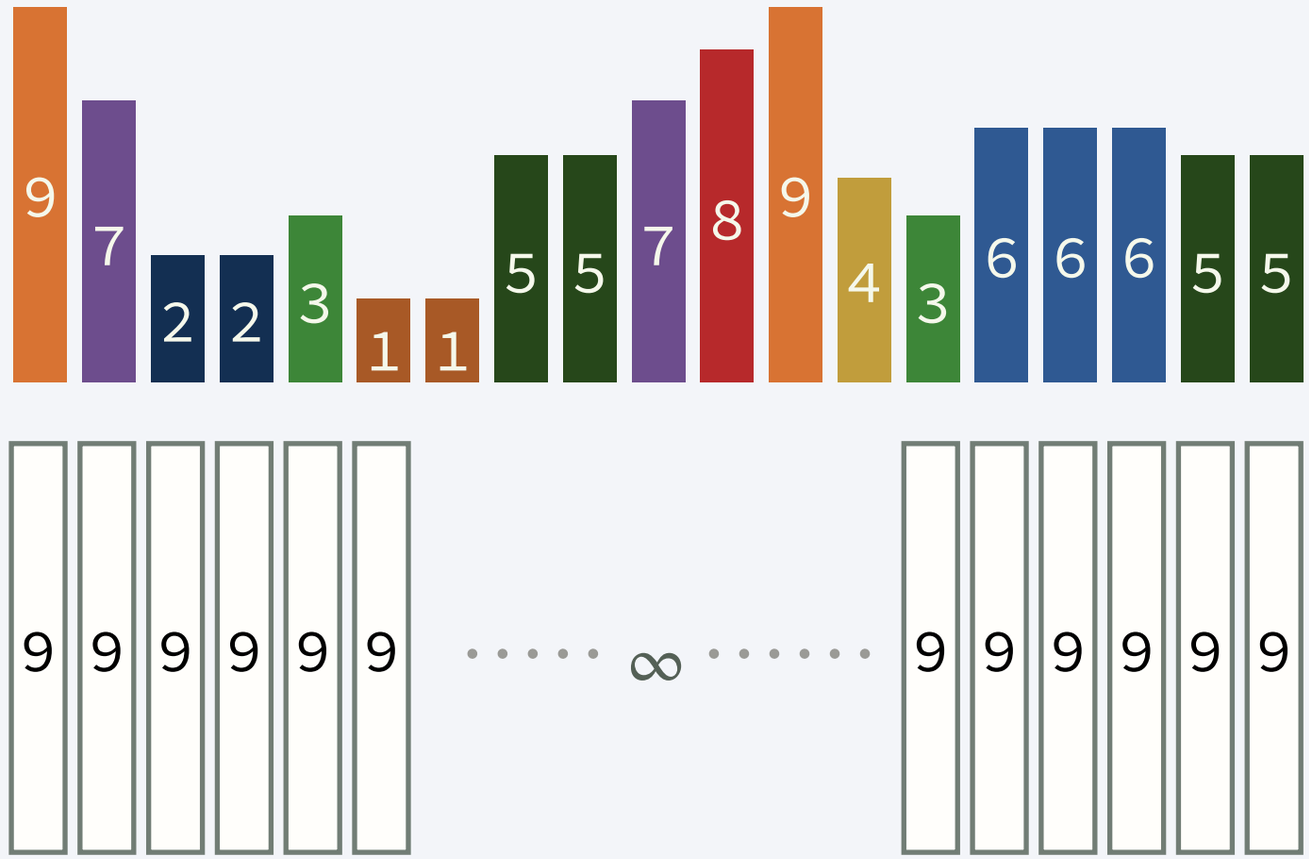
Decision problem:
Requires a **yes/no** answer.

Examples

Bin-Packing

Optimization problem:
Given an unlimited number of bins (each with capacity C), and n objects with sizes $s_1, \ldots, s_n$ where $0 < s_i \leq C$, find the *minimum* number of bins needed to pack all objects

Decision problem:
Can the objects fit in *less than k bins* ?



Definitions

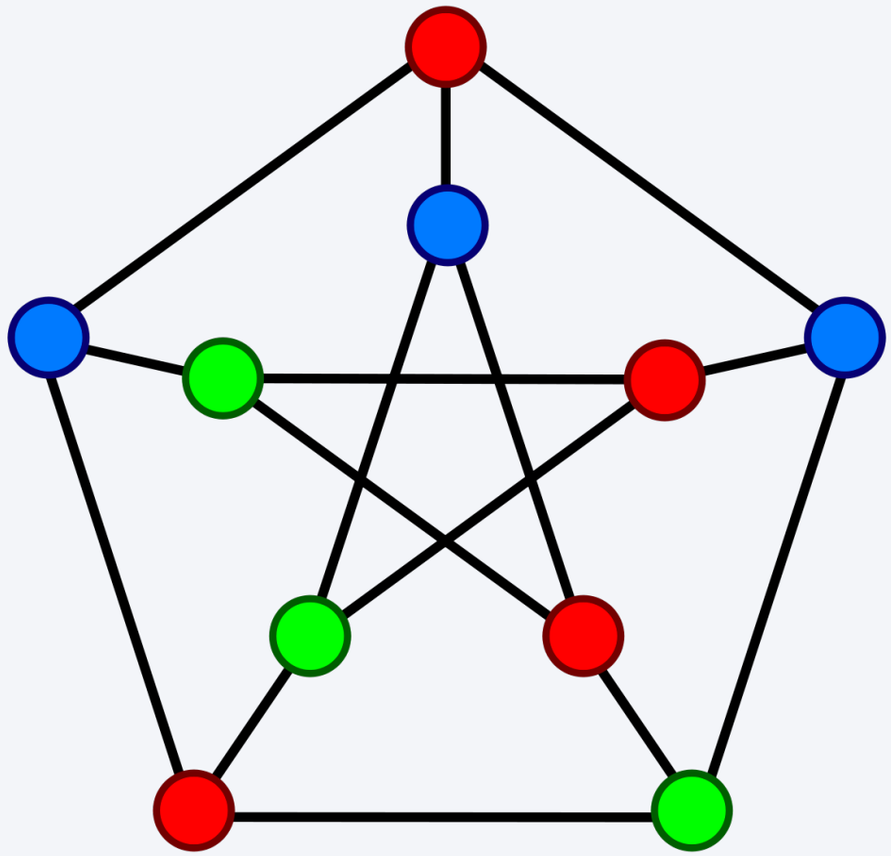
Optimization problem:
Find the ***best*** solution among a set of feasible solutions.

Decision problem:
Requires a **yes/no** answer.

Examples

Graph Coloring

Optimization problem:
Find the *minimum* number of colors such that adjacent vertices are not assigned the same color.



Definitions

Optimization problem:
Find the ***best*** solution among a set of feasible solutions.

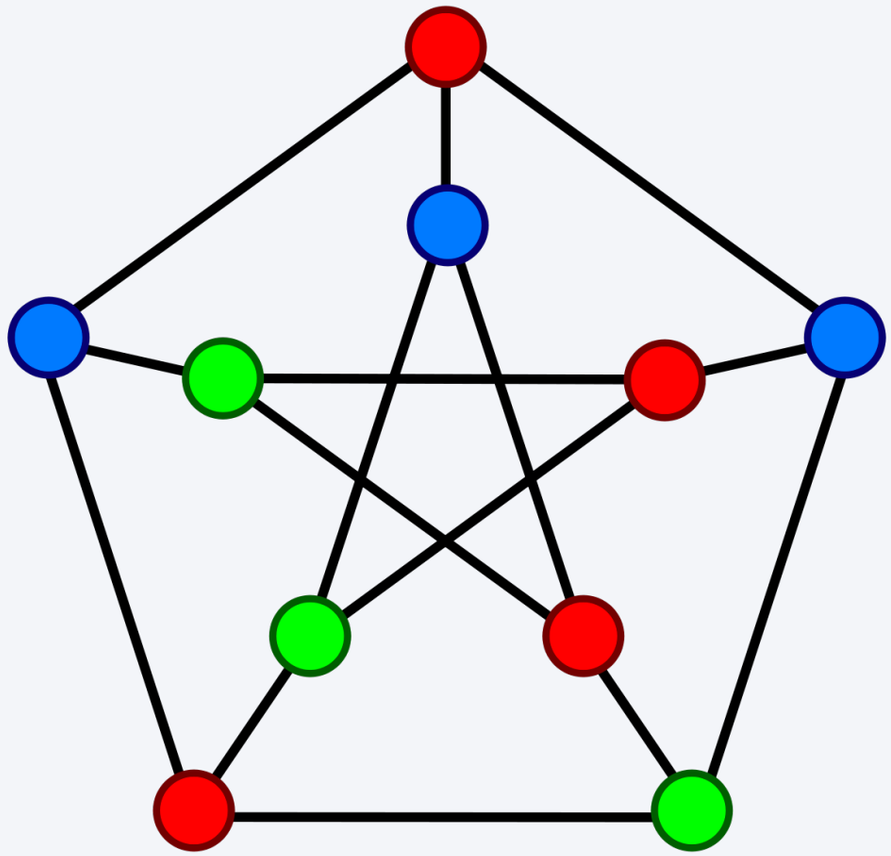
Decision problem:
Requires a **yes/no** answer.

Examples

Graph Coloring

Optimization problem:
Find the *minimum* number of colors such that adjacent vertices are not assigned the same color.

Decision problem:
Can the vertices be properly colored *in K or fewer* colors such that adjacent vertices are not assigned the same color?



Definitions

Optimization problem:
Find the ***best*** solution among a set of feasible solutions.

Decision problem:
Requires a **yes/no** answer.

Examples

Subset Sum

Optimization problem:
Given a multi-set S of integers and an integer k, find a *minimum* subset of S whose elements sum up to exactly k.

Example.
$S = \{1, 1, 1, 4, 4, 5, 6\}$, $k = 8$

Possible Subsets: $\{1, 1, 1, 5\}$
$\{1, 1, 6\}$
$\{4, 4\} \longleftarrow$ *minimum*

Definitions

Optimization problem:
Find the ***best*** solution among a set of feasible solutions.

Decision problem:
Requires a **yes/no** answer.

Examples

Subset Sum

Optimization problem:
Given a multi-set S of integers and an integer k, find a *minimum* subset of S whose elements sum up to exactly k.

Decision problem:
Does S contain a subset whose elements sum up to exactly k?

Example.

$S = \{1, 1, 1, 4, 4, 5, 6\}$, $k = 8$

Possible Subsets: $\{1, 1, 1, 5\}$

$\{1, 1, 6\}$

$\{4, 4\} \longleftarrow$ *minimum*

Definitions

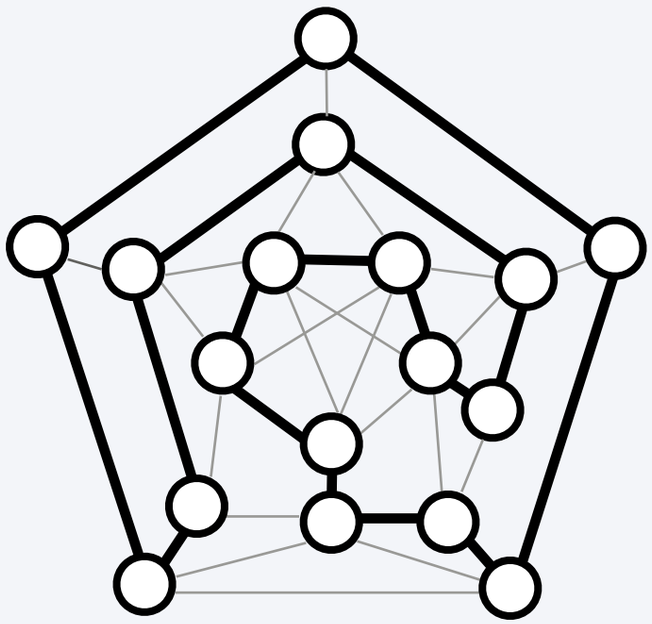
Optimization problem:
Find the ***best*** solution among a set of feasible solutions.

Decision problem:
Requires a **yes/no** answer.

Examples

Hamiltonian Cycle

Decision problem:
Is there a cycle that visits each vertex in the graph once?



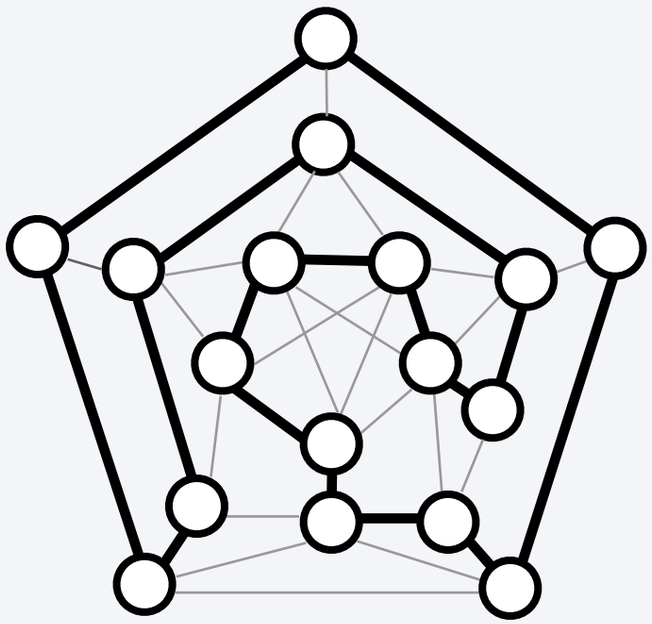
Definitions

Optimization problem:
Find the ***best*** solution among a set of feasible solutions.

Decision problem:
Requires a **yes/no** answer.

Examples Hamiltonian Cycle

Decision problem:
Is there a cycle that visits each vertex in the graph once?



Examples Subset Partition

Decision problem:
Given a set S of integers, Can we partition S into two subsets of exactly the same size?

Example. S = {1, 2, 3, 4}
YES: {1, 4} and {2, 3}

S = {1, 2, 3, 4, 5}
No

Quiz # 2

Given a solver for the optimization version of TSP, how can we solve the decision version?

Quiz # 2

Given a solver for the optimization version of TSP, how can we solve the decision version?

Answer. If we know the length of the shortest tour L, we can very easily answer the question *Is there a tour of length less than T* as follows:

If $L \geq T$: There is no tour of length less than T.
If $L < T$: There is a tour of length less than T.

Quiz # 2

Given a solver for the optimization version of TSP, how can we solve the decision version?

Answer. If we know the length of the shortest tour L, we can very easily answer the question *Is there a tour of length less than* T as follows:

If $L \geq T$: There is no tour of length less than T.
If $L < T$: There is a tour of length less than T.

$$\mathsf{TSP}_{dec} \leq_p \mathsf{TSP}_{opt}$$

Quiz # 3

If the decision version of a problem is hard, does this imply that the optimization version is also hard?

Quiz # 3

If the decision version of a problem is hard, does this imply that the optimization version is also hard?

Answer. Yes.

The decision version is no harder (as hard or easier) than the optimization version.

$$\mathsf{TSP}_{dec} \leq_p \mathsf{TSP}_{opt}$$

To discuss and prove hardness,
we will consider only *decision problems*!

Definitions (Complexity Classes)

Class **P**.

A decision problem is in P if it is solvable in polynomial time
(i.e. in $O(n^c)$, where n is the input size and c is a constant)

Definitions (Complexity Classes)

Class **P**.

A decision problem is in P if it is solvable in polynomial time.

Examples

- Given a list of integers L and an integer K:
  - is K in L?
  - Is there an integer in L that is greater than K ?
  - Do any two numbers in L sum to K ?
- Given a permutation of elements P:
  - is P sorted in ascending order?
  - is P a palindrome?
- Given a graph G:
  - Is there a spanning tree whose sum of edge weights is less than T ?
  - Is there a path between v and w in a graph G less than T ?
  - Is there a cycle in the graph?
  - Is the graph connected?

etc.

Quiz # 4

Which of the following decision problems are *not* in **P** ?

A. Traveling Salesman Problem.

B. 0-1 Knapsack.

C. Bin-Packing.

D. All of the above.

I don't know.

Quiz # 4

Which of the following decision problems are *not* in **P** ?

A. Traveling Salesman Problem.

B. 0-1 Knapsack.

C. Bin-Packing.

D. All of the above.

We don't know.

A problem is in **P** if it has a polynomial time solution.

A problem is *not* in **P** if there is a proof that it does not have a polynomial time solution.

While we don't have polynomial time solutions for these problems, no one proved that these problems do not have polynomial time solutions!

Definitions (Complexity Classes)

Class **P**.

A decision problem is in P if it is solvable in polynomial time
(i.e. in $O(n^c)$, where n is the input size and c is a constant)

Class **NP**.

A decision problem is in NP if it is verifiable in polynomial time.
(Given an instance I for a problem P and a witness W for the solution, can we verify in polynomial time if W proves that the answer for I is yes?)

Definitions (Complexity Classes)

Class **P**.

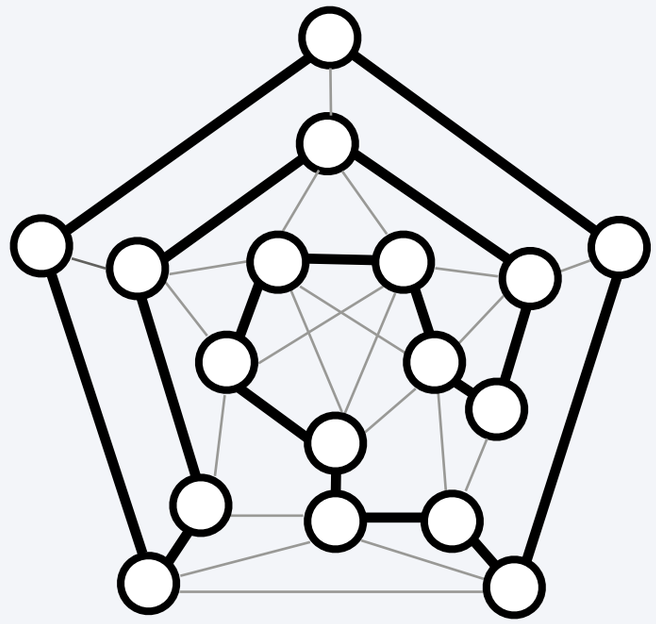
A decision problem is in P if it is solvable in polynomial time
(i.e. in $O(n^c)$, where n is the input size and c is a constant)

Class **NP**.

A decision problem is in NP if it is verifiable in polynomial time.
(Given an instance I for a problem P and a witness W for the solution, can we verify in polynomial time if W proves that the answer for I is yes?)

Example Is there A `HAMILTONIAN` Cycle?

Given a graph G, and a path C (a witness), can we verify in polynomial time if C is a hamiltonian cycle?



Yes!

1. Check that the first and last vertices are the same.
2. Check that no other vertices repeat.
3. Check that the path has exactly V edges and that they are all in G.

Definitions (Complexity Classes)

Class **P**.

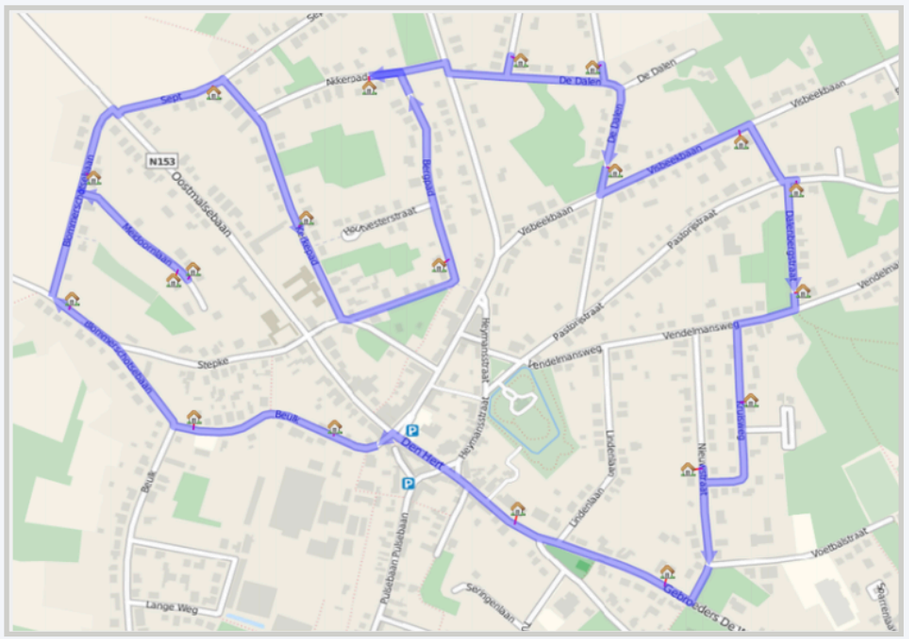
A decision problem is in P if it is solvable in polynomial time
(i.e. in $O(n^c)$, where n is the input size and c is a constant)

Class **NP**.

A decision problem is in NP if it is verifiable in polynomial time.
(Given an instance I for a problem P and a witness W for the solution, can we verify in polynomial time if W proves that the answer for I is yes?)

Example TSP is in NP

Given a graph G, a length L, and a path C (a witness), can we verify in polynomial time if C is a hamiltonian cycle of length less than L?



Definitions (Complexity Classes)

Class **P**.

A decision problem is in P if it is solvable in polynomial time
(i.e. in $O(n^c)$, where n is the input size and c is a constant)

Class **NP**.

A decision problem is in NP if it is verifiable in polynomial time.
(Given an instance I or a problem P and a witness W for the solution, can we verify in polynomial time if W proves that the answer for I is yes?)

Example TSP is in NP

Given a graph G, a length L, and a path C (a witness), can we verify in polynomial time if C is a hamiltonian cycle of length less than L?

Yes!

1. Check that C is a Hamiltonian cycle.
2. Check that the sum of the edge weights is less than L.

Definitions (Complexity Classes)

Class **P**.

A decision problem is in P if it is solvable in polynomial time
(i.e. in $O(n^c)$, where n is the input size and c is a constant)

Class **NP**.

A decision problem is in NP if it is verifiable in polynomial time.
(Given an instance I or a problem P and a witness W for the solution, can we verify in polynomial time if W proves that the answer for I is yes?)

Example SUBSET-SUM is in NP

Given a multi-set S, two integers K and L, and a subset H of S (a witness), can we verify in polynomial time if $|H| \leq K$ and that its elements sum to L?

Yes!

Definitions (Complexity Classes)

Class **P**.

A decision problem is in P if it is solvable in polynomial time
(i.e. in $O(n^c)$, where n is the input size and c is a constant)

Class **NP**.

A decision problem is in NP if it is verifiable in polynomial time.
(Given an instance I or a problem P and a witness W for the solution, can we verify in polynomial time if W proves that the answer for I is yes?)

Example SUBSET-SUM

Given a multi-set S, two integers K and L, and a subset H of S (a witness), can we verify in polynomial time if $|H| \leq K$ and that its elements sum to L?

Yes!

Example SUBSET-PARTITION

Given a multi-set S, two subsets H_1 *and* H_2 of S (a witness), can we verify in polynomial time if $|H_1| + |H_2| = |S|$ and that the sum of the elements in H_1 = the sum of the elements in H_2?

Yes!

Quiz # 5

Every problem that is in **P** is also in **NP**.

A. True.

B. False.

We don't know.

Quiz # 5

Every problem that is in **P** is also in **NP**.

A. True.

B. False.

We don't know.

If a problem is solvable in polynomial time, it is also verifiable in polynomial time.

We can always solve the problem to verify a given witness!

Quiz # 6

Every problem that is in **NP** is also in **P**.

A. True.

B. False.

We don't know.

Quiz # 6

Every problem that is in **NP** is also in **P**.

A. True.

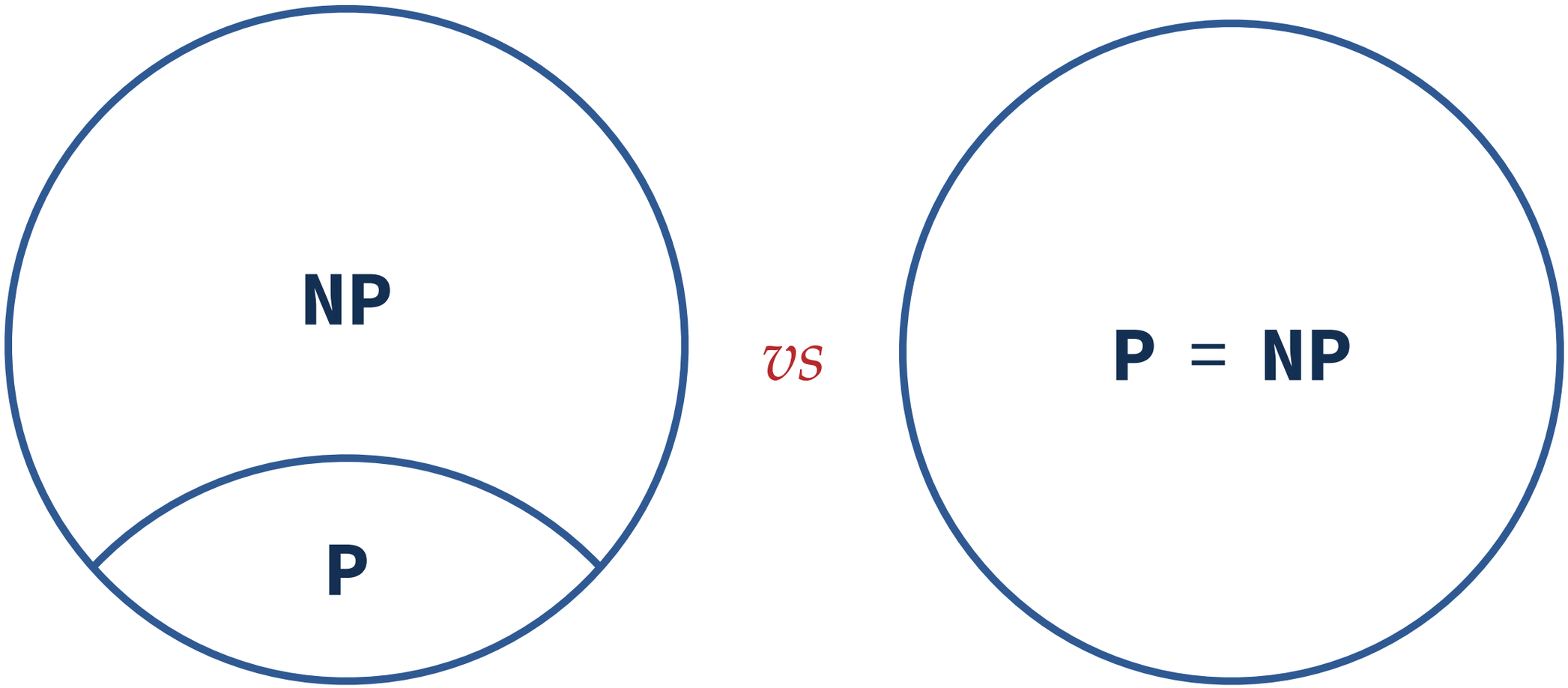
B. False.

We don't know.

Does easy verification imply that finding a solution is also easy?

- No one knows!
- No one yet found a problem that is in NP but is not in P !
- This is a $1,000,000 question! 💵💵💵

Two Possible World Views



No one knows which is true!

Quiz # 7

What are examples of problems that we do not know how to verify in polynomial time (hence, we are unable to place in **NP**)?

Quiz # 7

What are examples of problems that we do not know how to verify in polynomial time (hence, we are unable to place in **NP**)?

Example. Given a chessboard, is there a move that guarantees black to win?

Quiz # 7

What are examples of problems that we do not know how to verify in polynomial time (hence, we are unable to place in **NP**)?

Example. Given a chessboard, is there a move that guarantees black to win?

Given a chessboard and a certain move (a solution), we don't know how to verify in polynomial time if the move will guarantee black to win! (we can do exponential amount of work to check all possible black and white moves to see if black will win!)

Quiz # 7

What are examples of problems that we do not know how to verify in polynomial time (hence, we are unable to place in **NP**)?

Example. Given a chessboard, is there a move that guarantees black to win?

Given a chessboard and a certain move (a solution), we don't know how to verify in polynomial time if the move will guarantee black to win! (we can do exponential amount of work to check all possible black and white moves to see if black will win!)

Another Example. Does graph G have a *unique* Hamiltonian cycle?

What is in a name?

What does **NP** stand for?

A. **N**ot **P**olynomial.

B. **N**o **P**akeup Exam.

C. **N**o **P**roblem.

D. **N**one of the a**P**ove.

What is in a name?

What does **NP** stand for?

A. **N**ot **P**olynomial.

B. **N**o **P**akeup Exam.

C. **N**o **P**roblem.

D. **N**one of the a**P**ove.

NP stands for: **N**on-deterministically **P**olynomial.
I.e. Can be solved using a non-deterministic machine in polynomial time.

Assume that TM is a machine that can guess and verify an infinite number of solutions all at the same time (call TM a *non-deterministic* machine).

If a problem is verifiable in polynomial time, TM can solve the problem by guessing all the possible solutions and verifying them at once (in polynomial time!)

Definitions (Complexity Classes)

Class **P**.

A decision problem is in P if it is solvable in polynomial time
(i.e. in $O(n^c)$, where n is the input size and c is a constant)

Class **NP**.

A decision problem is in NP if it is verifiable in polynomial time.
(Given an instance I or a problem P and a witness W for the solution, can we verify in polynomial time if W proves that the answer for I is yes?)

Class **NP-Complete**.

A decision problem is NP-Complete if:

- It is in NP.
- All problems in NP reduce to it in polynomial time.

Definitions (Complexity Classes)

Class **P**.

A decision problem is in P if it is solvable in polynomial time
(i.e. in $O(n^c)$, where n is the input size and c is a constant)

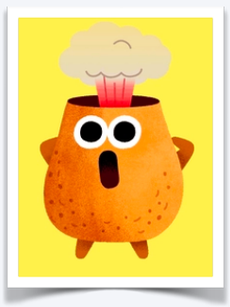
Class **NP**.

A decision problem is in NP if it is verifiable in polynomial time.
(Given an instance I or a problem P and a witness W for the solution, can we verify in polynomial time if W proves that the answer for I is yes?)

Class **NP-Complete**.

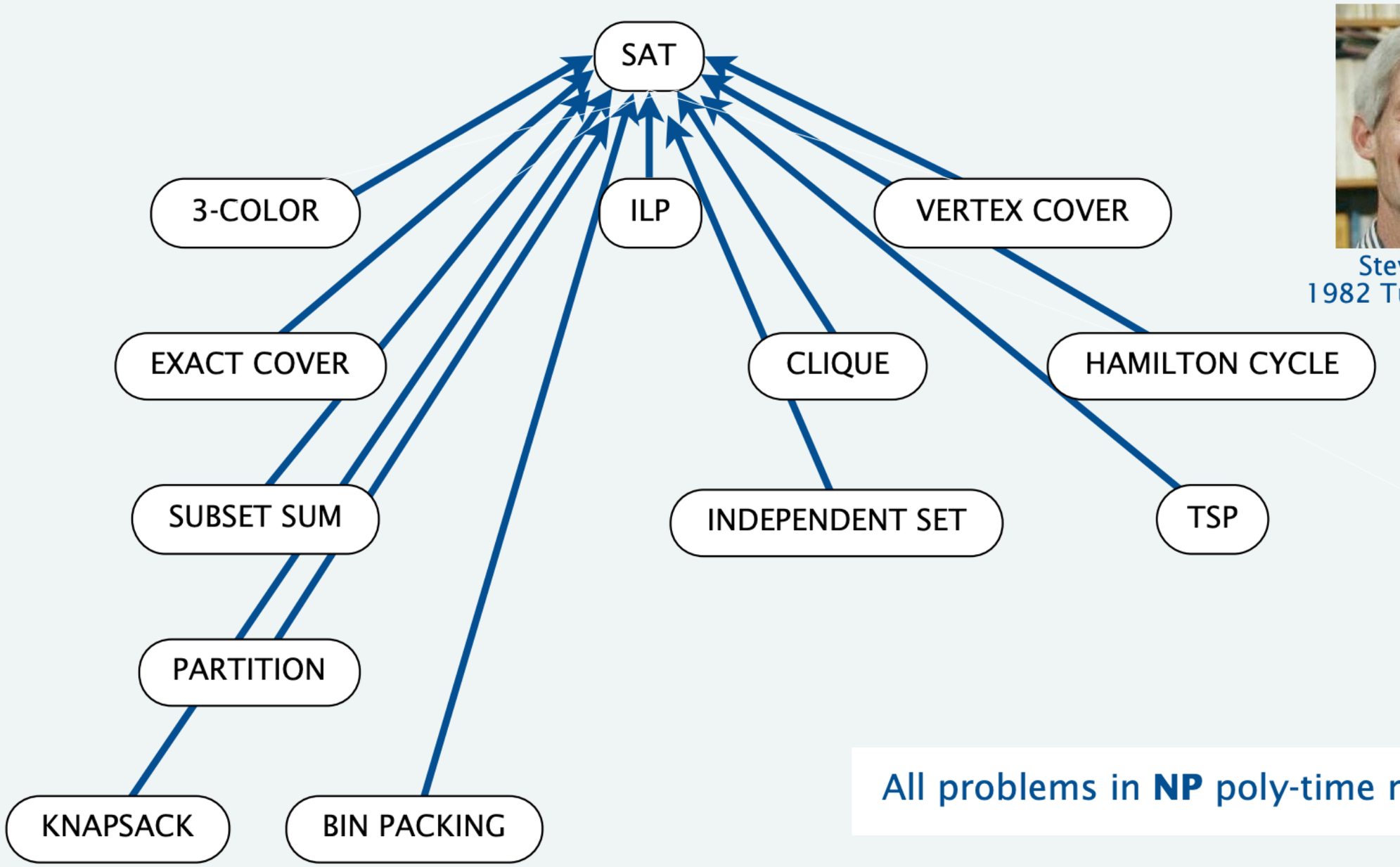
A decision problem is NP-Complete if:

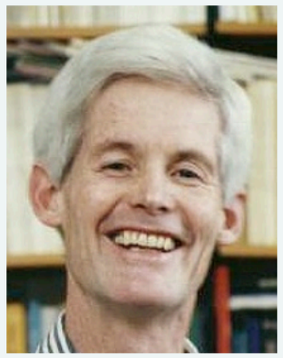
- It is in NP.
- All problems in NP reduce to it in polynomial time.



How do we show that *all* problems in **NP** reduce to a certain problem???

Cook-Levin Theorem (1971)





Steve Cook
1982 Turing Award

Leonid Levin

All problems in **NP** poly-time reduce to SAT.

What is **SAT**?

slide by Kevin Wayne

Boolean Satisfiability (SAT)

Literal. A Boolean variable or its negation. x_i or $\overline{x_i}$

Clause. A disjunction of literals. $C_j = x_1 \vee \overline{x_2} \vee x_3$

Conjunctive normal form (CNF). A propositional formula Φ that is a conjunction of clauses. $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$

SAT. Given a CNF formula Φ, does it have a satisfying truth assignment?

3-SAT. SAT where each clause contains exactly 3 literals (and each literal corresponds to a different variable).

adapted from a slide by Kevin Wayne

Boolean Satisfiability (SAT)

Literal. A Boolean variable or its negation. x_i or $\overline{x_i}$

Clause. A disjunction of literals. $C_j = x_1 \vee \overline{x_2} \vee x_3$

Conjunctive normal form (CNF). A propositional formula Φ that is a conjunction of clauses. $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$

SAT. Given a CNF formula Φ, does it have a satisfying truth assignment?

3-SAT. SAT where each clause contains exactly 3 literals (and each literal corresponds to a different variable).

Example What values for x_1, x_2, x_3 and x_4 satisfy the following formula?

$$\Phi = \left(\overline{x_1} \vee x_2 \vee x_3 \right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3 \right) \wedge \left(\overline{x_1} \vee x_2 \vee x_4 \right)$$

adapted from a slide by Kevin Wayne

Boolean Satisfiability (SAT)

Literal. A Boolean variable or its negation. x_i or $\overline{x_i}$

Clause. A disjunction of literals. $C_j = x_1 \vee \overline{x_2} \vee x_3$

Conjunctive normal form (CNF). A propositional formula Φ that is a conjunction of clauses. $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$

SAT. Given a CNF formula Φ, does it have a satisfying truth assignment?

3-SAT. SAT where each clause contains exactly 3 literals (and each literal corresponds to a different variable).

Example What values for x_1, x_2, x_3 and x_4 satisfy the following formula?

$$\Phi = \left(\overline{x_1} \vee x_2 \vee x_3 \right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3 \right) \wedge \left(\overline{x_1} \vee x_2 \vee x_4 \right)$$

Answer. x_1 = TRUE, x_2 = TRUE, x_3 = FALSE, x_4 = FALSE

adapted from a slide by Kevin Wayne

Boolean Satisfiability (SAT)

Key Facts.

- SAT is in NP.

adapted from a slide by Kevin Wayne

Boolean Satisfiability (SAT)

Key Facts.

- SAT is in NP.
  Given a formula and boolean values for the variables, it is easy to verify if these values satisfy the formula!

- It is not clear if SAT is also in P.
  - We can try all possible 2^N boolean assignments.
  - We don't know if a polynomial time solution exists.

adapted from a slide by Kevin Wayne

Boolean Satisfiability (SAT)

Key Facts.

- SAT is in NP.
  Given a formula and boolean values for the variables, it is easy to verify if these values satisfy the formula!

- It is not clear if SAT is also in P.
  - We can try all possible 2^N boolean assignments.
  - We don't know if a polynomial time solution exists.

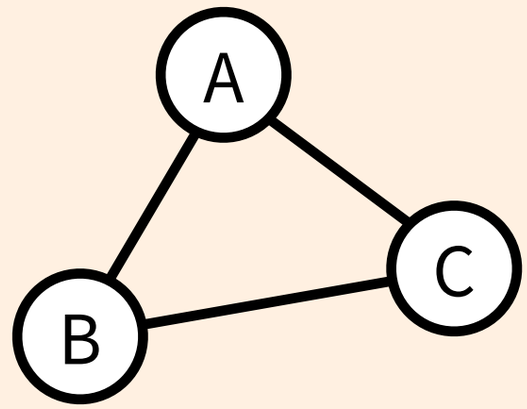
- All problems in NP reduce to SAT in polynomial time.
  - This is the Cook-Levin Theorem.
  - The details of the proof are beyond the scope of this course.
  - In a nutshell, Cook and Levin showed how any decision problem that is in NP can be converted (in polynomial time) to the problem of satisfying a boolean formula of a polynomial size).
    (i.e. a digital circuit can be designed for it that has a polynomial number of gates)

adapted from a slide by Kevin Wayne

Reduction Example

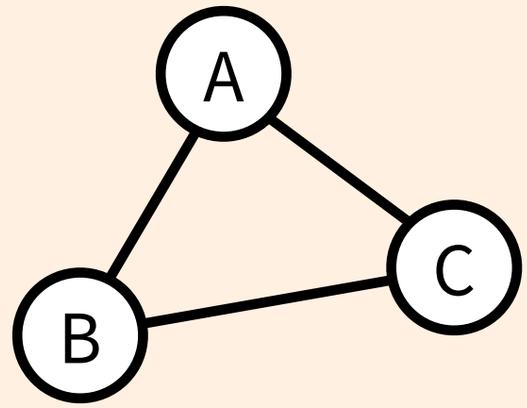
Graph Coloring reduces to SAT in polynomial time.

Assume that the problem is to check if the graph is 2-colorable.



Reduction Example

Graph Coloring reduces to SAT in polynomial time.

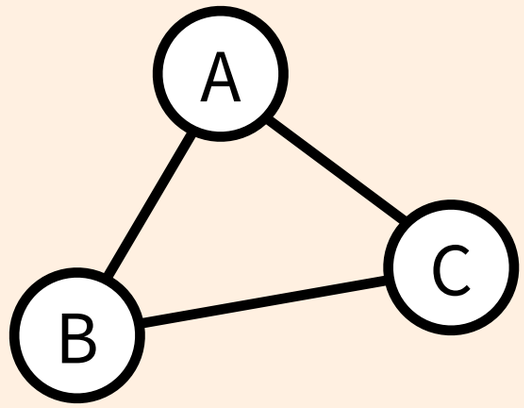


Assume that the problem is to check if the graph is 2-colorable.

1. Create the boolean variables:
   A_{red} , A_{blue} , B_{red} , B_{blue} , C_{red} , C_{blue}

Reduction Example

Graph Coloring reduces to SAT in polynomial time.



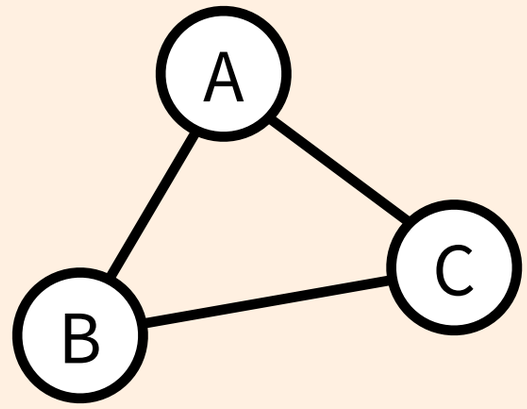
Assume that the problem is to check if the graph is 2-colorable.

1. Create the boolean variables:
   A_{red} , A_{blue} , B_{red} , B_{blue} , C_{red} , C_{blue}

2. Enforce that each vertex has one color:

Reduction Example

Graph Coloring reduces to SAT in polynomial time.



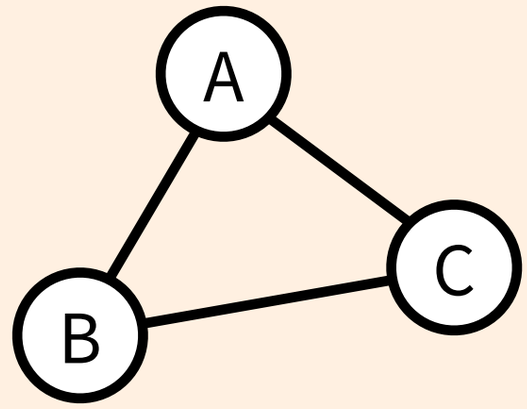
Assume that the problem is to check if the graph is 2-colorable.

1. Create the boolean variables:
   A_{red} , A_{blue} , B_{red} , B_{blue} , C_{red} , C_{blue}

2. Enforce that each vertex has one color:
   $(A_{red} \vee A_{blue}) \wedge \neg(A_{red} \wedge A_{blue}) = \text{TRUE}$
   $(B_{red} \vee B_{blue}) \wedge \neg(B_{red} \wedge B_{blue}) = \text{TRUE}$
   $(C_{red} \vee C_{blue}) \wedge \neg(C_{red} \wedge C_{blue}) = \text{TRUE}$

Reduction Example

Graph Coloring reduces to SAT in polynomial time.



Assume that the problem is to check if the graph is 2-colorable.

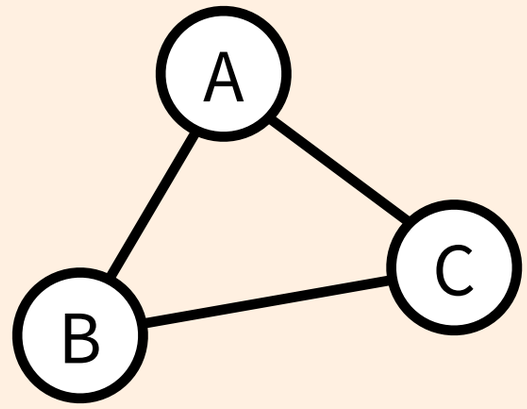
1. Create the boolean variables:
   A_{red} , A_{blue} , B_{red} , B_{blue} , C_{red} , C_{blue}

2. Enforce that each vertex has one color:
   $(A_{red} \vee A_{blue}) \wedge \neg(A_{red} \wedge A_{blue}) = \text{TRUE}$
   $(B_{red} \vee B_{blue}) \wedge \neg(B_{red} \wedge B_{blue}) = \text{TRUE}$
   $(C_{red} \vee C_{blue}) \wedge \neg(C_{red} \wedge C_{blue}) = \text{TRUE}$

3. Enforce that no adjacent vertices have the same color:

Reduction Example

Graph Coloring reduces to SAT in polynomial time.



Assume that the problem is to check if the graph is 2-colorable.

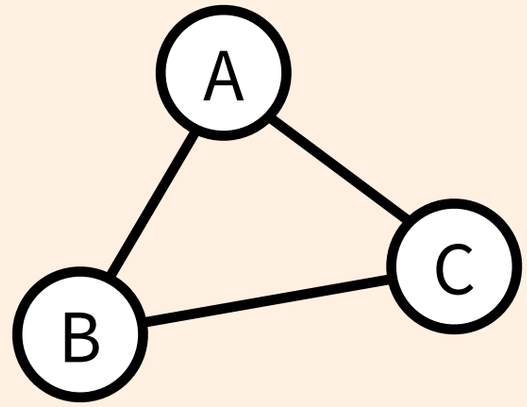
1. Create the boolean variables:
   A_{red} , A_{blue} , B_{red} , B_{blue} , C_{red} , C_{blue}

2. Enforce that each vertex has one color:
   $(A_{red} \vee A_{blue}) \wedge \neg(A_{red} \wedge A_{blue}) = \text{TRUE}$
   $(B_{red} \vee B_{blue}) \wedge \neg(B_{red} \wedge B_{blue}) = \text{TRUE}$
   $(C_{red} \vee C_{blue}) \wedge \neg(C_{red} \wedge C_{blue}) = \text{TRUE}$

3. Enforce that no adjacent vertices have the same color:
   $\neg(A_{red} \wedge B_{red}) \wedge \neg(A_{blue} \wedge B_{blue}) = \text{TRUE}$
   $\neg(A_{red} \wedge C_{red}) \wedge \neg(A_{blue} \wedge C_{blue}) = \text{TRUE}$
   $\neg(B_{red} \wedge C_{red}) \wedge \neg(B_{blue} \wedge C_{blue}) = \text{TRUE}$

Reduction Example

Graph Coloring reduces to SAT in polynomial time.



Assume that the problem is to check if the graph is 2-colorable.

1. Create the boolean variables:
   A_{red} , A_{blue} , B_{red} , B_{blue} , C_{red} , C_{blue}

2. Enforce that each vertex has one color:

$$(A_{red} \vee A_{blue}) \wedge \neg(A_{red} \wedge A_{blue}) = \text{TRUE}$$
$$(B_{red} \vee B_{blue}) \wedge \neg(B_{red} \wedge B_{blue}) = \text{TRUE}$$
$$(C_{red} \vee C_{blue}) \wedge \neg(C_{red} \wedge C_{blue}) = \text{TRUE}$$

3. Enforce that no adjacent vertices have the same color:

$$\neg(A_{red} \wedge B_{red}) \wedge \neg(A_{blue} \wedge B_{blue}) = \text{TRUE}$$
$$\neg(A_{red} \wedge C_{red}) \wedge \neg(A_{blue} \wedge C_{blue}) = \text{TRUE}$$
$$\neg(B_{red} \wedge C_{red}) \wedge \neg(B_{blue} \wedge C_{blue}) = \text{TRUE}$$

Can be converted to a CNF with $\Theta(E + V)$ clauses

The graph is 2-colorable if the above boolean expressions are satisfiable!

Quiz # 9

How do we show that a problem other than SAT is NP-Complete?

A. Be as clever as Cook and Levin and show how all problems in NP reduce to this new problem.

B. No need! SAT is the only NP-Complete Problem!

C. None of the above.

Quiz # 9

How do we show that a problem other than SAT is NP-Complete?

A. Be as clever as Cook and Levin and show how all problems in NP reduce to this new problem.

B. No need! SAT is the only NP-Complete Problem!

C. None of the above.

Quiz # 9

How do we show that a problem other than SAT is NP-Complete?

A. Be as clever as Cook and Levin and show how all problems in NP reduce to this new problem.

B. No need! SAT is the only NP-Complete Problem!

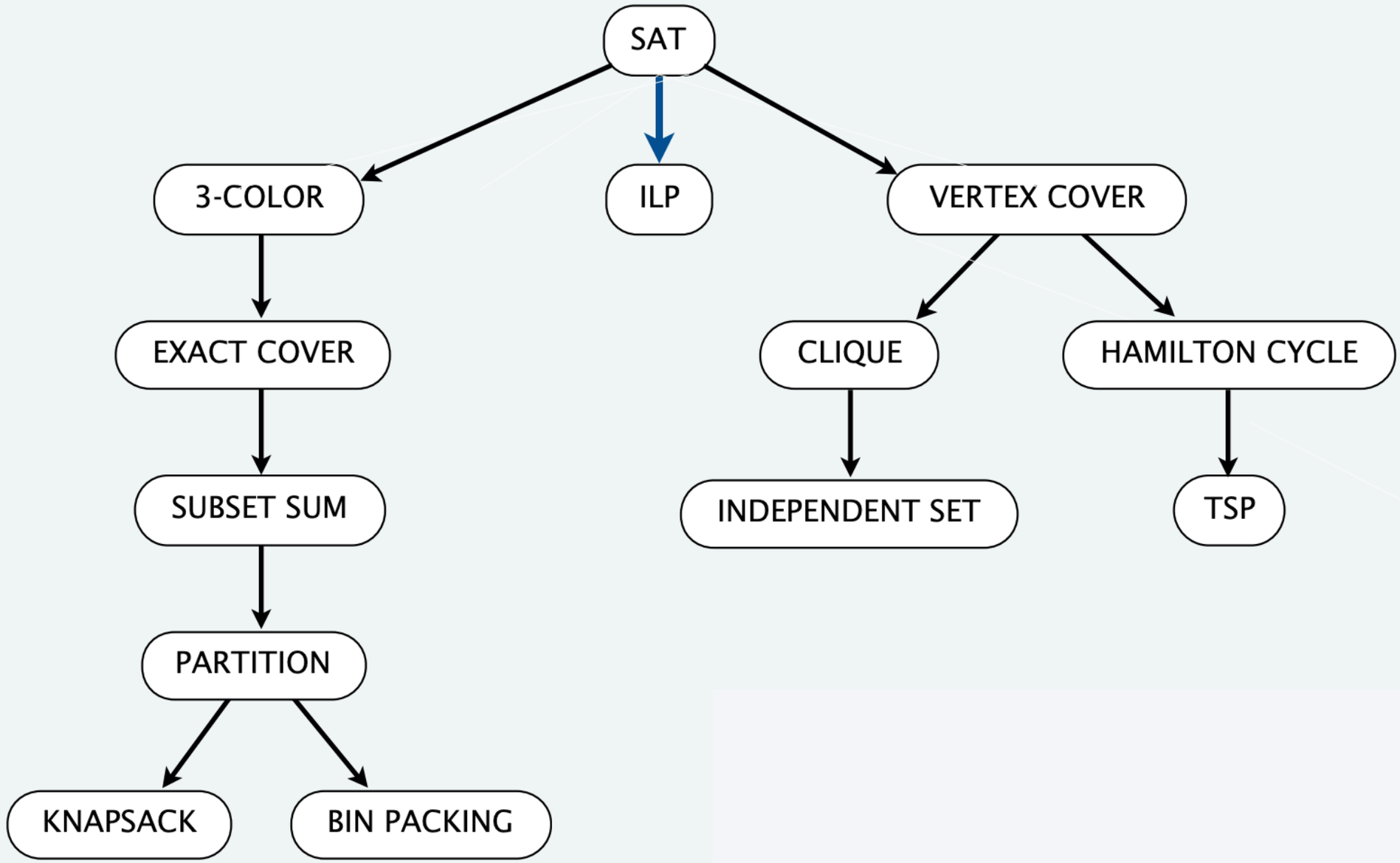
C. None of the above.

To show that a problem is NP-Complete:

1. Show that it is in NP.
2. Show that an NP-Complete problem reduces to it in polynomial time!

If all problems in NP poly-time reduce to A and A poly-time reduces to B, then all problems in NP poly-time reduce to B !

SAT is not The Only NP-Complete Problem!



Dick Karp
1985 Turing Award

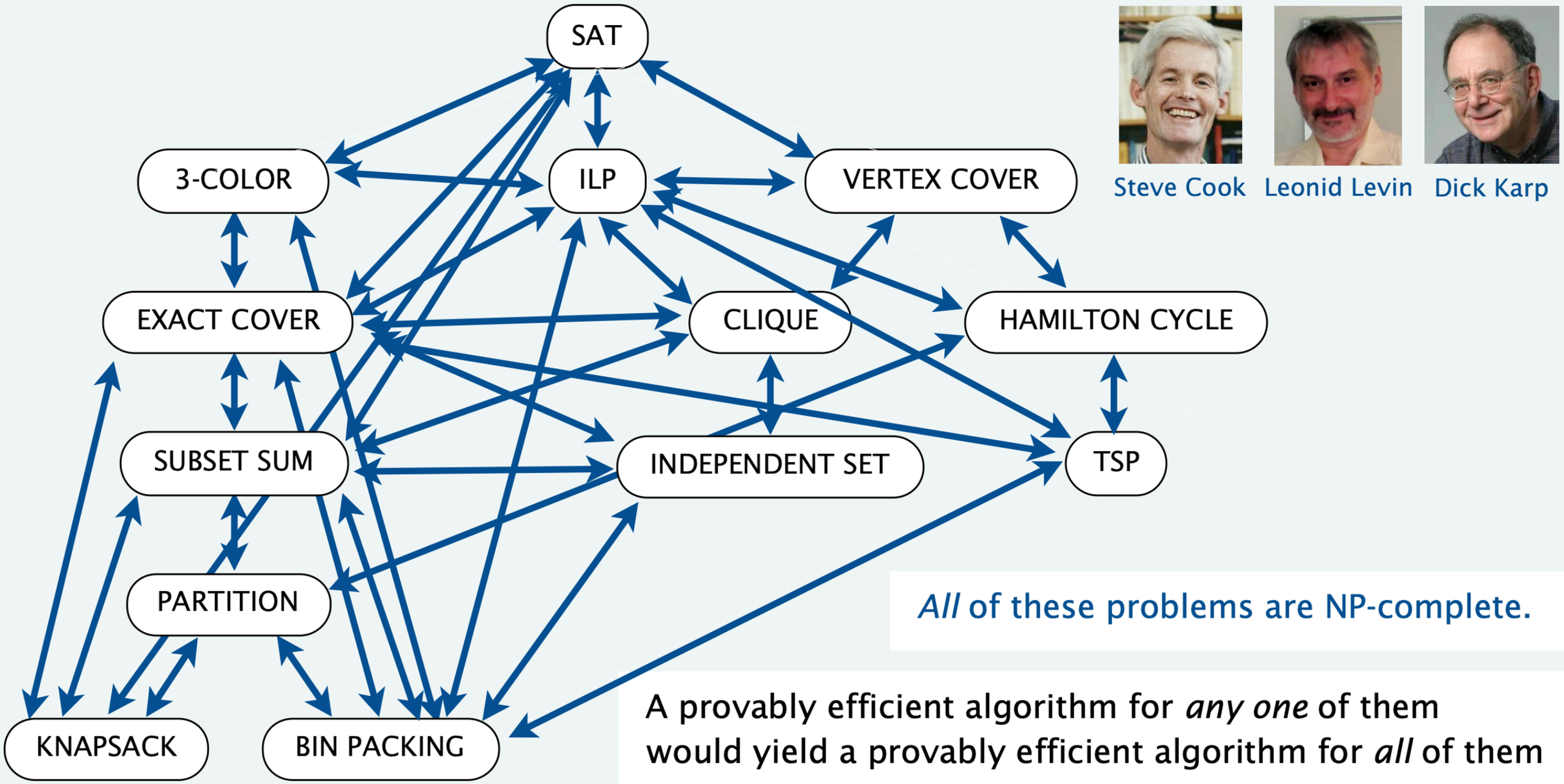
Key Finding. SAT poly-time reduces to many problems!

Implication. All of these problems are NP-Complete!

slide by Kevin Wayne

SAT is not The Only NP-Complete Problem!

adapted from a slide by Kevin Wayne



Steve Cook

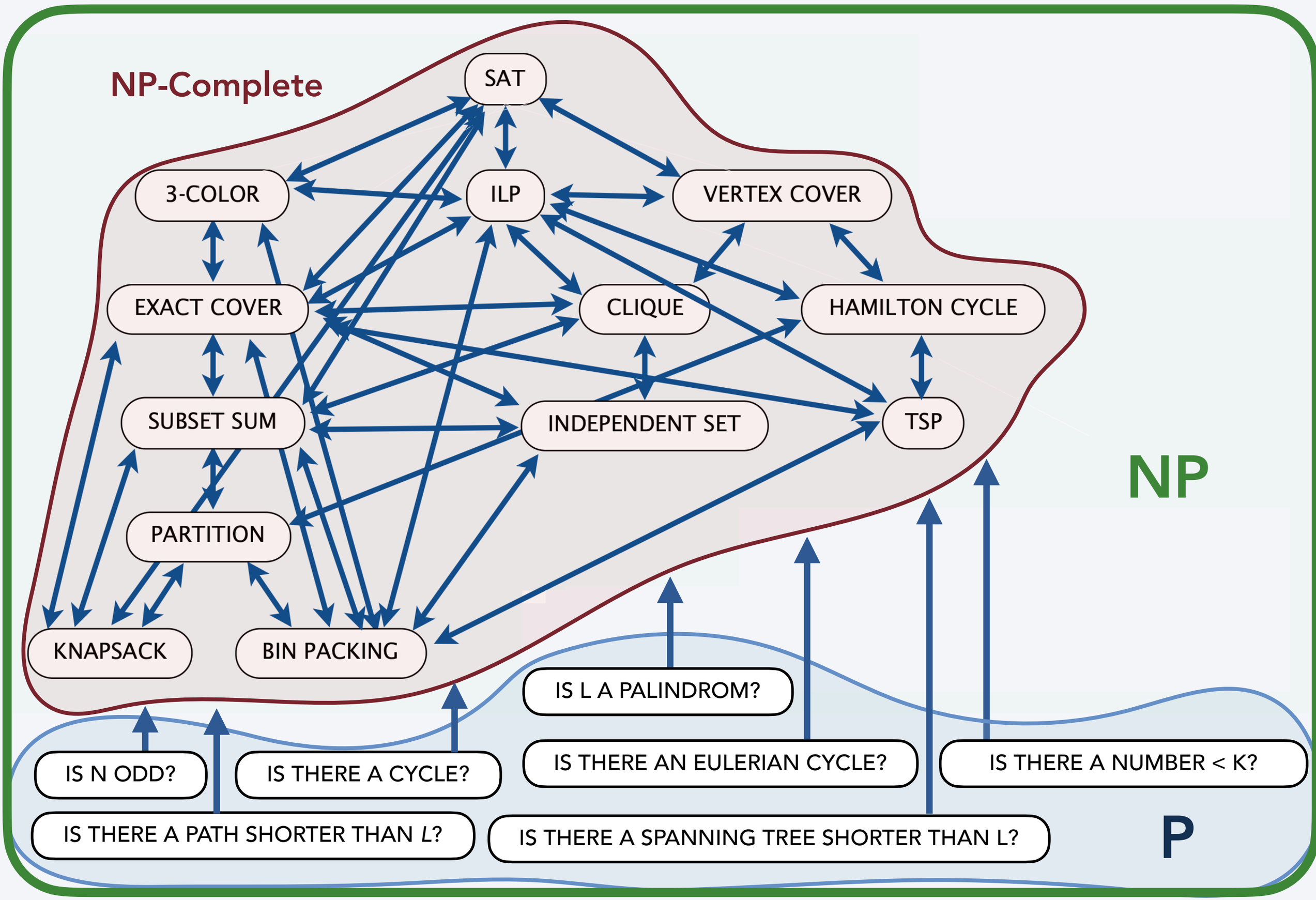
Leonid Levin

Dick Karp

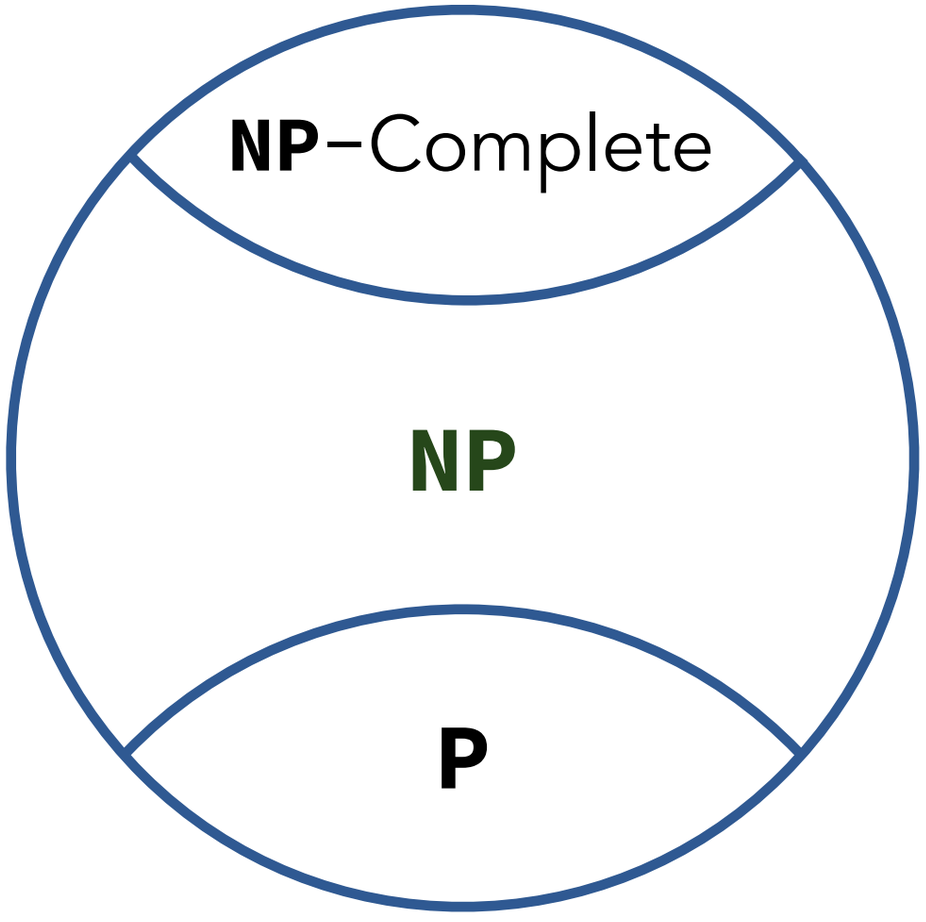
All of these problems are NP-complete.

A provably efficient algorithm for *any one* of them would yield a provably efficient algorithm for *all* of them

World View if P != NP



Again ... Two Possible World Views



If P ≠ NP

vs

If P = NP

Quiz # 10

Are there problems that are in **NP** but are not in **P** and are not **NP**-Complete.

A. Yes.

B. No.

C. None of the above.

Quiz # 10

Are there problems that are in **NP** but are not in **P** and are not **NP**-Complete.

A. Yes.

B. No.

C. None of the above.

Maybe if $\mathbf{P} \neq \mathbf{NP}$.

No if $\mathbf{P} = \mathbf{NP}$.

Quiz # 10

Are there problems that are in **NP** but are not in **P** and are not **NP**-Complete.

A. Yes.

B. No.

C. None of the above.

Maybe if **P** $\neq$ **NP**.

No if **P = NP**.

There are, however, problems in NP that we could not yet prove to be in P and could not also prove to be NP-Complete!

Examples. Integer Factoring and Graph Isomorphism.

NP-Completeness (Proof Examples)

`ILP` (*binary* Integer Linear Programming)

Given a set of inequalities, is there a `0-1` solution?

$$
\begin{array}{rrrrr}
 & x_1 + & x_2 \geq & 1 \\
x_0 & + & x_2 \geq & 1 \\
x_0 + & x_1 + & x_2 \leq & 2
\end{array}
$$

Task. Show that `ILP` is `NP`-Complete.

Example. A solution for the above is:

$$x_0 = 1, \quad x_1 = 1, \quad x_2 = 0$$

NP-Completeness (Proof Examples)

`ILP` (*binary* Integer Linear Programming)

Given a set of inequalities, is there a `0-1` solution?

$$
\begin{array}{rrrrr}
 & x_1 + & x_2 \geq & 1 \\
x_0 & + & x_2 \geq & 1 \\
x_0 + & x_1 + & x_2 \leq & 2
\end{array}
$$

Example. A solution for the above is:

$$x_0 = 1, \quad x_1 = 1, \quad x_2 = 0$$

Task. Show that `ILP` is NP-Complete.

1. `ILP` is in NP.

Given values for the variables, we can verify in polynomial time if the inequalities are true.

NP-Completeness (Proof Examples)

ILP (*binary* Integer Linear Programming)

Given a set of inequalities, is there a 0-1 solution?

$$
\begin{array}{rrrrrr}
 & x_1 & + & x_2 & \geq & 1 \\
x_0 & & + & x_2 & \geq & 1 \\
x_0 & + & x_1 & + & x_2 & \leq & 2
\end{array}
$$

Example. A solution for the above is:

$$x_0 = 1, \quad x_1 = 1, \quad x_2 = 0$$

Task. Show that ILP is NP-Complete.

1. ILP is in NP.

Given values for the variables, we can verify in polynomial time if the inequalities are true.

2. SAT poly-time reduces to ILP.

$$
\begin{array}{rcrcrcl}
\bar{x}_1 & \vee & x_2 & \vee & x_3 & & = \text{TRUE} \\
x_1 & \vee & \bar{x}_2 & \vee & x_3 & & = \text{TRUE} \\
\bar{x}_1 & \vee & \bar{x}_2 & \vee & \bar{x}_3 & & = \text{TRUE} \\
\bar{x}_1 & \vee & \bar{x}_2 & & \vee & x_4 & = \text{TRUE} \\
 & & \bar{x}_2 & \vee & x_3 & \vee & x_4 = \text{TRUE}
\end{array}
$$

Example SAT instance

$$
\begin{array}{rcrcrcrcl}
(1 - x_1) & + & x_2 & + & x_3 & & & & \geq 1 \\
x_1 & + & (1 - x_2) & + & x_3 & & & & \geq 1 \\
(1 - x_1) & + & (1 - x_2) & + & (1 - x_3) & & & & \geq 1 \\
(1 - x_1) & + & (1 - x_2) & & & + & x_4 & & \geq 1 \\
 & & (1 - x_2) & + & x_3 & + & x_4 & & \geq 1
\end{array}
$$

Equivalent ILP instance.

Examples by Kevin Wayne

NP-Completeness (Proof Examples)

ILP (*binary* Integer Linear Programming)

Given a set of inequalities, is there a 0-1 solution?

$$
\begin{array}{rrrrr}
 & x_1 + & x_2 \geq & 1 \\
x_0 & + & x_2 \geq & 1 \\
x_0 + & x_1 + & x_2 \leq & 2
\end{array}
$$

Task. Show that ILP is NP-Complete.

Example. A solution for the above is:

$$x_0 = 1, \quad x_1 = 1, \quad x_2 = 0$$

2. SAT poly-time reduces to ILP.

$$
\begin{array}{rcrcrcrl}
\bar{x}_1 & \vee & x_2 & \vee & x_3 & & & = \text{TRUE} \\
x_1 & \vee & \bar{x}_2 & \vee & x_3 & & & = \text{TRUE} \\
\bar{x}_1 & \vee & \bar{x}_2 & \vee & \bar{x}_3 & & & = \text{TRUE} \\
\bar{x}_1 & \vee & \bar{x}_2 & & & \vee & x_4 & = \text{TRUE} \\
 & & \bar{x}_2 & \vee & x_3 & \vee & x_4 & = \text{TRUE}
\end{array}
$$

Example SAT instance

$$
\begin{array}{rcrcrcrcrl}
(1 - x_1) & + & & & x_2 & + & & & x_3 & \geq 1 \\
x_1 & + & (1 - x_2) & + & & & x_3 & & & \geq 1 \\
(1 - x_1) & + & (1 - x_2) & + & (1 - x_3) & & & & & \geq 1 \\
(1 - x_1) & + & (1 - x_2) & & & & & + & x_4 & \geq 1 \\
 & & (1 - x_2) & + & & & x_3 & + & x_4 & \geq 1
\end{array}
$$

Equivalent ILP instance.

A clause is true iff any variable is true and is not negated or is false and is negated.

An inequality ≥ 1 iff any variable is 1 and is not negated or is 0 and is negated.

Creating these inequalities is linear in the number of boolean clauses (i.e. this is a polytime reduction)

Examples by Kevin Wayne

NP-Completeness (Proof Examples)

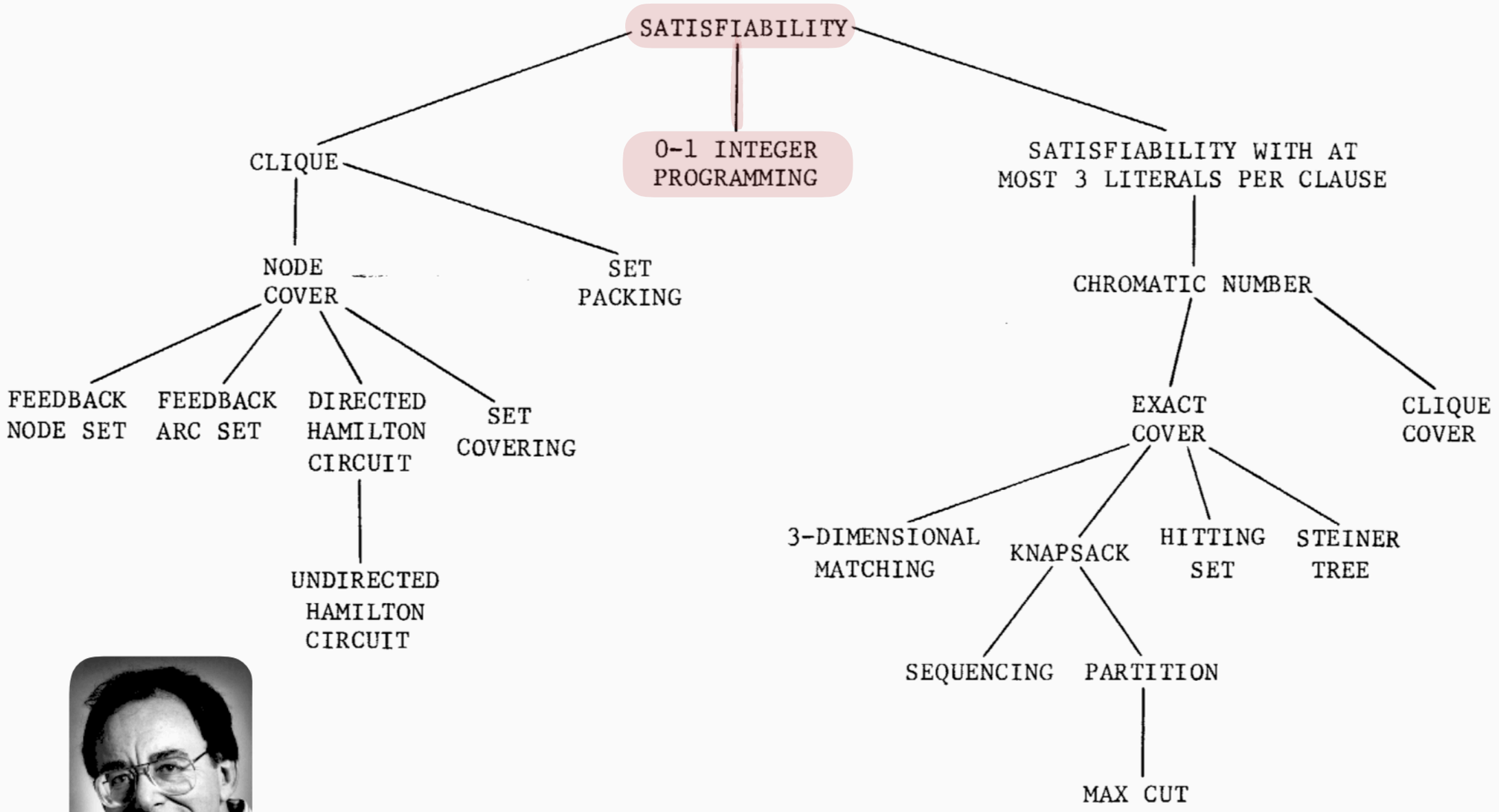


FIGURE 1 - Complete Problems

Dick Karp (1972)
1985 Turing Award

NP-Completeness (Proof Examples)

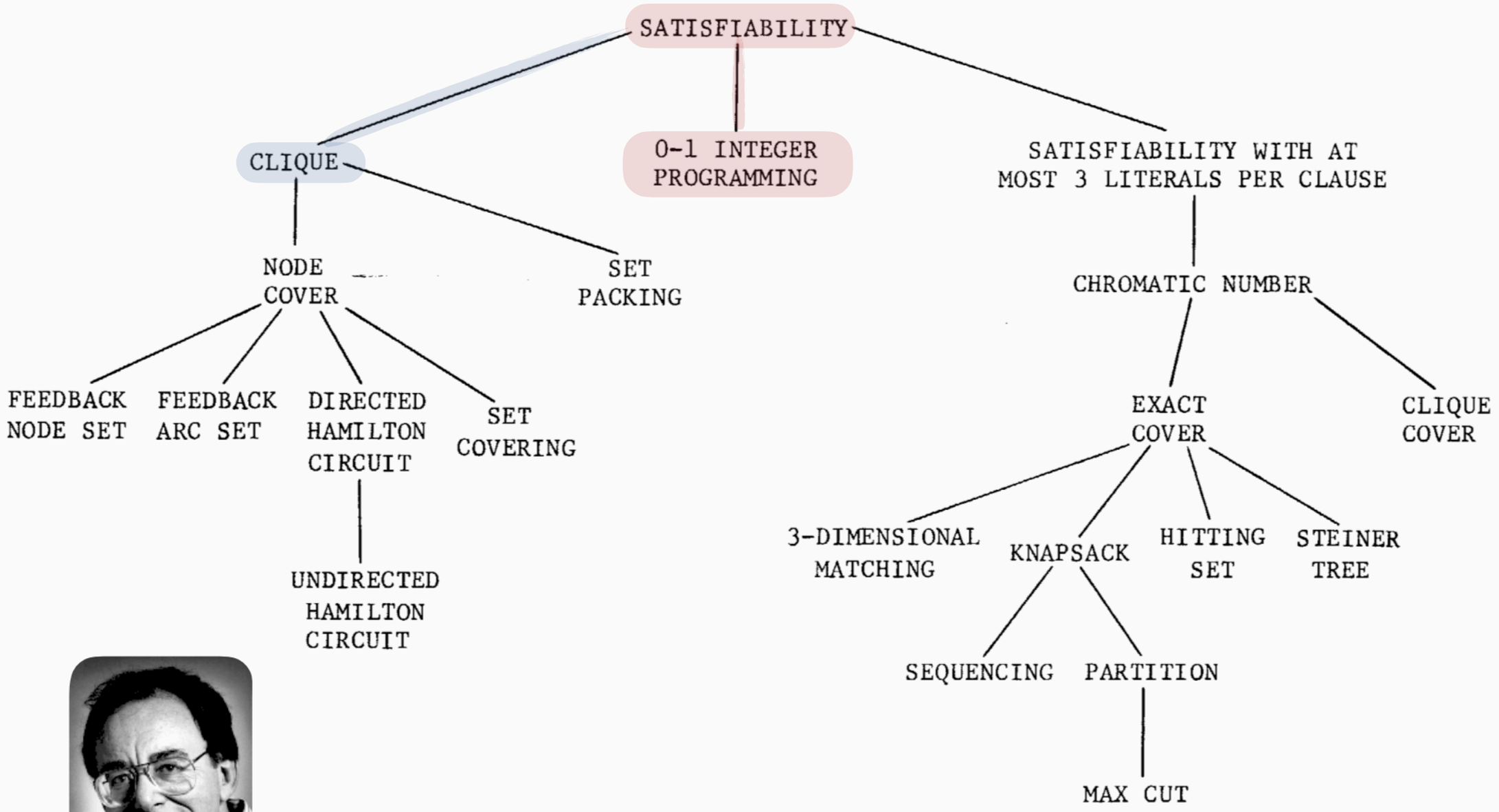


FIGURE 1 - Complete Problems

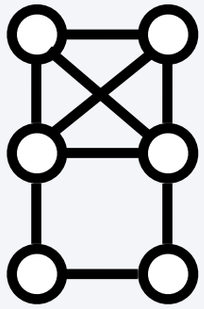
RICHARD M. KARP

Dick Karp (1972)
1985 Turing Award

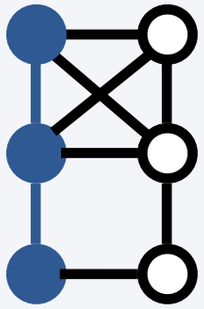
NP-Completeness (Proof Examples)

CLIQUE Given a graph G, and an integer m, is there a complete subgraph of size m vertices?

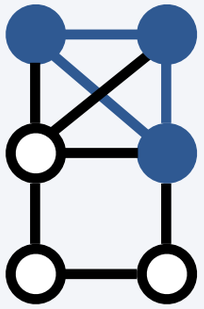
Task. Show that CLIQUE is NP-Complete.



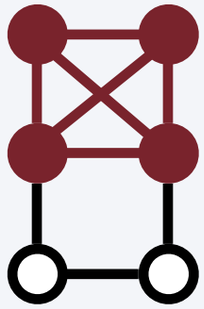
original graph



not a clique



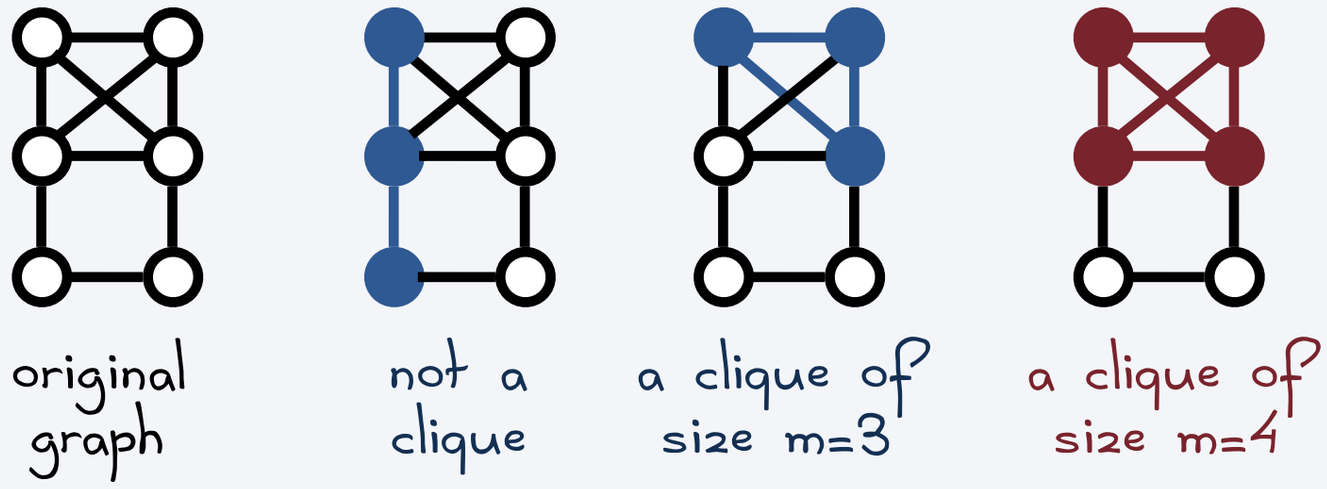
a clique of size m=3



a clique of size m=4

NP-Completeness (Proof Examples)

CLIQUE Given a graph G, and an integer m, is there a complete subgraph of size m vertices?



Task. Show that CLIQUE is NP-Complete.

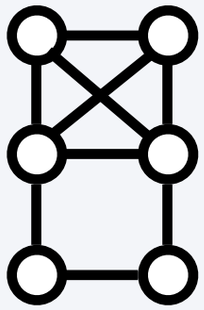
1. CLIQUE is in NP.

Given a graph G and a subgraph of vertices S, we can check in polynomial time if the size of the set is m and every vertex in S is connected to every other vertex in S.

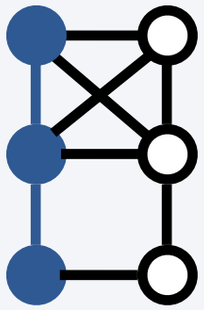
Examples by Kevin Wayne

NP-Completeness (Proof Examples)

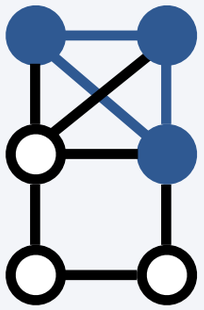
CLIQUE Given a graph G, and an integer m, is there a complete subgraph of size m vertices?



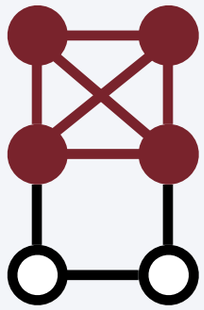
original graph



not a clique



a clique of size m=3



a clique of size m=4

Task. Show that CLIQUE is NP-Complete.

1. CLIQUE is in NP.
2. SAT poly-time reduces to CLIQUE.

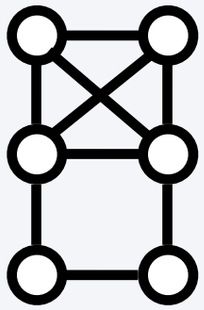
Example

$$(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3)$$

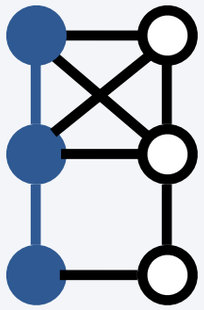
Examples by Kevin Wayne

NP-Completeness (Proof Examples)

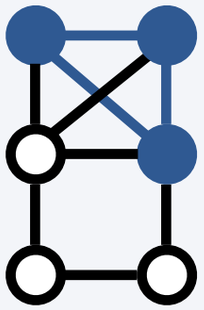
CLIQUE Given a graph G, and an integer m, is there a complete subgraph of size m vertices?



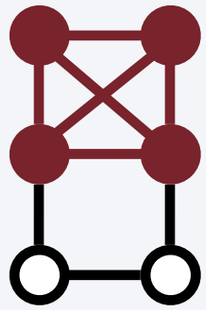
original graph



not a clique



a clique of size m=3



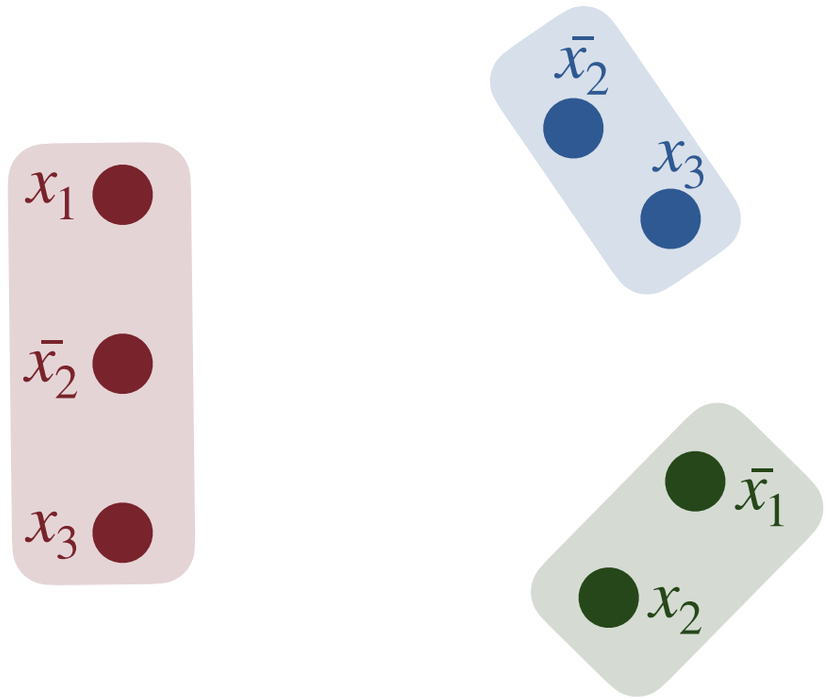
a clique of size m=4

Task. Show that CLIQUE is NP-Complete.

1. CLIQUE is in NP.
2. SAT poly-time reduces to CLIQUE.
   - Create a group of vertices for every clause (m groups)

Example

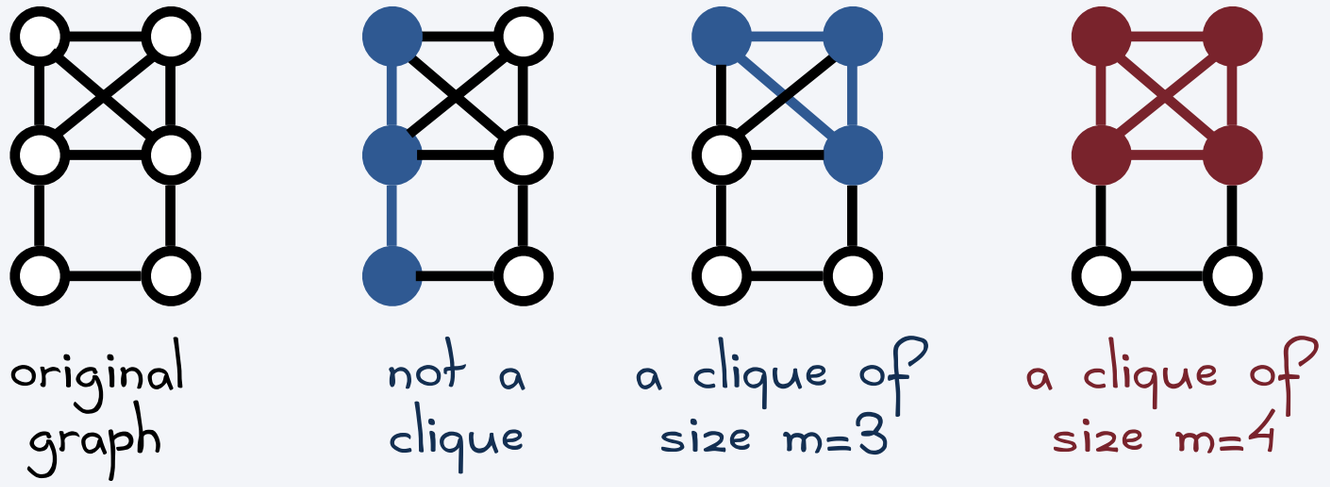
$(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3)$



Examples by Kevin Wayne

NP-Completeness (Proof Examples)

CLIQUE Given a graph G, and an integer m, is there a complete subgraph of size m vertices?



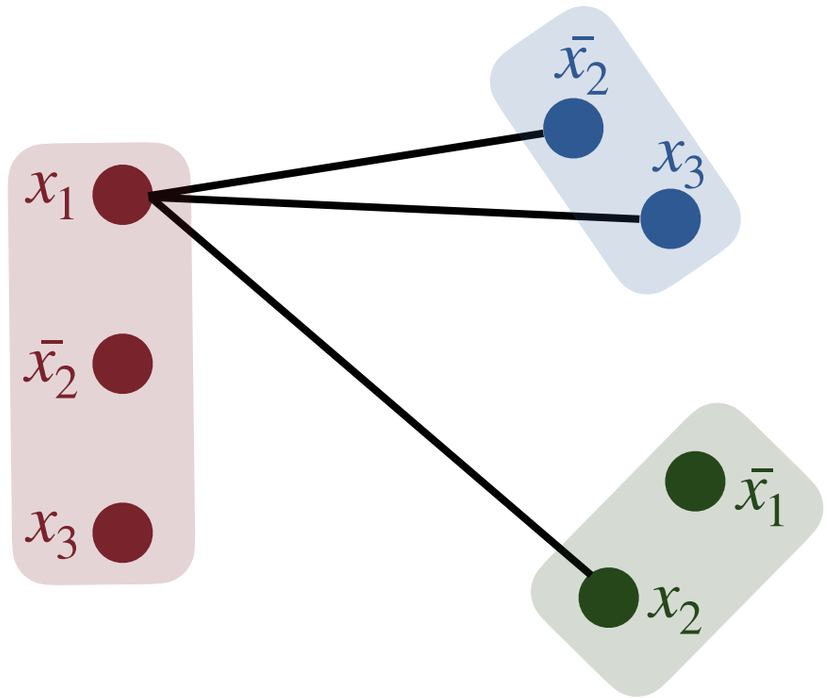
Task. Show that CLIQUE is NP-Complete.

1. CLIQUE is in NP.
2. SAT poly-time reduces to CLIQUE.

- Create a group of vertices for every clause (m groups)
- Connect every vertex to all the other vertices in the other groups unless the variable is its negation.

Example

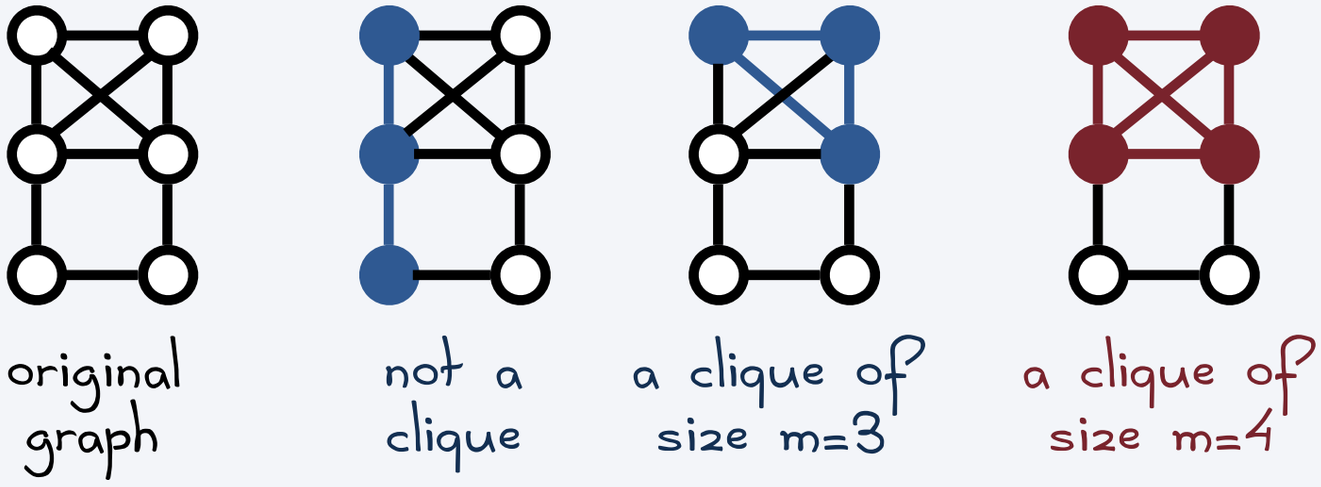
$(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3)$



Examples by Kevin Wayne

NP-Completeness (Proof Examples)

CLIQUE Given a graph G, and an integer m, is there a complete subgraph of size m vertices?

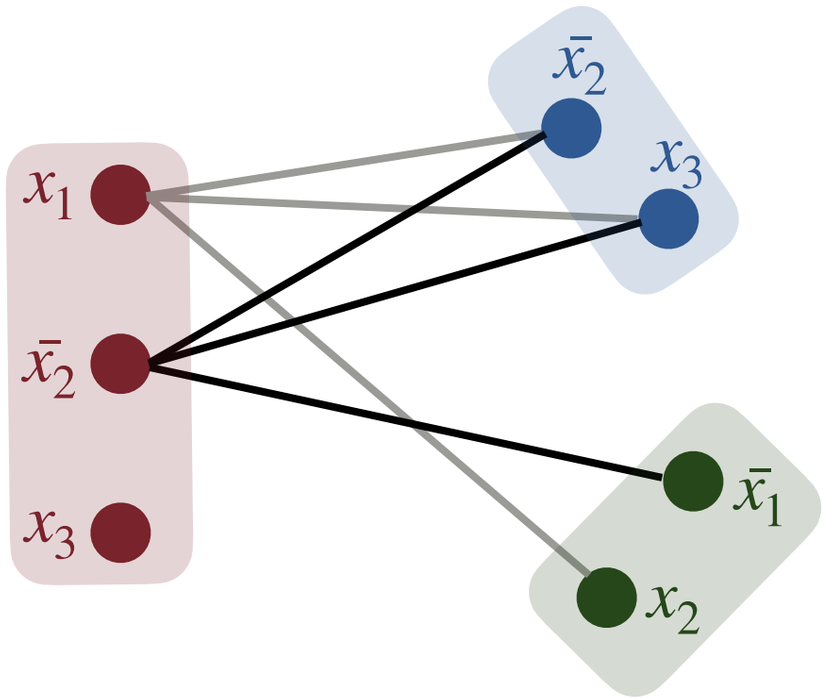


Task. Show that CLIQUE is NP-Complete.

1. CLIQUE is in NP.
2. SAT poly-time reduces to CLIQUE.
   - Create a group of vertices for every clause (m groups)
   - Connect every vertex to all the other vertices in the other groups unless the variable is its negation.

Example

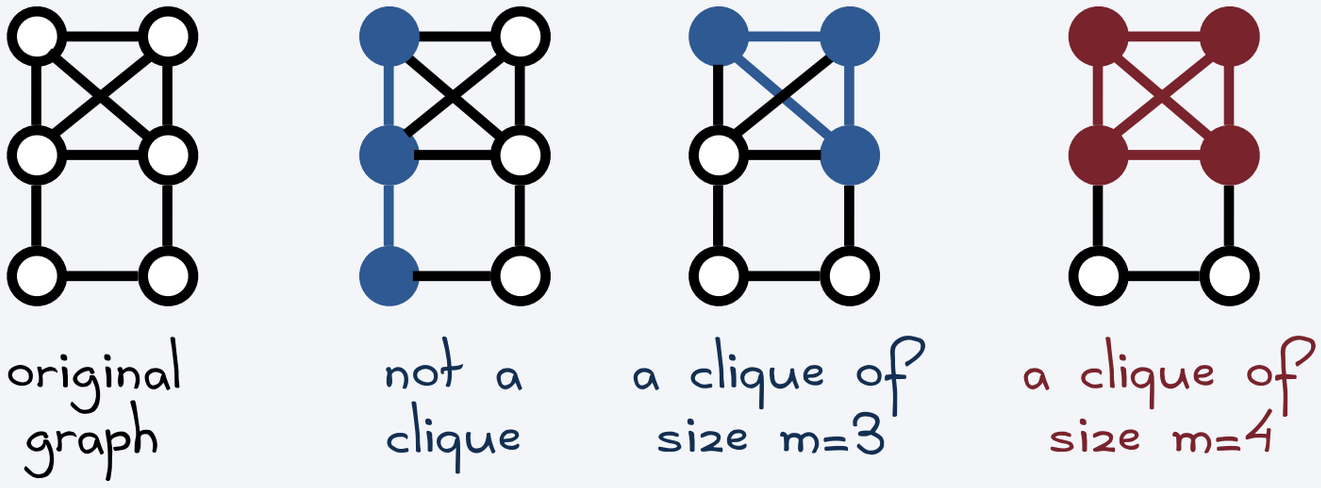
$(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3)$



Examples by Kevin Wayne

NP-Completeness (Proof Examples)

CLIQUE Given a graph G, and an integer m, is there a complete subgraph of size m vertices?

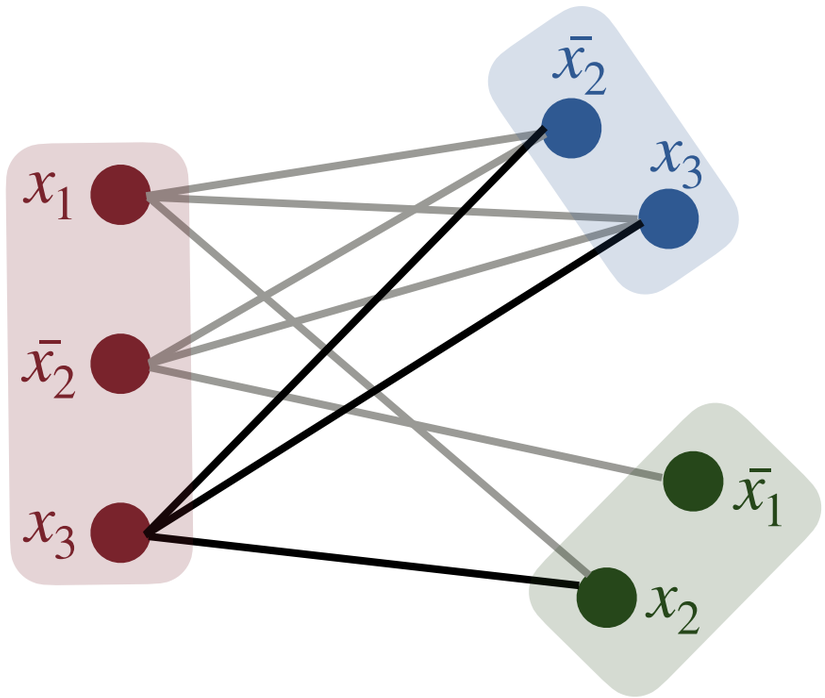


Task. Show that CLIQUE is NP-Complete.

1. CLIQUE is in NP.
2. SAT poly-time reduces to CLIQUE.
   - Create a group of vertices for every clause (m groups)
   - Connect every vertex to all the other vertices in the other groups unless the variable is its negation.

Example

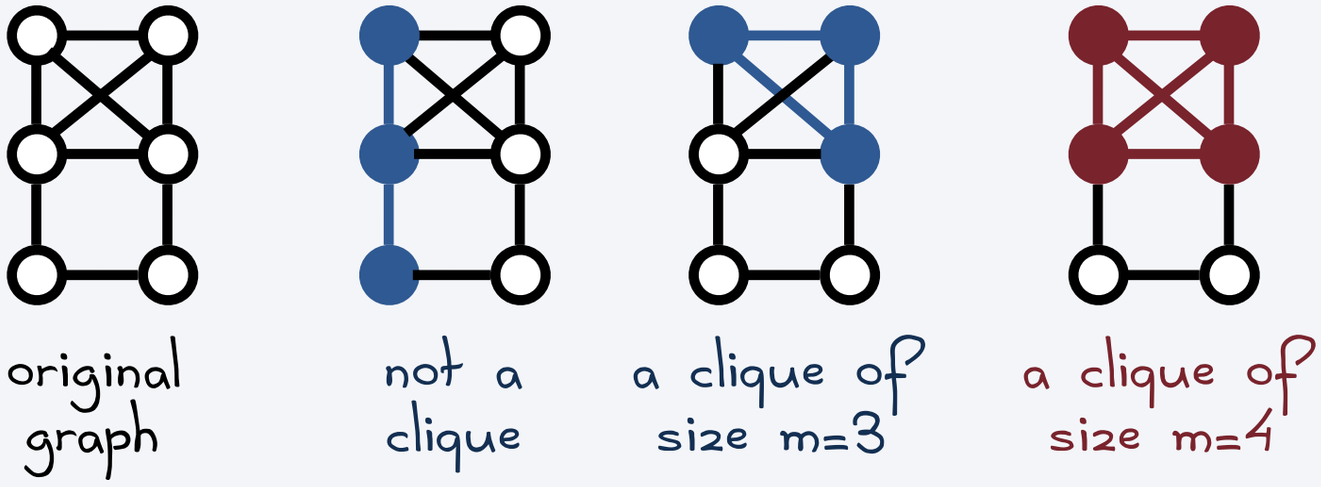
$(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3)$



Examples by Kevin Wayne

NP-Completeness (Proof Examples)

CLIQUE Given a graph G, and an integer m, is there a complete subgraph of size m vertices?



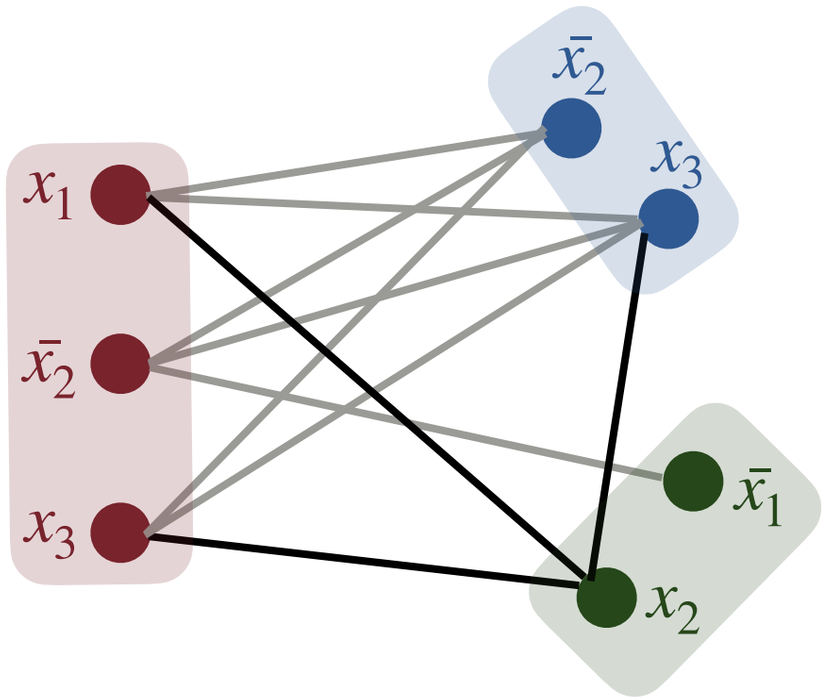
Task. Show that CLIQUE is NP-Complete.

1. CLIQUE is in NP.
2. SAT poly-time reduces to CLIQUE.

- Create a group of vertices for every clause (m groups)
- Connect every vertex to all the other vertices in the other groups unless the variable is its negation.

Example

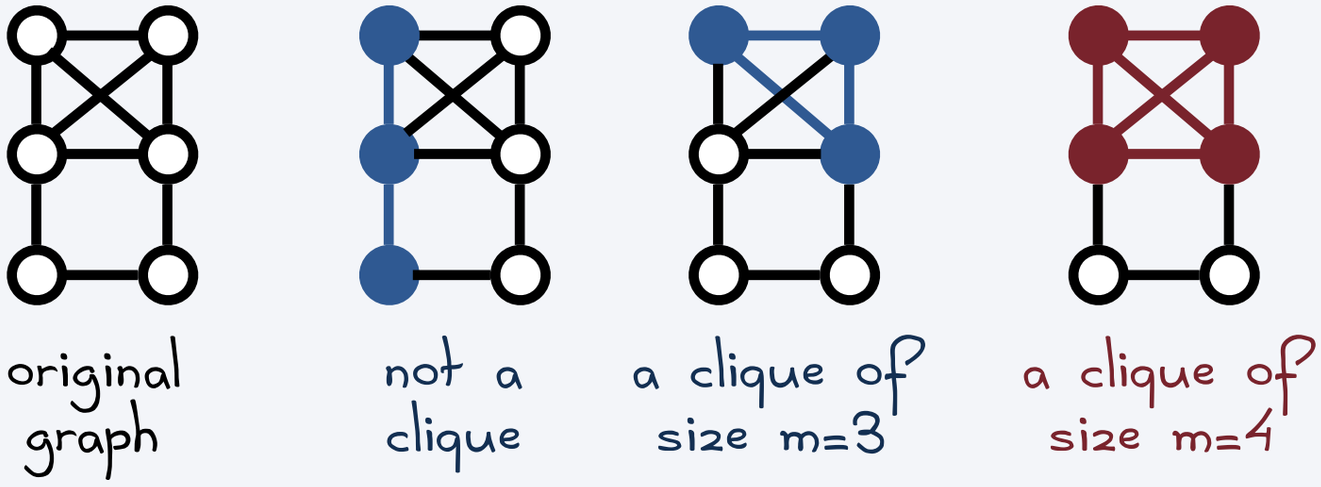
$(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3)$



Examples by Kevin Wayne

NP-Completeness (Proof Examples)

CLIQUE Given a graph G, and an integer m, is there a complete subgraph of size m vertices?



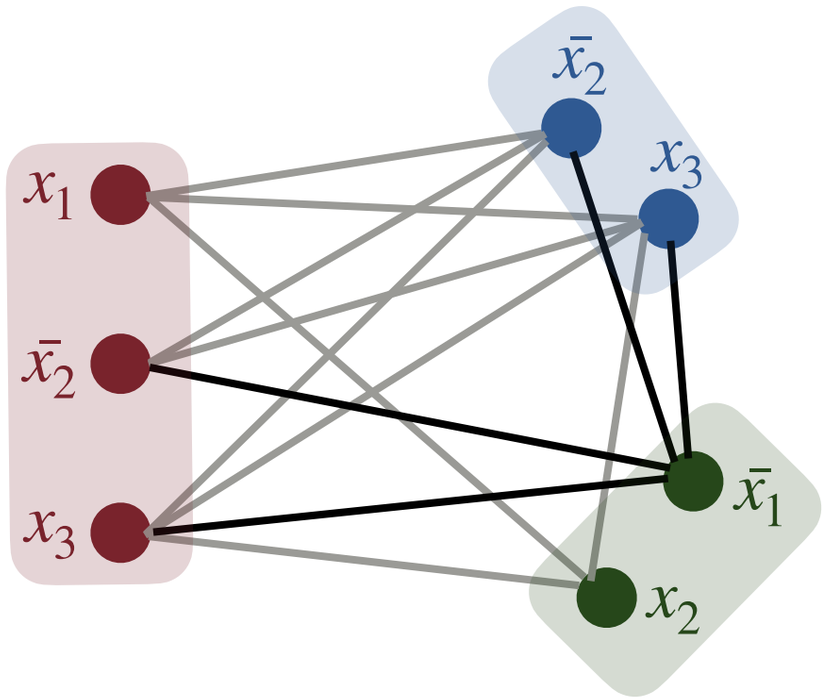
Task. Show that CLIQUE is NP-Complete.

1. CLIQUE is in NP.
2. SAT poly-time reduces to CLIQUE.

- Create a group of vertices for every clause (m groups)
- Connect every vertex to all the other vertices in the other groups unless the variable is its negation.

Example

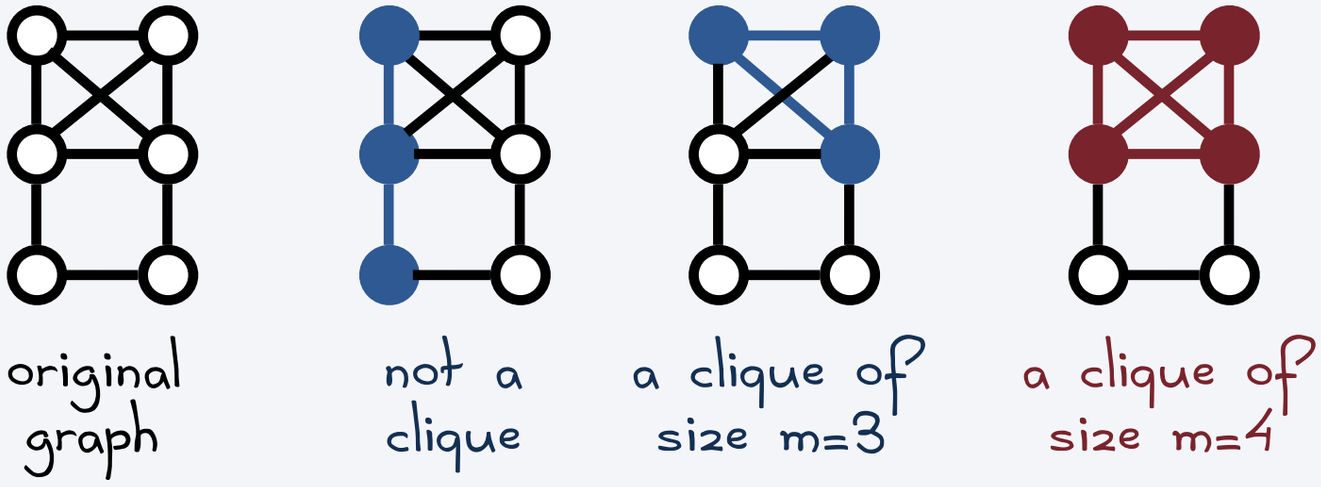
$(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3)$



Examples by Kevin Wayne

NP-Completeness (Proof Examples)

CLIQUE Given a graph G, and an integer m, is there a complete subgraph of size m vertices?



Task. Show that CLIQUE is NP-Complete.

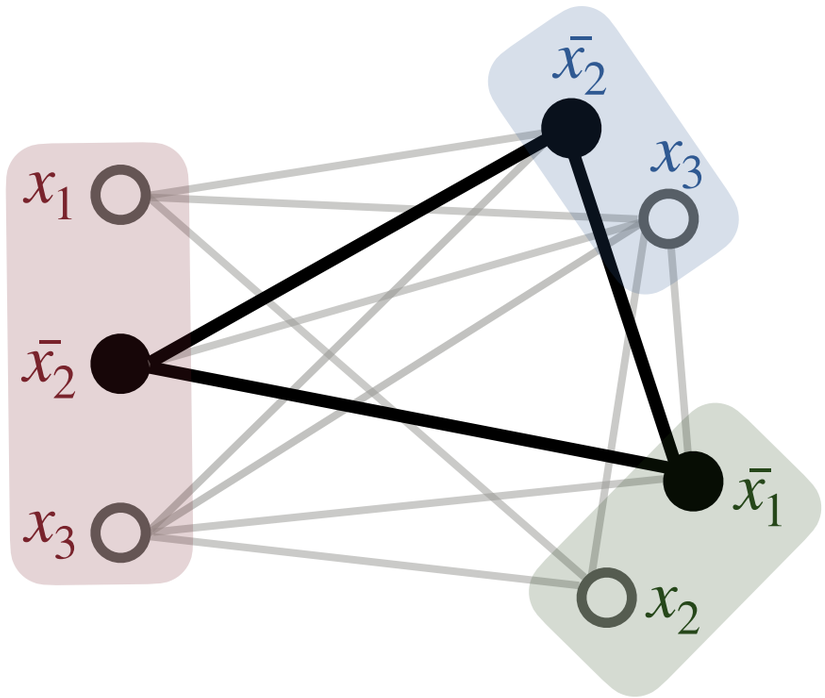
1. CLIQUE is in NP.
2. SAT poly-time reduces to CLIQUE.

- Create a group of vertices for every clause (m groups)
- Connect every vertex to all the other vertices in the other groups unless the variable is its negation.
- A clique of size m corresponds to m literals being true (formula is satisfiable).

The clique contains exactly 1 vertex from each group (vertices in the same group are not connected) and a variable and its negation can't be in the clique.

Example

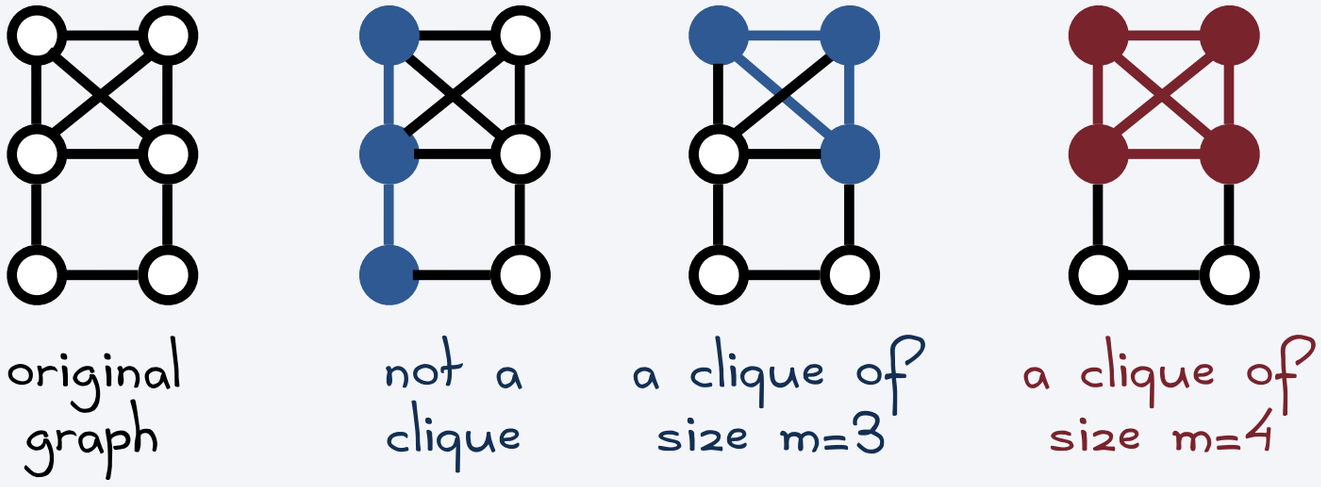
$(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3)$



Examples by Kevin Wayne

NP-Completeness (Proof Examples)

CLIQUE Given a graph G, and an integer m, is there a complete subgraph of size m vertices?



Task. Show that CLIQUE is NP-Complete.

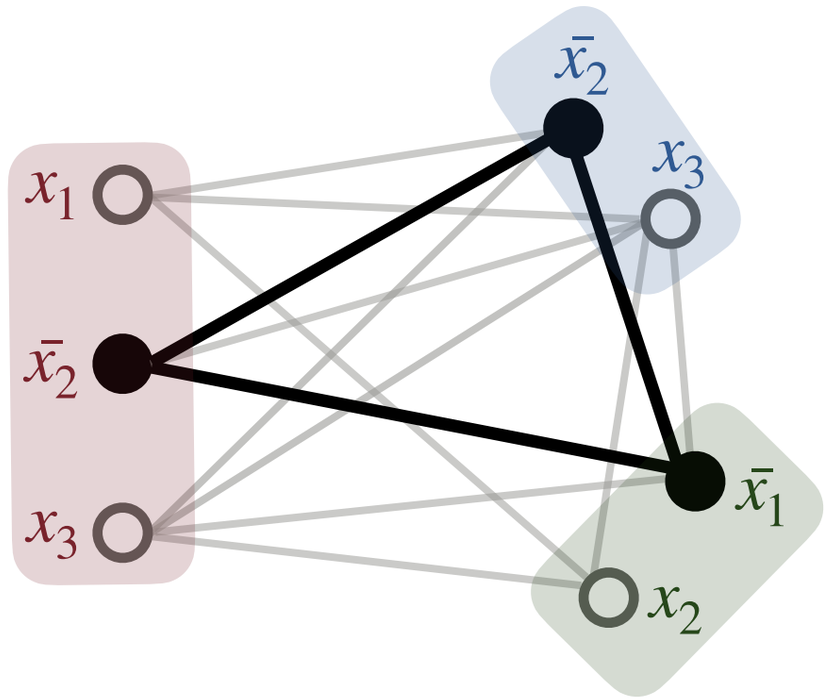
1. CLIQUE is in NP.
2. SAT poly-time reduces to CLIQUE.

- Create a group of vertices for every clause (m groups)
- Connect every vertex to all the other vertices in the other groups unless the variable is its negation.
- A clique of size m corresponds to m literals being true (formula is satisfiable).

If the number of literals = N, the construction takes $O(N^2)$ time (i.e. this is a polytime reduction)

Example

$(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3)$



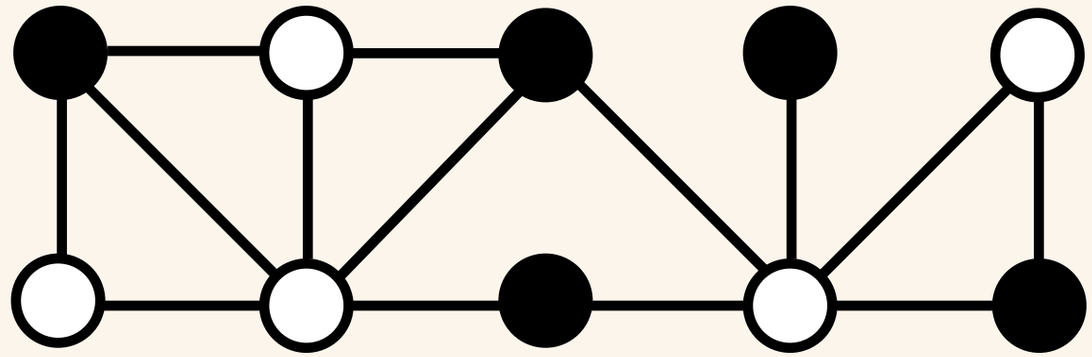
Examples by Kevin Wayne

NP-Completeness (Proof Examples)

`INDEPENDENT-SET (IS)`

Given a graph and an integer k, is there a subset of k vertices such that no two vertices are adjacent?

Task. Show that `IS` is NP-Complete.

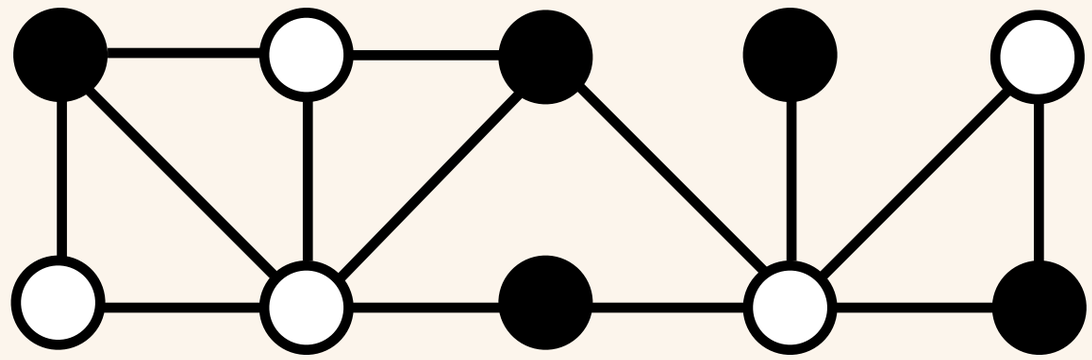


Example. Black vertices form an independent set of size 5

NP-Completeness (Proof Examples)

`INDEPENDENT-SET (IS)`

Given a graph and an integer k, is there a subset of k vertices such that no two vertices are adjacent?



Example. Black vertices form an independent set of size 5

Task. Show that `IS` is NP-Complete.

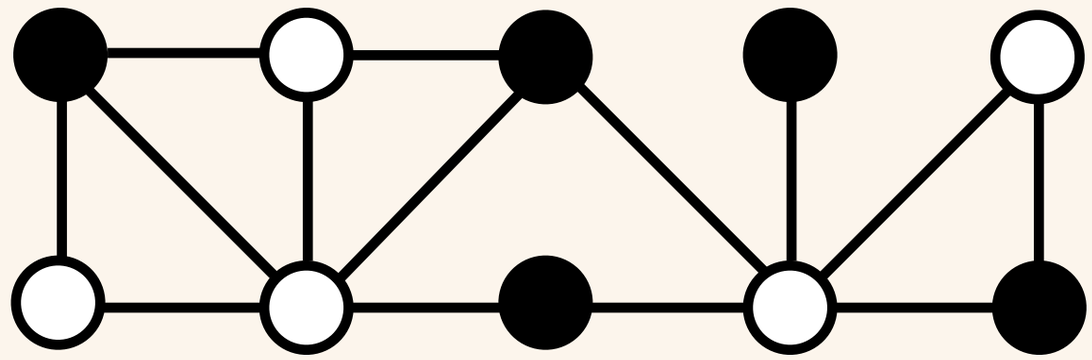
1. `IS` is in NP.

   Given a set S of vertices in G, we can verify in polynomial time if any two are adjacent and if $|S| = k$.

NP-Completeness (Proof Examples)

INDEPENDENT-SET (IS)

Given a graph and an integer k, is there a subset of k vertices such that no two vertices are adjacent?



Example. Black vertices form an independent set of size 5

Task. Show that IS is NP-Complete.

1. IS is in NP.

Given a set S of vertices in G, we can verify in polynomial time if any two are adjacent and if $|S| = k$.

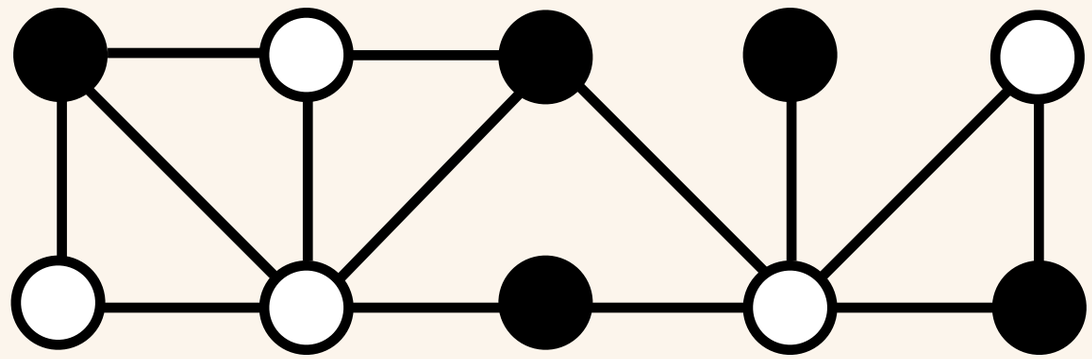
2. SAT poly-time reduces to IS.

Example. $(x_1 \vee \bar{x_2} \vee \bar{x_3}) \wedge (\bar{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x_2} \vee x_3)$

NP-Completeness (Proof Examples)

INDEPENDENT-SET (IS)

Given a graph and an integer k, is there a subset of k vertices such that no two vertices are adjacent?



Example. Black vertices form an independent set of size 5

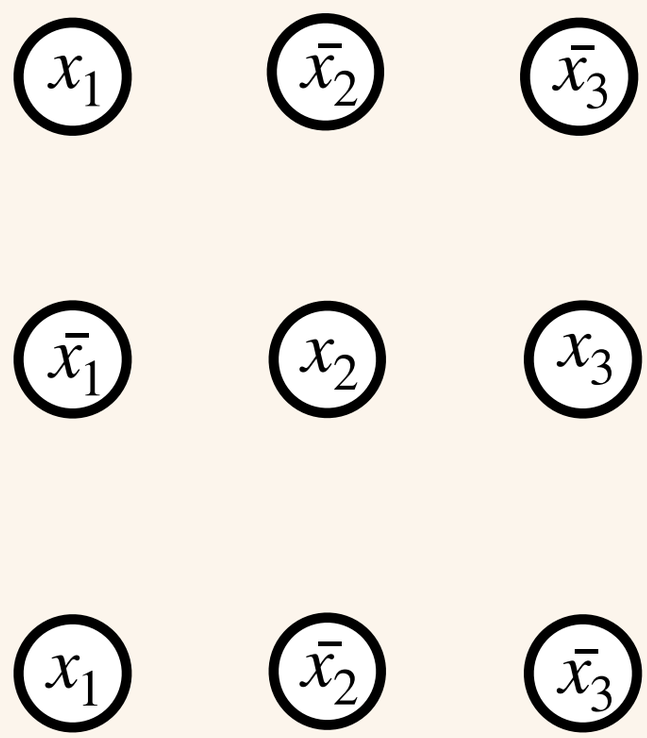
Task. Show that IS is NP-Complete.

1. IS is in NP.

   Given a set S of vertices in G, we can verify in polynomial time if any two are adjacent and if $|S| = k$.

2. SAT poly-time reduces to IS.

   - Create a node for each literal in each clause.

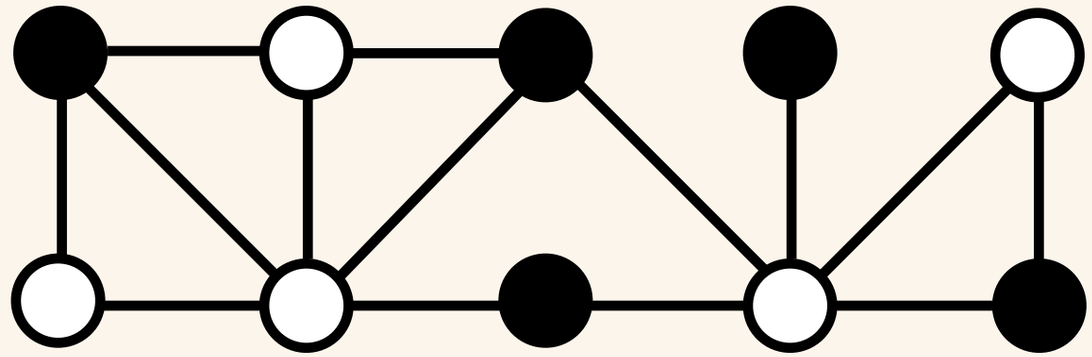


Example. $(x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$

NP-Completeness (Proof Examples)

`INDEPENDENT-SET (IS)`

Given a graph and an integer k, is there a subset of k vertices such that no two vertices are adjacent?



Example. Black vertices form an independent set of size 5

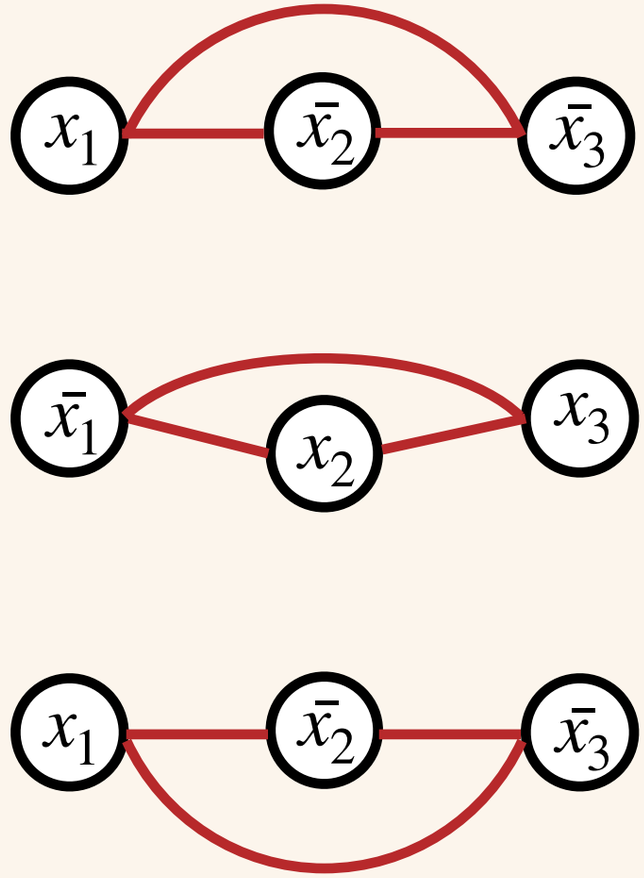
Task. Show that `IS` is NP-Complete.

1. `IS` is in NP.

   Given a set S of vertices in G, we can verify in polynomial time if any two are adjacent and if $|S| = k$.

2. `SAT` poly-time reduces to `IS`.

   - Create a node for each literal in each clause.
   - Connect each node to the literals in the same clause.

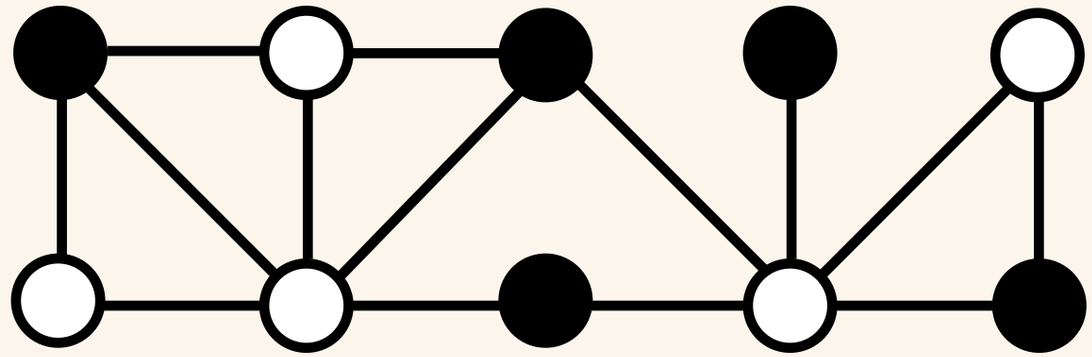


Example. $(x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$

NP-Completeness (Proof Examples)

`INDEPENDENT-SET (IS)`

Given a graph and an integer k, is there a subset of k vertices such that no two vertices are adjacent?



Example. Black vertices form an independent set of size 5

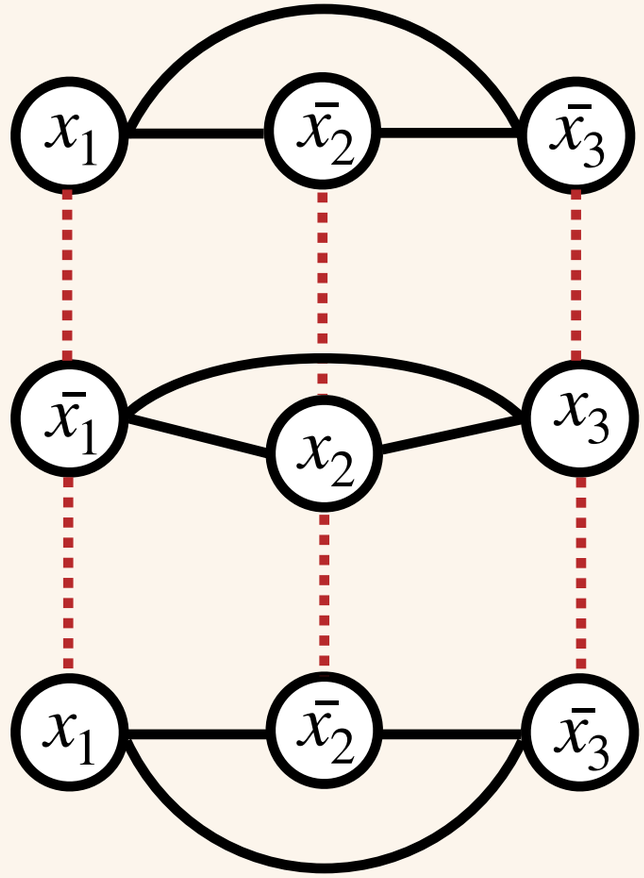
Task. Show that `IS` is NP-Complete.

1. `IS` is in NP.

Given a set S of vertices in G, we can verify in polynomial time if any two are adjacent and if $|S| = k$.

2. `SAT` poly-time reduces to `IS`.

- Create a node for each literal in each clause.
- Connect each node to the literals in the same clause.
- Connect each literal to its negation.

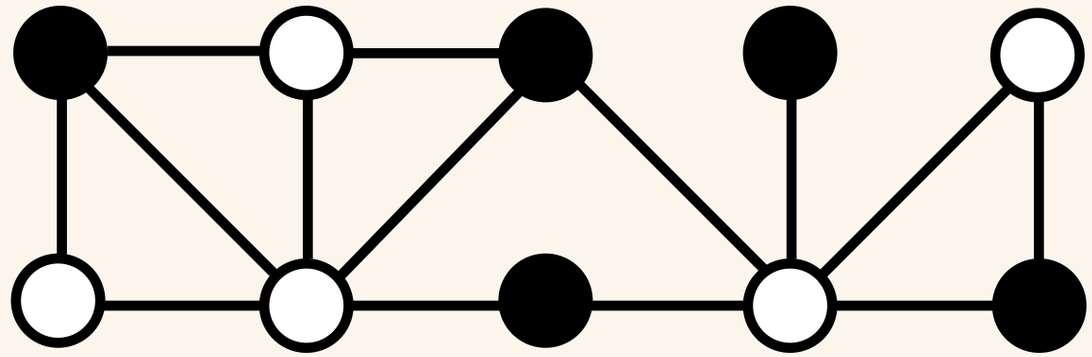


Example. $(x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$

NP-Completeness (Proof Examples)

INDEPENDENT-SET (IS)

Given a graph and an integer k, is there a subset of k vertices such that no two vertices are adjacent?



Example. Black vertices form an independent set of size 5

Task. Show that IS is NP-Complete.

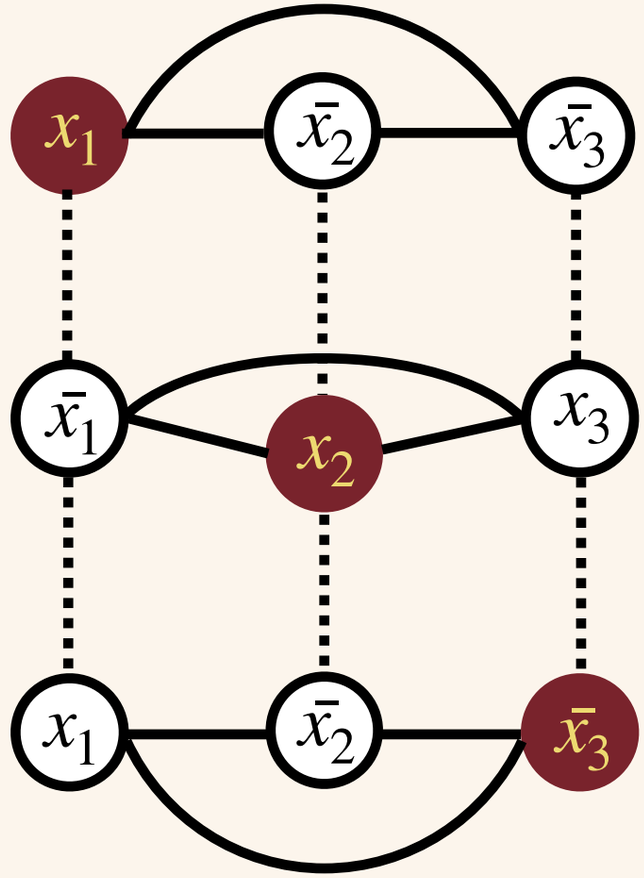
1. IS is in NP.

Given a set S of vertices in G, we can verify in polynomial time if any two are adjacent and if $|S| = k$.

2. SAT poly-time reduces to IS.

- Create a node for each literal in each clause.
- Connect each node to the literals in the same clause.
- Connect each literal to its negation.
- The expression is satisfiable iff there is an independent set of size = the number of clauses.

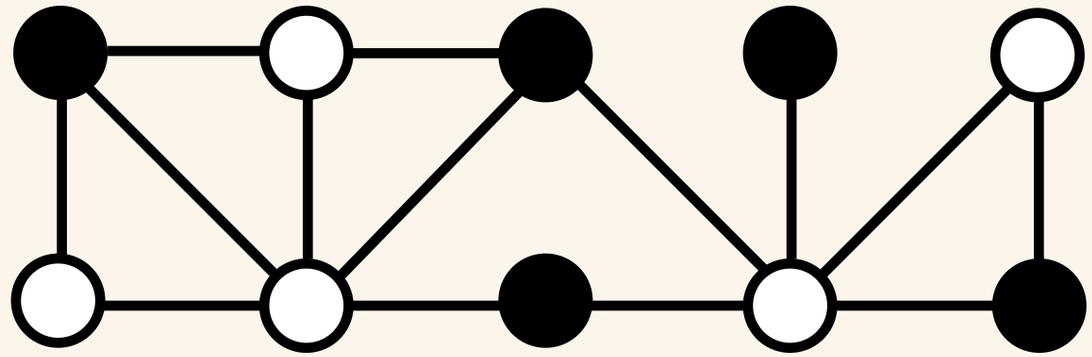
Example. $(x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$



NP-Completeness (Proof Examples)

`INDEPENDENT-SET (IS)`

Given a graph and an integer k, is there a subset of k vertices such that no two vertices are adjacent?



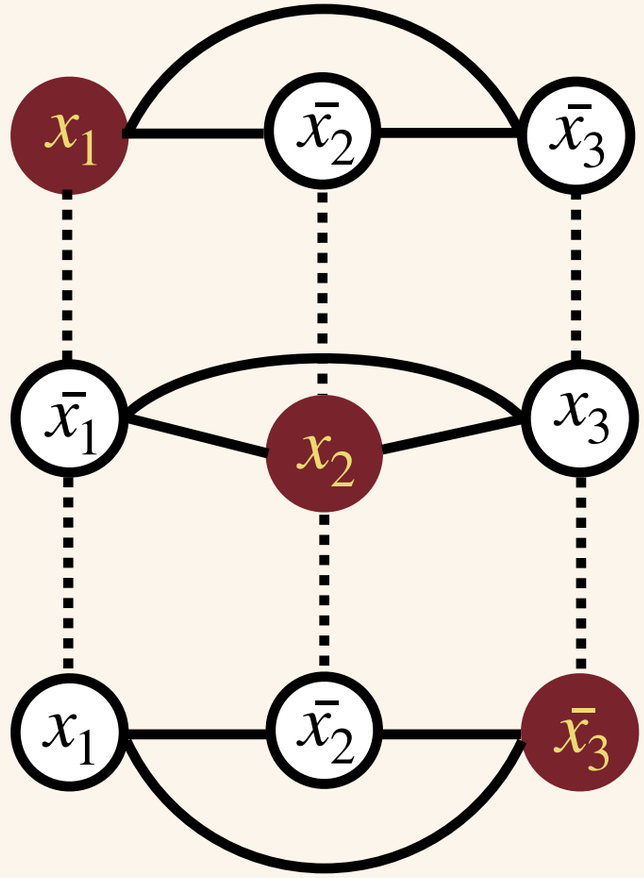
Example. Black vertices form an independent set of size 5

Task. Show that `IS` is NP-Complete.

1. `IS` is in NP.
2. `SAT` poly-time reduces to `IS`.

- Create a node for each literal in each clause.
- Connect each node to the literals in the same clause.
- Connect each literal to its negation.
- The expression is satisfiable iff there is an independent set of size = the number of clauses.

The independent set contains only 1 vertex from each group. Two vertices in the same group can't be in the set and a variable and its negation can't be in the set (because these are connected with edges in the constructed graph)

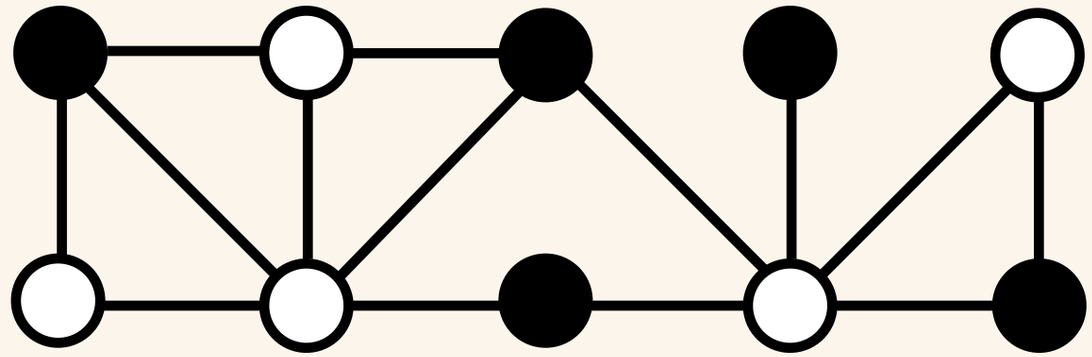


Example. $(x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$

NP-Completeness (Proof Examples)

`INDEPENDENT-SET (IS)`

Given a graph and an integer k, is there a subset of k vertices such that no two vertices are adjacent?



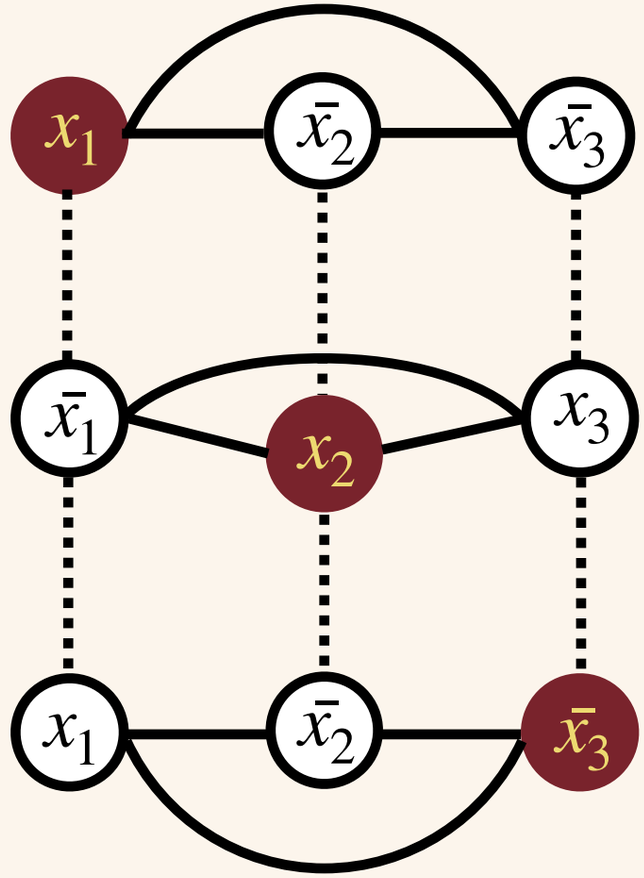
Example. Black vertices form an independent set of size 5

Task. Show that `IS` is NP-Complete.

1. `IS` is in NP.
2. `SAT` poly-time reduces to `IS`.

- Create a node for each literal in each clause.
- Connect each node to the literals in the same clause.
- Connect each literal to its negation.
- The expression is satisfiable iff there is an independent set of size = the number of clauses.

If the number of literals = N, the construction takes $O(N^2)$ time (i.e. this is a polytime reduction)

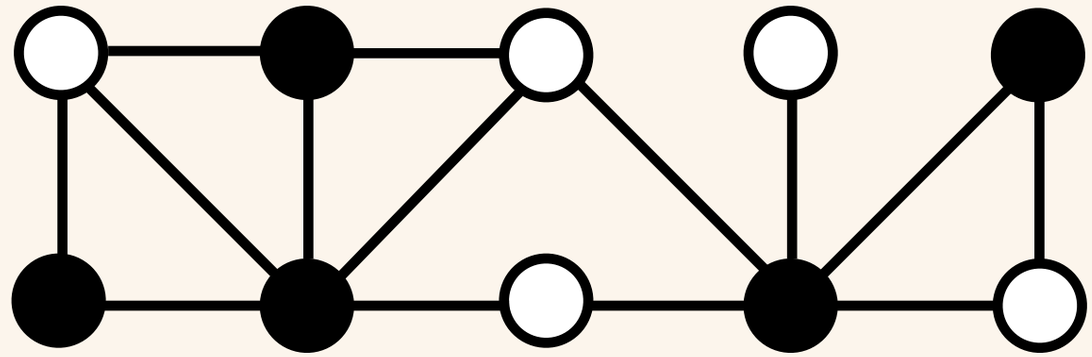


Example. $(x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$

NP-Completeness (Proof Examples)

VERTEX-COVER (VC)

Given a graph and an integer k, is there a subset of k vertices such that each edge is incident to at least one vertex in the subset?



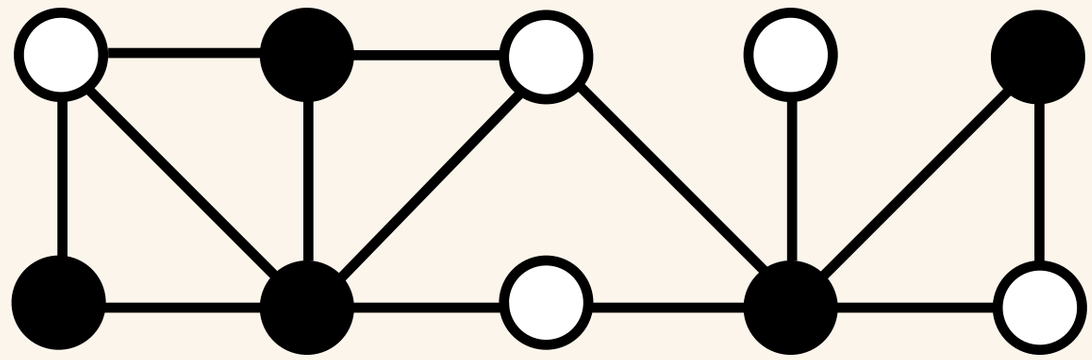
Task. Show that VC is NP-Complete.

Example. Black vertices form a vertex cover of size 5

NP-Completeness (Proof Examples)

VERTEX-COVER (VC)

Given a graph and an integer k, is there a subset of k vertices such that each edge is incident to at least one vertex in the subset?



Example. Black vertices form a vertex cover of size 5

Task. Show that VC is NP-Complete.

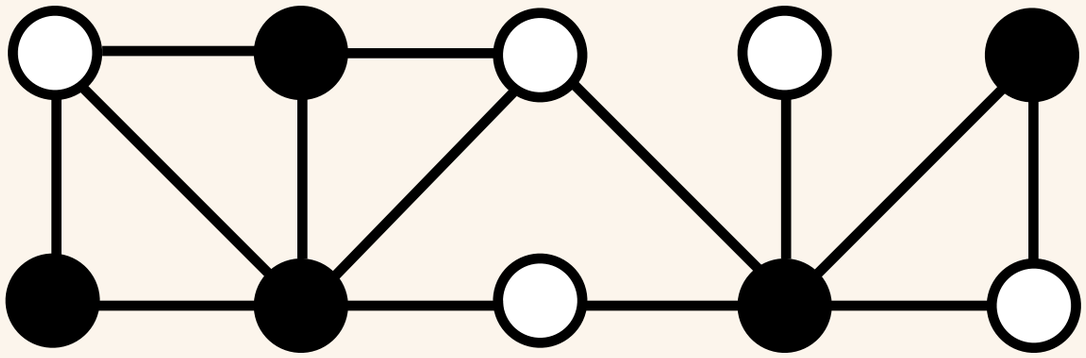
1. VC is in NP.

Given a set S of vertices in G, we can verify in polynomial time if each edge in the graph is incident to a vertex in S and if $|S| = k$.

NP-Completeness (Proof Examples)

VERTEX-COVER (VC)

Given a graph and an integer k, is there a subset of k vertices such that each edge is incident to at least one vertex in the subset?



Example. Black vertices form a vertex cover of size 5

Task. Show that VC is NP-Complete.

1. VC is in NP.

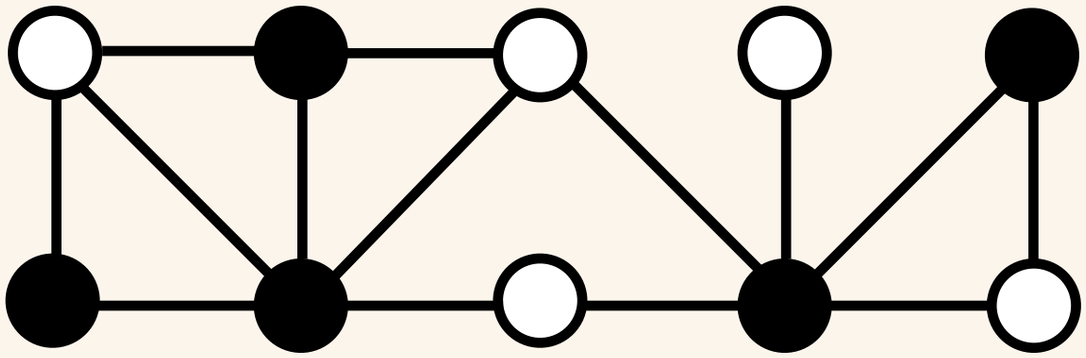
Given a set S of vertices in G, we can verify in polynomial time if each edge in the graph is incident to a vertex in S and if $|S| = k$.

2. INDEPENDENT-SET poly-time reduces to VERTEX-COVER.

NP-Completeness (Proof Examples)

`VERTEX-COVER (VC)`

Given a graph and an integer k, is there a subset of k vertices such that each edge is incident to at least one vertex in the subset?



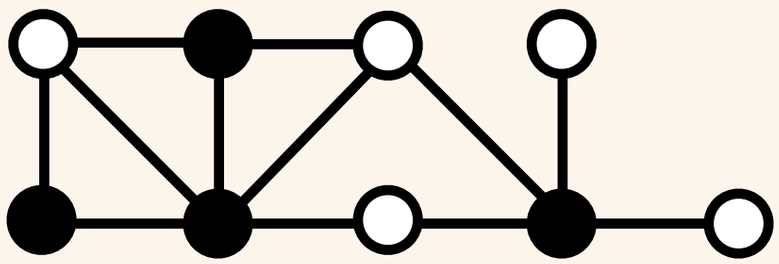
Example. Black vertices form a vertex cover of size 5

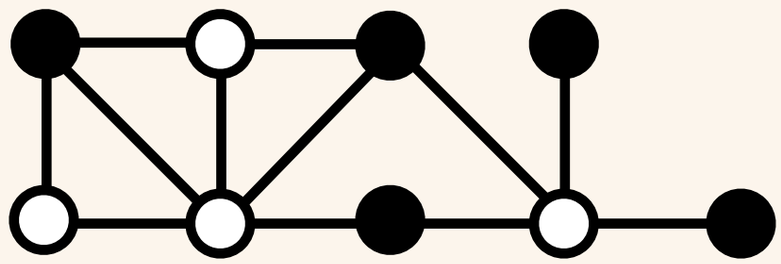
Task. Show that VC is NP-Complete.

1. VC is in NP.

Given a set S of vertices in G, we can verify in polynomial time if each edge in the graph is incident to a vertex in S and if $|S| = k$.

2. `INDEPENDENT-SET` poly-time reduces to `VERTEX-COVER`.

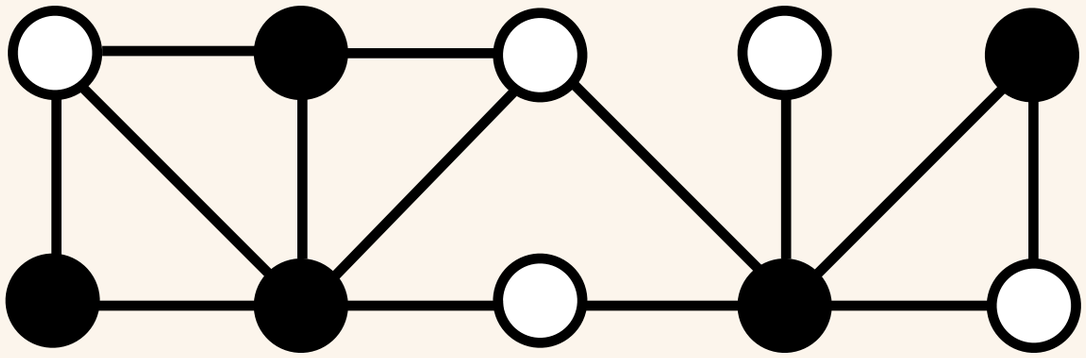




NP-Completeness (Proof Examples)

`VERTEX-COVER (VC)`

Given a graph and an integer k, is there a subset of k vertices such that each edge is incident to at least one vertex in the subset?



Example. Black vertices form a vertex cover of size 5

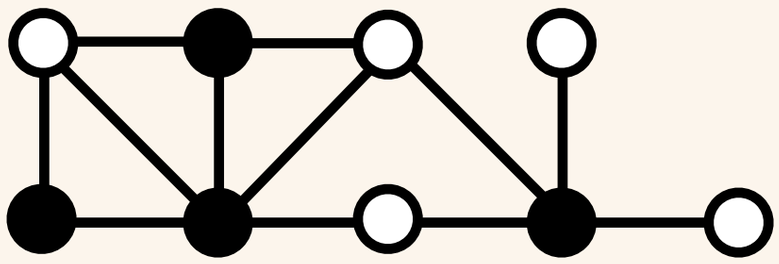
Task. Show that VC is NP-Complete.

1. VC is in NP.

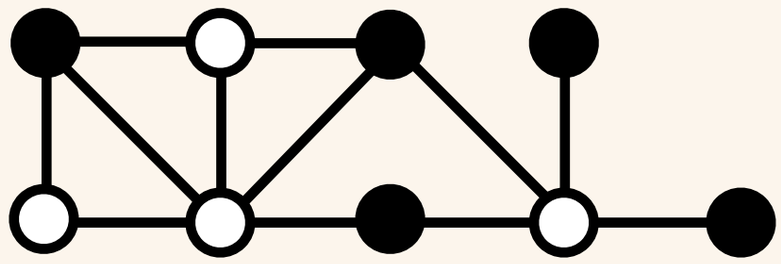
   Given a set S of vertices in G, we can verify in polynomial time if each edge in the graph is incident to a vertex in S and if $|S| = k$.

2. `INDEPENDENT-SET` poly-time reduces to `VERTEX-COVER`.

   S is an independent set of size k iff $V - S$ is a vertex cover of size $|V| - k$.



Vertex Cover of size 4



Independent Set of size 5

Vertex Cover and Independent Set

Claim. VERTEX-COVER $\equiv_P$ INDEPENDENT-SET.

Pf. We show S is an independent set iff V – S is a vertex cover.

$\Rightarrow$

- Let S be any independent set.
- Consider an arbitrary edge (u, v).
- S independent $\Rightarrow$ $u \notin S$ or $v \notin S$ $\Rightarrow$ $u \in V - S$ or $v \in V - S$.
- Thus, V – S covers (u, v).

$\Leftarrow$

- Let V – S be any vertex cover.
- Consider two nodes $u \in S$ and $v \in S$.
- Observe that $(u, v) \notin E$ since V – S is a vertex cover.
- Thus, no two nodes in S are joined by an edge $\Rightarrow$ S independent set. ▪

Slide by Kevin Wayne

NP-Completeness (Proof Examples)

TSP Given a complete weighted graph G, does G contain a simple circuit of *length* $\leq T$ that visits each node exactly once?

HAMILTONIAN Given a graph G, does G contain a simple circuit that visits each node only once.

Task. Show that TSP is NP-Complete knowing that HAMILTONIAN is NP-Complete.

1. Show that TSP is in NP. ← straight-forward

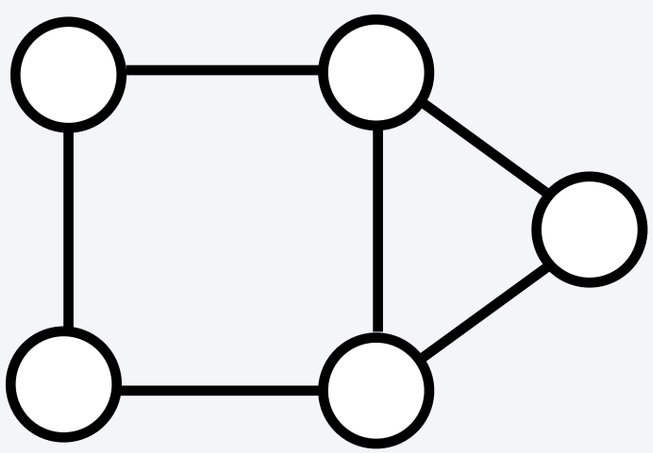
NP-Completeness (Proof Examples)

TSP Given a complete weighted graph G, does G contain a simple circuit of *length* $\leq T$ that visits each node exactly once?

HAMILTONIAN Given a graph G, does G contain a simple circuit that visits each node only once.

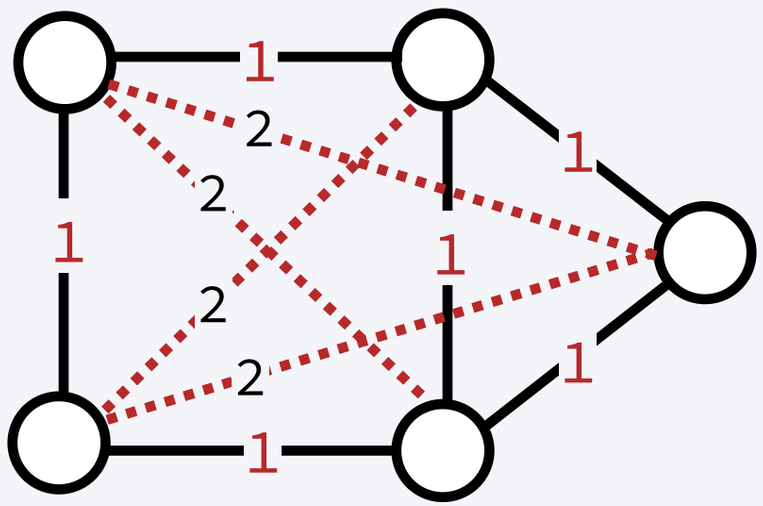
Task. Show that TSP is NP-Complete knowing that HAMILTONIAN is NP-Complete.

1. Show that TSP is in NP. ← straight-forward
2. HAMILTONIAN poly-time reduces to TSP.



G

Input to HAMILTONIAN



G'

Input to TSP

G has a hamiltonian cycle iff G' has a tour of length V

(If length > V then an edge that is not part of the original graph was used)

Add edge (u, v) with weight **1** if (u, v) is in G.
Add edge (u, v) with weight **2** if (u, v) is not in G.

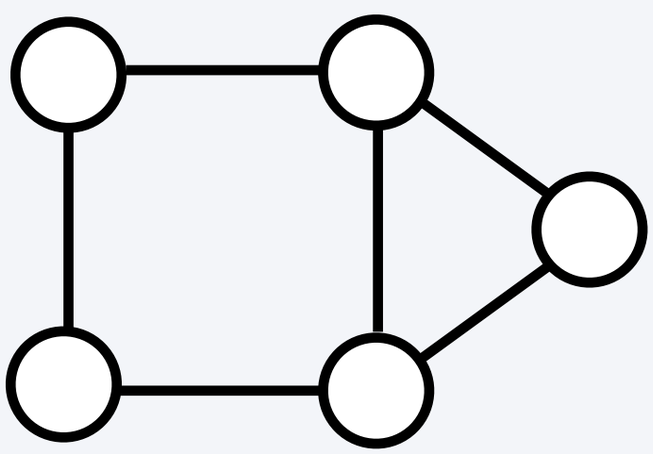
NP-Completeness (Proof Examples)

TSP Given a complete weighted graph G, does G contain a simple circuit of *length* $\leq T$ that visits each node exactly once?

HAMILTONIAN Given a graph G, does G contain a simple circuit that visits each node only once.

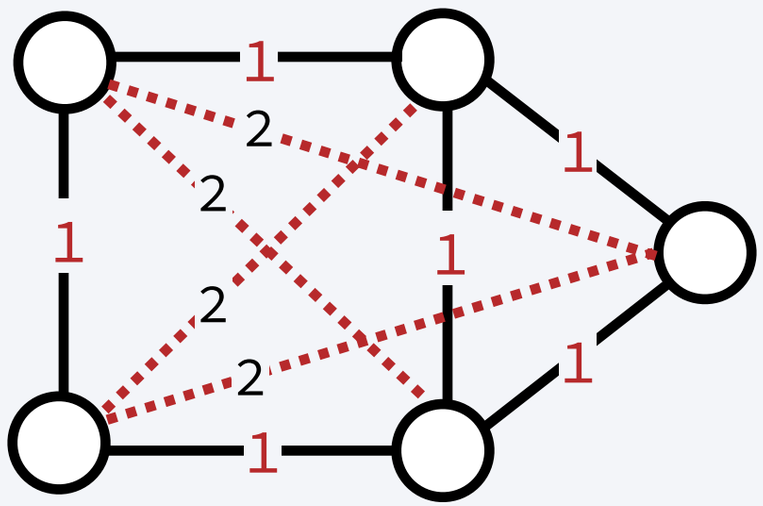
Task. Show that TSP is NP-Complete knowing that HAMILTONIAN is NP-Complete.

1. Show that TSP is in NP. ← straight-forward
2. HAMILTONIAN poly-time reduces to TSP.



G

Input to HAMILTONIAN



G'

Input to TSP

G has a hamiltonian cycle iff G' has a tour of length V

(If length > V then an edge that is not part of the original graph was used)

This construction runs in $\Theta(V^2)$ = polytime reduction!

Add edge (u, v) with weight **1** if (u, v) is in G.
Add edge (u, v) with weight **2** if (u, v) is not in G.

Definitions (Complexity Classes)

Class **P**.

A decision problem is in P if it is solvable in polynomial time
(i.e. in $O(n^c)$, where n is the input size and c is a constant)

Class **NP**.

A decision problem is in NP if it is verifiable in polynomial time.
(Given an instance I or a problem P and a witness W for the solution, can we verify in polynomial time if W proves that the answer for I is yes?)

Class **NP-Complete**.

A decision problem is NP-Complete if:

- It is in NP.
- All problems in NP reduce to it in polynomial time.

Class **NP-Hard**.

A problem is NP-Hard if all problems in NP reduce to it in polynomial time.
(*at least as hard as the hardest problems in* NP)

Definitions (Complexity Classes)

Class **P**.

A decision problem is in P if it is solvable in polynomial time
(i.e. in $O(n^c)$, where n is the input size and c is a constant)

Class **NP**.

A decision problem is in NP if it is verifiable in polynomial time.
(Given an instance I or a problem P and a witness W for the solution, can we verify in polynomial time if W proves that the answer for I is yes?)

Class **NP-Complete**.

A decision problem is NP-Complete if:

- It is in NP.
- All problems in NP reduce to it in polynomial time.

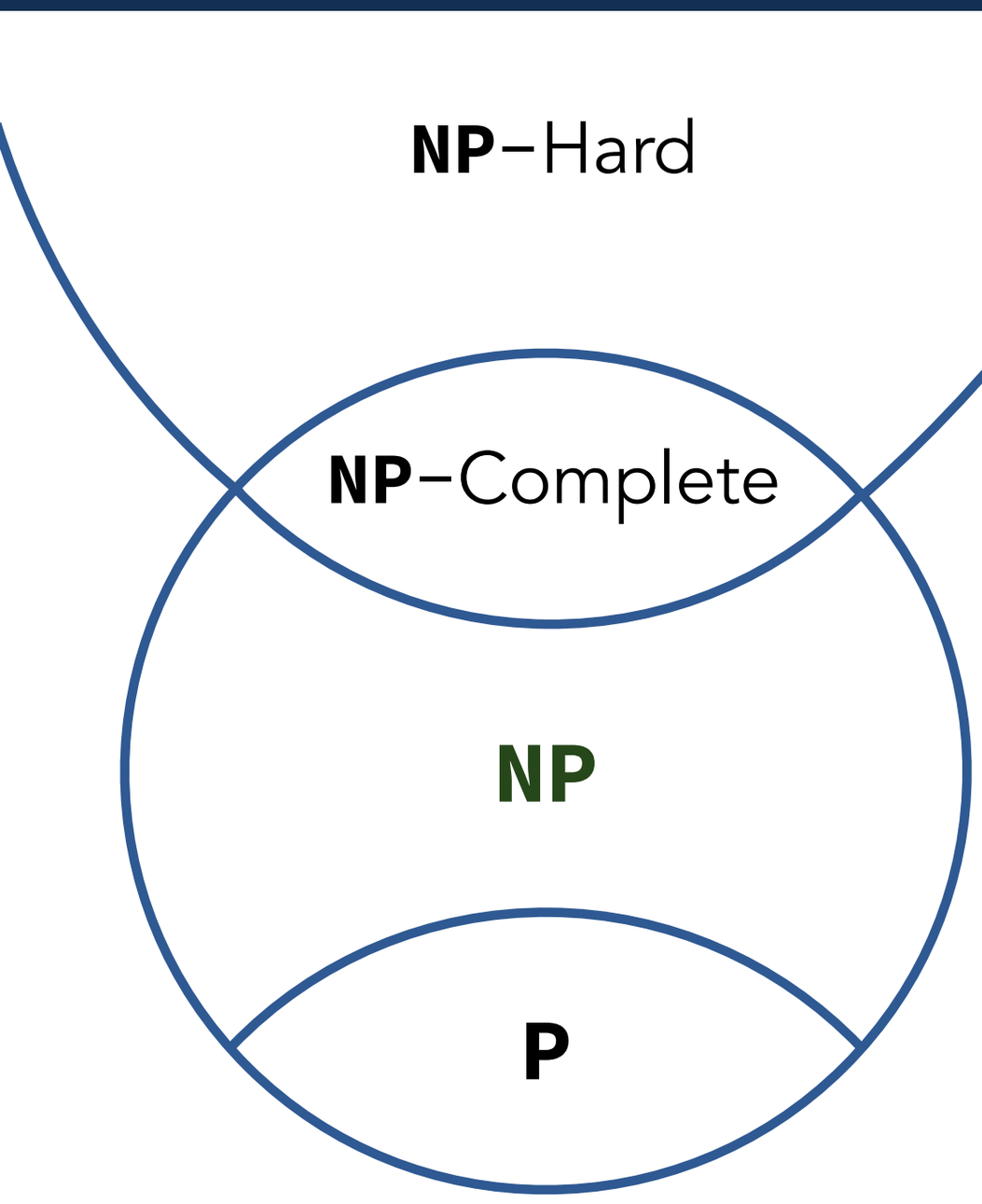
Examples.

- All NP-Complete Problems.
- TSP Optimization.
- Finding the Longest Simple Path.

Class **NP-Hard**.

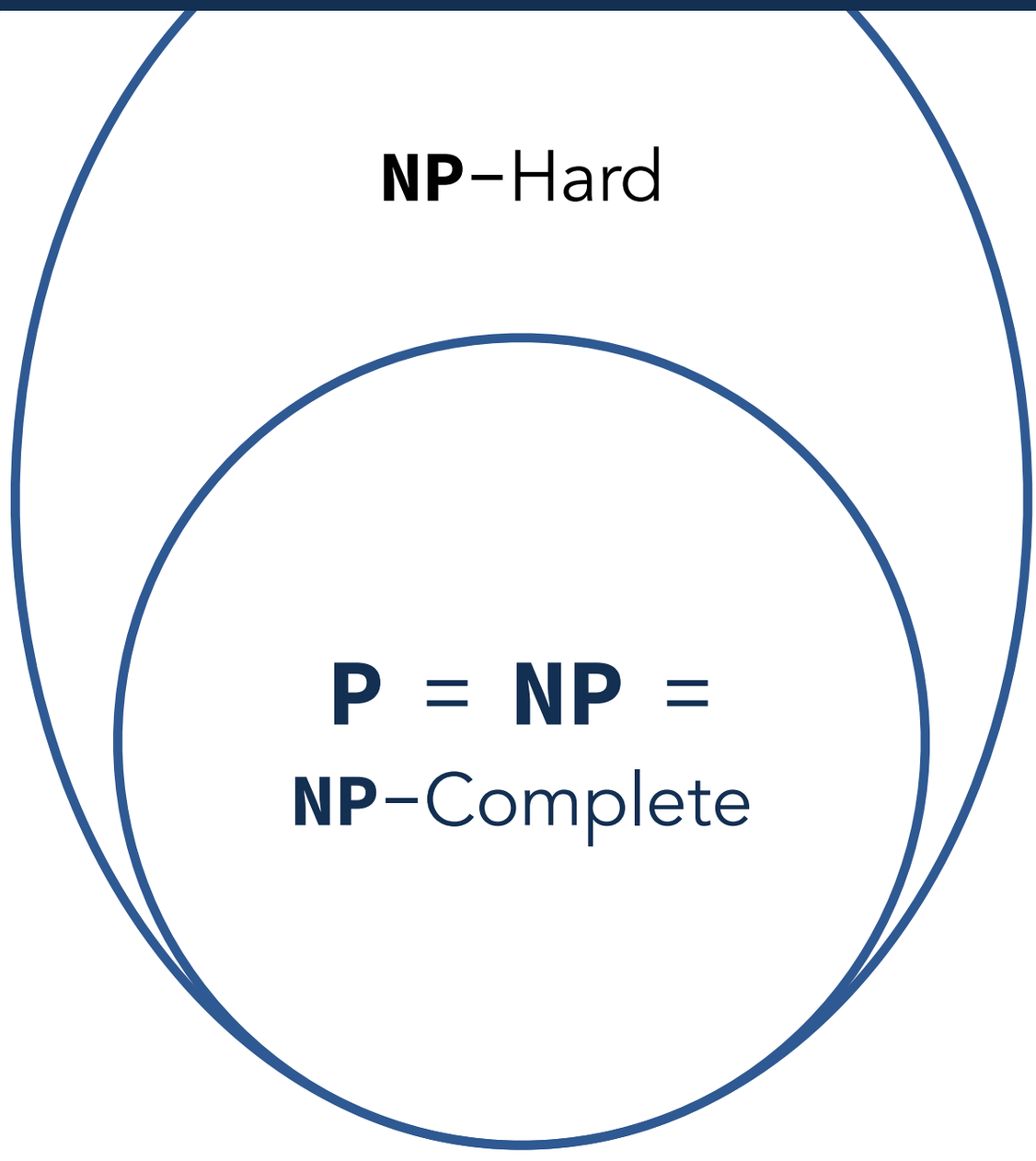
A problem is NP-Hard if all problems in NP reduce to it in polynomial time.
(*at least as hard as the hardest problems in* NP)

Two Possible World Views



If P ≠ NP

vs



If P = NP

Living with Intractability

When you encounter an NP-complete problem

- It is safe to assume that it is intractable. ← does not have an algorithm that solve all instances in polynomial time.
- What to do?

Four successful approaches

- Don't try to solve intractable problems.
- Try to solve real-world problem instances.
- Look for approximate solutions (not discussed in this lecture).
- Exploit intractability.

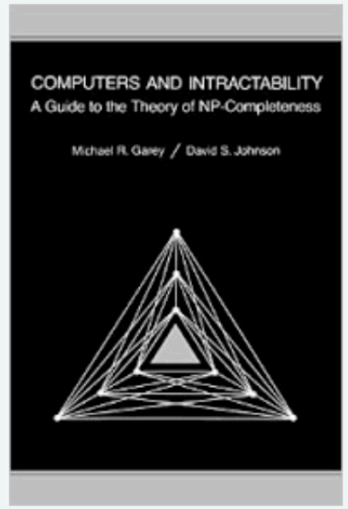
slide by Kevin Wayne

Living with Intractability: Don't Try To Solve It!

Knows no theory

I can't find an efficient algorithm.

I guess I'm just to dumb.



Knows computability

I can't find an efficient algorithm,
because no such algorithm is possible!

Knows intractability

I can't find an efficient algorithm,
but neither can all these famous people!

slide by Kevin Wayne

Living with Intractability: Solve Real-World Instances

Observations

- Worst-case inputs may not occur for practical problems.
- Instances that do occur in practice may be easier to solve.

Reasonable approach: relax the condition of *guaranteed* poly-time algorithms.

SAT

- *Chaff* solves real-world instances with 10,000+ variables.
- Princeton senior independent work (!) in 2000.

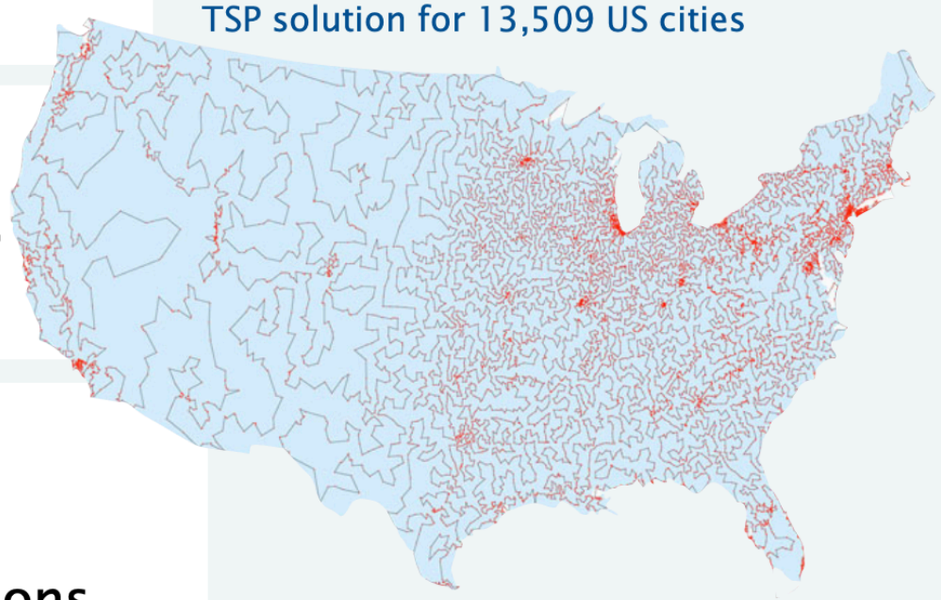
TSP

- *Concorde* routinely solves large real-world instances.
- 85,900-city instance solved in 2006.

ILP

- *CPLEX* routinely solves large real-world instances.
- Routinely used in scientific and commercial applications.

TSP solution for 13,509 US cities



slide by Kevin Wayne