NP-Complete Problems

T. M. Murali

April 21, 28, 2021
Review: Definitions of $\mathcal{NP}$-Complete and $\mathcal{NP}$-Hard

A problem $X$ is $\mathcal{NP}$-Complete if

(i) $X \in \mathcal{NP}$ and

(ii) for every problem $Y \in \mathcal{NP}$, $Y \leq^P X$.

A problem $X$ is $\mathcal{NP}$-Hard if

(i) for every problem $Y \in \mathcal{NP}$, $Y \leq^P X$.

![Diagram showing $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NPc}$, and $\mathcal{NP}$-hard relationships]
Claim: If $Y$ is $\mathcal{NP}$-Complete and $X \in \mathcal{NP}$ such that $Y \leq_P X$, then $X$ is $\mathcal{NP}$-Complete.
Claim: If $Y$ is $\mathcal{NP}$-Complete and $X \in \mathcal{NP}$ such that $Y \leq_P X$, then $X$ is $\mathcal{NP}$-Complete.

Given a new problem $X$, a general strategy for proving it $\mathcal{NP}$-Complete is
Claim: If $Y$ is $\mathcal{NP}$-Complete and $X \in \mathcal{NP}$ such that $Y \leq_P X$, then $X$ is $\mathcal{NP}$-Complete.

Given a new problem $X$, a general strategy for proving it $\mathcal{NP}$-Complete is

1. Prove that $X \in \mathcal{NP}$.
2. Select a problem $Y$ known to be $\mathcal{NP}$-Complete.
3. Prove that $Y \leq_P X$. 
Claim: If $Y$ is $\mathcal{NP}$-Complete and $X \in \mathcal{NP}$ such that $Y \leq_P X$, then $X$ is $\mathcal{NP}$-Complete.

Given a new problem $X$, a general strategy for proving it $\mathcal{NP}$-Complete is:

1. Prove that $X \in \mathcal{NP}$.
2. Select a problem $Y$ known to be $\mathcal{NP}$-Complete.
3. Prove that $Y \leq_P X$.

To prove $X$ is $\mathcal{NP}$-Complete, reduce a known $\mathcal{NP}$-Complete problem $Y$ to $X$. Do not prove reduction in the opposite direction, i.e., $X \leq_P Y$. 
Proving a Problem $\mathcal{NP}$-Complete with Karp Reduction

1. Prove that $X \in \mathcal{NP}$.

2. Select a problem $Y$ known to be $\mathcal{NP}$-Complete.

3. Consider an arbitrary input $s$ to problem $Y$. Show how to construct, in polynomial time, an input $t$ to problem $X$ such that
   
   (a) If $Y(s) = \text{yes}$, then $X(t) = \text{yes}$ and
   (b) If $X(t) = \text{yes}$, then $Y(s) = \text{yes}$ (equivalently, if $Y(s) = \text{no}$, then $X(t) = \text{no}$).
3-SAT is $\mathcal{NP}$-Complete

- Why is 3-SAT in NP?
3-SAT is \( \mathcal{NP} \)-Complete

- Why is 3-SAT in NP?
- Circuit Satisfiability \( \leq_P \) 3-SAT.
  1. Given an input to Circuit Satisfiability, create an input to SAT, in which each clause has at most three variables.
  2. Convert this input to SAT into an input to 3-SAT.

\[\text{Figure 8.4} \text{ A circuit with three inputs, two additional sources that have assigned truth values, and one output.}\]
Circuit Satisfiability $\leq_P$ 3-SAT: Transformation

- Given an arbitrary circuit $K$, associate each node $v$ with a Boolean variable $x_v$.
- Encode the requirements of each gate as a clause.
Circuit Satisfiability $\leq_P$ 3-SAT: Transformation

- Given an arbitrary circuit $K$, associate each node $v$ with a Boolean variable $x_v$.
- Encode the requirements of each gate as a clause.
- node $v$ has $\neg$ and edge entering from node $u$: guarantee that $x_v = \overline{x_u}$ using clauses.
Circuit Satisfiability $\leq_P$ 3-SAT: Transformation

- Given an arbitrary circuit $K$, associate each node $v$ with a Boolean variable $x_v$.
- Encode the requirements of each gate as a clause.
- Node $v$ has $\neg$ and edge entering from node $u$: guarantee that $x_v = \overline{x_u}$ using clauses $(x_v \lor x_u)$ and $(\overline{x_v} \lor \overline{x_u})$. 

Constants at sources: single-variable clauses.
Output: if $o$ is the output node, use the clause $(x_o)$.
Circuit Satisfiability $\leq_P$ 3-SAT: Transformation

- Given an arbitrary circuit $K$, associate each node $v$ with a Boolean variable $x_v$.
- Encode the requirements of each gate as a clause.
- Node $v$ has $\neg$ and edge entering from node $u$: guarantee that $x_v = \overline{x_u}$ using clauses $(x_v \lor x_u)$ and $(\overline{x_v} \lor \overline{x_u})$.
- Node $v$ has $\lor$ and edges entering from nodes $u$ and $w$: ensure $x_v = x_u \lor x_w$ using clauses.
Circuit Satisfiability $\leq_P$ 3-SAT: Transformation

- Given an arbitrary circuit $K$, associate each node $v$ with a Boolean variable $x_v$.
- Encode the requirements of each gate as a clause.
- node $v$ has $\neg$ and edge entering from node $u$: guarantee that $x_v = \overline{x_u}$ using clauses $(x_v \lor x_u)$ and $(\overline{x_v} \lor \overline{x_u})$.
- node $v$ has $\lor$ and edges entering from nodes $u$ and $w$: ensure $x_v = x_u \lor x_w$ using clauses $(x_v \lor \overline{x_u})$, $(x_v \lor \overline{x_w})$, and $(\overline{x_v} \lor x_u \lor x_w)$. 

Constants at sources: single-variable clauses. Output: if $o$ is the output node, use the clause $(x_o)$. 
Circuit Satisfiability $\leq_P$ 3-SAT: Transformation

- Given an arbitrary circuit $K$, associate each node $v$ with a Boolean variable $x_v$.
- Encode the requirements of each gate as a clause.
- node $v$ has $\neg$ and edge entering from node $u$: guarantee that $x_v = \overline{x_u}$ using clauses $(x_v \lor x_u)$ and $(\overline{x_v} \lor \overline{x_u})$.
- node $v$ has $\lor$ and edges entering from nodes $u$ and $w$: ensure $x_v = x_u \lor x_w$ using clauses $(x_v \lor \overline{x_u})$, $(x_v \lor \overline{x_w})$, and $(\overline{x_v} \lor x_u \lor x_w)$.
- node $v$ has $\land$ and edges entering from nodes $u$ and $w$: ensure $x_v = x_u \land x_w$ using clauses
Strategy

3-SAT

Sequencing Problems

Partitioning Problems

Other Problems

Circuit Satisfiability $\leq_P$ 3-SAT: Transformation

- Given an arbitrary circuit $K$, associate each node $v$ with a Boolean variable $x_v$.
- Encode the requirements of each gate as a clause.
- node $v$ has $\neg$ and edge entering from node $u$: guarantee that $x_v = \overline{x_u}$ using clauses $(x_v \lor x_u)$ and $(\overline{x_v} \lor \overline{x_u})$.
- node $v$ has $\lor$ and edges entering from nodes $u$ and $w$: ensure $x_v = x_u \lor x_w$ using clauses $(x_v \lor \overline{x_u})$, $(x_v \lor \overline{x_w})$, and $(\overline{x_v} \lor x_u \lor x_w)$.
- node $v$ has $\land$ and edges entering from nodes $u$ and $w$: ensure $x_v = x_u \land x_w$ using clauses $(\overline{x_v} \lor x_u)$, $(\overline{x_v} \lor x_w)$, and $(x_v \lor \overline{x_u} \lor \overline{x_w})$.

Constants at sources: single-variable clauses.
Output: if $o$ is the output node, use the clause $(x_o)$. 
Circuit Satisfiability $\leq_P$ 3-SAT: Transformation

- Given an arbitrary circuit $K$, associate each node $v$ with a Boolean variable $x_v$.
- Encode the requirements of each gate as a clause.
- node $v$ has $\neg$ and edge entering from node $u$: guarantee that $x_v = \overline{x_u}$ using clauses $(x_v \lor x_u)$ and $(\overline{x_v} \lor \overline{x_u})$.
- node $v$ has $\lor$ and edges entering from nodes $u$ and $w$: ensure $x_v = x_u \lor x_w$ using clauses $(x_v \lor \overline{x_u})$, $(x_v \lor \overline{x_w})$, and $(\overline{x_v} \lor x_u \lor x_w)$.
- node $v$ has $\land$ and edges entering from nodes $u$ and $w$: ensure $x_v = x_u \land x_w$ using clauses $(\overline{x_v} \lor x_u)$, $(\overline{x_v} \lor x_w)$, and $(x_v \lor \overline{x_u} \lor \overline{x_w})$.
- Constants at sources: single-variable clauses.
Circuit Satisfiability $\leq_P$ 3-SAT: Transformation

- Given an arbitrary circuit $K$, associate each node $v$ with a Boolean variable $x_v$.
- Encode the requirements of each gate as a clause.
- Node $v$ has $\neg$ and edge entering from node $u$: guarantee that $x_v = \overline{x_u}$ using clauses $(x_v \lor x_u)$ and $(\overline{x_v} \lor \overline{x_u})$.
- Node $v$ has $\lor$ and edges entering from nodes $u$ and $w$: ensure $x_v = x_u \lor x_w$ using clauses $(x_v \lor \overline{x_u})$, $(x_v \lor \overline{x_w})$, and $(\overline{x_v} \lor x_u \lor x_w)$.
- Node $v$ has $\land$ and edges entering from nodes $u$ and $w$: ensure $x_v = x_u \land x_w$ using clauses $(\overline{x_v} \lor x_u)$, $(\overline{x_v} \lor x_w)$, and $(x_v \lor \overline{x_u} \lor \overline{x_w})$.
- Constants at sources: single-variable clauses.
- Output: if $o$ is the output node, use the clause $(x_o)$. 
Circuit Satisfiability $\leq_P$ 3-SAT: Proof

- Prove that $K$ is equivalent to the input to SAT.
  - $K$ is satisfiable $\rightarrow$ clauses are satisfiable.
Circuit Satisfiability $\leq_P$ 3-SAT: Proof

- Prove that $K$ is equivalent to the input to SAT.
  - $K$ is satisfiable $\rightarrow$ clauses are satisfiable.
  - clauses are satisfiable $\rightarrow K$ is satisfiable.
**Circuit Satisfiability \( \leq_P 3\text{-SAT} \): Proof**

- Prove that \( K \) is equivalent to the input to SAT.
  - \( K \) is satisfiable \( \rightarrow \) clauses are satisfiable.
  - Clauses are satisfiable \( \rightarrow \) \( K \) is satisfiable. Observe that we have constructed clauses so that the value assigned to a node’s variable is precisely what the circuit will compute.
Circuit Satisfiability \( \leq_P 3\text{-SAT} \): Proof

- Prove that \( K \) is equivalent to the input to SAT.
  - \( K \) is satisfiable \( \rightarrow \) clauses are satisfiable.
  - clauses are satisfiable \( \rightarrow \) \( K \) is satisfiable. Observe that we have constructed clauses so that the value assigned to a node’s variable is precisely what the circuit will compute.

- Converting input to SAT to an input to 3-SAT.
Circuit Satisfiability \( \leq_p \) 3-SAT: Proof

- Prove that \( K \) is equivalent to the input to \( \text{SAT} \).
  - \( K \) is satisfiable \( \rightarrow \) clauses are satisfiable.
  - clauses are satisfiable \( \rightarrow \) \( K \) is satisfiable. Observe that we have constructed clauses so that the value assigned to a node's variable is precisely what the circuit will compute.

- Converting input to \( \text{SAT} \) to an input to 3-SAT.
  - Create four new variables \( z_1, z_2, z_3, z_4 \) such that any satisfying assignment will have \( z_1 = z_2 = 0 \) by adding clauses
Circuit Satisfiability \( \leq_P 3\text{-SAT}: \) Proof

- Prove that \( K \) is equivalent to the input to \( \text{SAT} \).
  - \( K \) is satisfiable \( \rightarrow \) clauses are satisfiable.
  - clauses are satisfiable \( \rightarrow \) \( K \) is satisfiable. Observe that we have constructed clauses so that the value assigned to a node's variable is precisely what the circuit will compute.

- Converting input to \( \text{SAT} \) to an input to \( 3\text{-SAT} \).
  - Create four new variables \( z_1, z_2, z_3, z_4 \) such that any satisfying assignment will have \( z_1 = z_2 = 0 \) by adding clauses \((\overline{z_i} \lor z_3 \lor z_4), (\overline{z_i} \lor \overline{z_3} \lor z_4), (\overline{z_i} \lor z_3 \lor \overline{z_4}), \) and \((\overline{z_i} \lor \overline{z_3} \lor \overline{z_4})\), for \( i = 1 \) and \( i = 2 \).
Circuit Satisfiability $\leq_P$ 3-SAT: Proof

- Prove that $K$ is equivalent to the input to SAT.
  - $K$ is satisfiable $\rightarrow$ clauses are satisfiable.
  - clauses are satisfiable $\rightarrow$ $K$ is satisfiable. Observe that we have constructed clauses so that the value assigned to a node’s variable is precisely what the circuit will compute.

- Converting input to SAT to an input to 3-SAT.
  - Create four new variables $z_1, z_2, z_3, z_4$ such that any satisfying assignment will have $z_1 = z_2 = 0$ by adding clauses $(\overline{z_i} \lor z_3 \lor z_4)$, $(\overline{z_i} \lor \overline{z_3} \lor z_4)$, $(\overline{z_i} \lor z_3 \lor \overline{z_4})$, and $(\overline{z_i} \lor \overline{z_3} \lor \overline{z_4})$, for $i = 1$ and $i = 2$.
  - If a clause has a single term $t$, replace the clause with $(t \lor z_1 \lor z_2)$. 
Circuit Satisfiability $\leq_P$ 3-SAT: Proof

- Prove that $K$ is equivalent to the input to SAT.
  - $K$ is satisfiable $\rightarrow$ clauses are satisfiable.
  - clauses are satisfiable $\rightarrow$ $K$ is satisfiable. Observe that we have constructed clauses so that the value assigned to a node's variable is precisely what the circuit will compute.
- Converting input to SAT to an input to 3-SAT.
  - Create four new variables $z_1, z_2, z_3, z_4$ such that any satisfying assignment will have $z_1 = z_2 = 0$ by adding clauses $(\overline{z_i} \lor z_3 \lor z_4), (\overline{z_i} \lor \overline{z_3} \lor z_4), (\overline{z_i} \lor z_3 \lor \overline{z_4})$, and $(\overline{z_i} \lor \overline{z_3} \lor \overline{z_4})$, for $i = 1$ and $i = 2$.
  - If a clause has a single term $t$, replace the clause with $(t \lor z_1 \lor z_2)$.
  - If a clause has a two terms $t$ and $t'$, replace the clause with $t \lor t' \lor z_1$. 
More $\mathcal{NP}$-Complete problems

- **Circuit Satisfiability** is $\mathcal{NP}$-Complete.
- We just showed that **Circuit Satisfiability** $\leq_P$ 3-SAT.
- We know that

$$3\text{-SAT} \leq_P \text{Independent Set} \leq_P \text{Vertex Cover} \leq_P \text{Set Cover}$$

- All these problems are in $\mathcal{NP}$.
- Therefore, **Independent Set**, **Vertex Cover**, and **Set Cover** are $\mathcal{NP}$-Complete.
**Hamiltonian Cycle**

- Problems we have seen so far involve searching over subsets of a collection of objects.
- Another type of computationally hard problem involves searching over the set of all permutations of a collection of objects.
Hamiltonian Cycle

- Problems we have seen so far involve searching over subsets of a collection of objects.
- Another type of computationally hard problem involves searching over the set of all permutations of a collection of objects.
- In a directed graph $G(V,E)$, a cycle $C$ is a *Hamiltonian cycle* if $C$ visits each vertex exactly once.

**Hamiltonian Cycle**

**INSTANCE:** A directed graph $G$.

**QUESTION:** Does $G$ contain a Hamiltonian cycle?
Hamiltonian Cycle

- Problems we have seen so far involve searching over subsets of a collection of objects.
- Another type of computationally hard problem involves searching over the set of all permutations of a collection of objects.
- In a directed graph $G(V, E)$, a cycle $C$ is a *Hamiltonian cycle* if $C$ visits each vertex exactly once.

**Hamiltonian Cycle**

**INSTANCE:** A directed graph $G$.

**QUESTION:** Does $G$ contain a Hamiltonian cycle?
Hamiltonian Cycle is $\mathcal{NP}$-Complete

- Why is the problem in $\mathcal{NP}$?
Hamiltonian Cycle is $\mathcal{NP}$-Complete

- Why is the problem in $\mathcal{NP}$?
- Claim: $3$-SAT $\leq_P$ HAMILTONIAN CYCLE.
Hamiltonian Cycle is $\mathcal{NP}$-Complete

- Why is the problem in $\mathcal{NP}$?
- Claim: $3$-SAT $\leq_P$ Hamiltonian Cycle. ▶ Jump to TSP
- Consider an arbitrary input to $3$-SAT with variables $x_1, x_2, \ldots, x_n$ and clauses $C_1, C_2, \ldots, C_k$.
- Strategy:
  1. Construct a graph $G$ with $O(nk)$ nodes and edges and $2^n$ Hamiltonian cycles with a one-to-one correspondence with $2^n$ truth assignments.
  2. Add nodes to impose constraints arising from clauses.
  3. Construction takes $O(nk)$ time.
- $G$ contains $n$ paths $P_1, P_2, \ldots, P_n$, one for each variable.
- Each $P_i$ contains $b = 3k + 3$ nodes $v_{i,1}, v_{i,2}, \ldots, v_{i,b}$, three for each clause and some extra nodes.
3-SAT $\leq_P$ Hamiltonian Cycle: Constructing $G$
3-SAT $\leq_P$ Hamiltonian Cycle: Modelling clauses

- Consider the clause $C_1 = x_1 \lor \overline{x}_2 \lor x_3$. 
3-SAT $\leq_P$ Hamiltonian Cycle: Modelling clauses

- Consider the clause $C_1 = x_1 \lor \overline{x_2} \lor x_3$. 
Example

- Two clauses $C_1 = x_1 \lor \overline{x_2}$, $C_2 = x_1 \lor x_2$. 

Diagram:

- Starting node $S$.
- Path $P_1$.
- Path $P_2$.
- Ending node $t$. 

NP-Complete Problems
Example

- Two clauses $C_1 = x_1 \lor \overline{x_2}$, $C_2 = x_1 \lor x_2$. 

Nodes for $c_1$  

Nodes for $c_2$
Example

- Two clauses $C_1 = x_1 \lor \overline{x_2}$, $C_2 = x_1 \lor x_2$.
Example

- Two clauses \( C_1 = x_1 \lor \overline{x_2} \), \( C_2 = x_1 \lor x_2 \).
Example

- Two clauses $C_1 = x_1 \lor \overline{x_2}$, $C_2 = x_1 \lor x_2$. 
Example

- Two clauses $C_1 = x_1 \lor \overline{x_2}$, $C_2 = x_1 \lor x_2$. 
3-SAT $\leq_P$ Hamiltonian Cycle: Proof Part 1

- 3-SAT input is satisfiable $\rightarrow$ $G$ has a Hamiltonian cycle.
3-SAT $\leq_P$ Hamiltonian Cycle: Proof Part 1

3-SAT input is satisfiable $\Rightarrow$ $G$ has a Hamiltonian cycle.

- Construct a Hamiltonian cycle $C$ as follows:
  - If $x_i = 1$, traverse $P_i$ from left to right in $C$.
  - Otherwise, traverse $P_i$ from right to left in $C$.
  - For each clause $C_j$, there is at least one term set to 1. If the term is $x_i$, splice $c_j$ into $C$ using edge from $v_{i,3j}$ and edge to $v_{i,3j+1}$. Analogous construction if term is $\overline{x_i}$.
3-SAT $\leq_p$ Hamiltonian Cycle: Proof Part 2

- $G$ has a Hamiltonian cycle $C \rightarrow$ Input to 3-SAT is satisfiable.
  - If $C$ enters $c_j$ on an edge from $v_{i,3j}$, it must leave $c_j$ along the edge to $v_{i,3j+1}$.
  - Analogous statement if $C$ enters $c_j$ on an edge from $v_{i,3j+1}$. 
3-SAT \(\leq_P\) Hamiltonian Cycle: Proof Part 2

- \(G\) has a Hamiltonian cycle \(C\) → Input to 3-SAT is satisfiable.
  - If \(C\) enters \(c_j\) on an edge from \(v_{i,3j}\), it must leave \(c_j\) along the edge to \(v_{i,3j+1}\).
  - Analogous statement if \(C\) enters \(c_j\) on an edge from \(v_{i,3j+1}\).
  - Nodes immediately before and after \(c_j\) in \(C\) are themselves connected by an edge \(e\) in \(G\).
3-SAT $\leq_P$ Hamiltonian Cycle: Proof Part 2

- $G$ has a Hamiltonian cycle $C \to$ Input to 3-SAT is satisfiable.
  - If $C$ enters $c_j$ on an edge from $v_{i,3j}$, it must leave $c_j$ along the edge to $v_{i,3j+1}$.
  - Analogous statement if $C$ enters $c_j$ on an edge from $v_{i,3j+1}$.
  - Nodes immediately before and after $c_j$ in $C$ are themselves connected by an edge $e$ in $G$.
  - If we remove all such edges $e$ from $C$, we get a Hamiltonian cycle $C'$ in $G - \{c_1, c_2, \ldots, c_k\}$.
  - Use $C'$ to construct truth assignment to variables; prove assignment is satisfying.
The Traveling Salesman Problem

- A salesman must visit $n$ cities $v_1, v_2, \ldots v_n$ starting at home city $v_1$.
- Salesman must find a tour, an order in which to visit each city exactly once, and return home.
- Goal is to find as short a tour as possible.
The Traveling Salesman Problem

- A salesman must visit \( n \) cities \( v_1, v_2, \ldots v_n \) starting at home city \( v_1 \).
- Salesman must find a *tour*, an order in which to visit each city exactly once, and return home.
- Goal is to find as short a tour as possible.
- For every pair of cities \( v_i \) and \( v_j \), \( d(v_i, v_j) > 0 \) is the distance from \( v_i \) to \( v_j \).
- A *tour* is a permutation \( v_{i_1} = v_1, v_{i_2}, \ldots v_{i_n} \).
- The *length* of the tour is \( \sum_{j=1}^{n-1} d(v_{i_j} v_{i_{j+1}}) + d(v_{i_n}, v_{i_1}) \).
The Traveling Salesman Problem

- A salesman must visit $n$ cities $v_1, v_2, \ldots v_n$ starting at home city $v_1$.
- Salesman must find a *tour*, an order in which to visit each city exactly once, and return home.
- Goal is to find as short a tour as possible.
- For every pair of cities $v_i$ and $v_j$, $d(v_i, v_j) > 0$ is the distance from $v_i$ to $v_j$.
- A *tour* is a permutation $v_{i_1} = v_1, v_{i_2}, \ldots v_{i_n}$.
- The *length* of the tour is $\sum_{j=1}^{n-1} d(v_{i_j}v_{i_{j+1}}) + d(v_{i_n}, v_{i_1})$.

**Travelling Salesman**

**INSTANCE:** A set $V$ of $n$ cities, a function $d : V \times V \rightarrow \mathbb{R}^+$, and a number $D > 0$.

**QUESTION:** Is there a tour of length at most $D$?
Examples of Travelling Salesman

(1977) 120 cities, Groetschel
Images taken from http://tsp.gatech.edu
Examples of Travelling Salesman

(1987) 532 AT&T switch locations, Padberg and Rinaldi
Images taken from http://tsp.gatech.edu
Examples of Travelling Salesman

(1987) 13,509 cities with population \( \geq 500 \), Applegate, Bixby, Chvátal, and Cook
Images taken from http://tsp.gatech.edu
Examples of Travelling Salesman

(2001) 15,112 cities, Applegate, Bixby, Chvátal, and Cook
Images taken from http://tsp.gatech.edu
Examples of Travelling Salesman

(2004) 24978, cities, Applegate, Bixby, Chvátal, Cook, and Helsgaun
Images taken from http://tsp.gatech.edu
Travelling Salesman is \( \mathcal{NP} \)-Complete

- Why is the problem in \( \mathcal{NP} \)?
- Why is the problem \( \mathcal{NP} \)-Complete?
Travelling Salesman is \( \mathcal{NP} \)-Complete

- Why is the problem in \( \mathcal{NP} \)?
- Why is the problem \( \mathcal{NP} \)-Complete?
- Claim: \textsc{Hamiltonian Cycle} \( \leq_p \textsc{Travelling Salesman} \).
**Travelling Salesman is \( \mathcal{NP} \)-Complete**

- Why is the problem in \( \mathcal{NP} \)?
- Why is the problem \( \mathcal{NP} \)-Complete?
- **Claim:** **Hamiltonian Cycle** \( \leq_{p} \) **Travelling Salesman.**

<table>
<thead>
<tr>
<th>Hamiltonian Cycle</th>
<th>Travelling Salesman</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directed graph ( G(V, E) )</td>
<td>Cities</td>
</tr>
<tr>
<td>Edges have identical weights</td>
<td>Distances between cities can vary</td>
</tr>
<tr>
<td>Not all pairs of nodes are connected in ( G )</td>
<td>Every pair of cities has a distance</td>
</tr>
<tr>
<td>((u, v)) and ((v, u)) may both be edges</td>
<td>( d(v_i, v_j) \neq d(v_j, v_i) ), in general</td>
</tr>
<tr>
<td>Does a cycle exist?</td>
<td>Does a tour of length ( \leq D ) exist?</td>
</tr>
</tbody>
</table>
Travelling Salesman is $\mathcal{NP}$-Complete

- Why is the problem in $\mathcal{NP}$?
- Why is the problem $\mathcal{NP}$-Complete?
- Claim: Hamiltonian Cycle $\leq_P$ Travelling Salesman.

<table>
<thead>
<tr>
<th>Hamiltonian Cycle</th>
<th>Travelling Salesman</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directed graph $G(V, E)$</td>
<td>Cities</td>
</tr>
<tr>
<td>Edges have identical weights</td>
<td>Distances between cities can vary</td>
</tr>
<tr>
<td>Not all pairs of nodes are connected in $G$</td>
<td>Every pair of cities has a distance</td>
</tr>
<tr>
<td>$(u, v)$ and $(v, u)$ may both be edges</td>
<td>$d(v_i, v_j) \neq d(v_j, v_i)$, in general</td>
</tr>
<tr>
<td>Does a cycle exist?</td>
<td>Does a tour of length $\leq D$ exist?</td>
</tr>
</tbody>
</table>

- Given a directed graph $G(V, E)$ (input to Hamiltonian Cycle),
  - Create a city $v_i$ for each node $i \in V$.
  - Define $d(v_i, v_j) = 1$ if $(i, j) \in E$.
  - Define $d(v_i, v_j) = 2$ if $(i, j) \notin E$. 
  
T. M. Murali  
April 21, 28, 2021  
NP-Complete Problems
Travelling Salesman is \( \mathcal{NP} \)-Complete

- Why is the problem in \( \mathcal{NP} \)?
- Why is the problem \( \mathcal{NP} \)-Complete?
- **Claim:** Hamiltonian Cycle \( \leq_P \) Travelling Salesman.

<table>
<thead>
<tr>
<th>Hamiltonian Cycle</th>
<th>Travelling Salesman</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directed graph ( G(V, E) )</td>
<td>Cities</td>
</tr>
<tr>
<td>Edges have identical weights</td>
<td>Distances between cities can vary</td>
</tr>
<tr>
<td>Not all pairs of nodes are connected in ( G )</td>
<td>Every pair of cities has a distance</td>
</tr>
<tr>
<td>((u, v)) and ((v, u)) may both be edges</td>
<td>(d(v_i, v_j) \neq d(v_j, v_i)), in general</td>
</tr>
<tr>
<td>Does a cycle exist?</td>
<td>Does a tour of length ( \leq D ) exist?</td>
</tr>
</tbody>
</table>

- Given a directed graph \( G(V, E) \) (input to Hamiltonian Cycle),
  - Create a city \( v_i \) for each node \( i \in V \).
  - Define \( d(v_i, v_j) = 1 \) if \((i, j) \in E\).
  - Define \( d(v_i, v_j) = 2 \) if \((i, j) \notin E\).

- **Claim:** \( G \) has a Hamiltonian cycle iff the input to Travelling Salesman has a tour of length at most \( D \).
Travelling Salesman is $\mathcal{NP}$-Complete

- Why is the problem in $\mathcal{NP}$?
- Why is the problem $\mathcal{NP}$-Complete?
- Claim: Hamiltonian Cycle $\leq_P$ Travelling Salesman.

<table>
<thead>
<tr>
<th>Hamiltonian Cycle</th>
<th>Travelling Salesman</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directed graph $G(V, E)$</td>
<td>Cities</td>
</tr>
<tr>
<td>Edges have identical weights</td>
<td>Distances between cities can vary</td>
</tr>
<tr>
<td>Not all pairs of nodes are connected in $G$</td>
<td>Every pair of cities has a distance</td>
</tr>
<tr>
<td>$(u, v)$ and $(v, u)$ may both be edges</td>
<td>$d(v_i, v_j) \neq d(v_j, v_i)$, in general</td>
</tr>
<tr>
<td>Does a cycle exist?</td>
<td>Does a tour of length $\leq D$ exist?</td>
</tr>
</tbody>
</table>

- Given a directed graph $G(V, E)$ (input to Hamiltonian Cycle),
  - Create a city $v_i$ for each node $i \in V$.
  - Define $d(v_i, v_j) = 1$ if $(i, j) \in E$.
  - Define $d(v_i, v_j) = 2$ if $(i, j) \not\in E$.

- Claim: $G$ has a Hamiltonian cycle iff the input to Travelling Salesman has a tour of length at most $n$. 
Special Cases and Extensions that are $NP$-Complete

- **Hamiltonian Cycle** for undirected graphs.
- **Hamiltonian Path** for directed and undirected graphs.
- **Travelling Salesman** with symmetric distances (by reducing Hamiltonian Cycle for undirected graphs to it).
- **Travelling Salesman** with distances defined by points on the plane.
2-Dimensional Matching

**Bipartite Matching**

**INSTANCE:** Disjoint sets $X$, $Y$, each of size $n$, and a set $T \subseteq X \times Y$ of pairs

**QUESTION:** Is there a set of $n$ pairs in $T$ such that each element of $X \cup Y$ is contained in exactly one of these pairs?
**3-Dimensional Matching**

**Bipartite Matching**

**INSTANCE:** Disjoint sets $X$, $Y$, each of size $n$, and a set $T \subseteq X \times Y$ of pairs

**QUESTION:** Is there a set of $n$ pairs in $T$ such that each element of $X \cup Y$ is contained in exactly one of these pairs?
3-Dimensional Matching

- **3-Dimensional Matching** is a harder version of **Bipartite Matching**.

**Bipartite Matching**

**INSTANCE:** Disjoint sets $X$, $Y$, each of size $n$, and a set $T \subseteq X \times Y$ of pairs.

**QUESTION:** Is there a set of $n$ pairs in $T$ such that each element of $X \cup Y$ is contained in exactly one of these pairs?
**3-Dimensional Matching**

- **3-Dimensional Matching is a harder version of Bipartite Matching.**

**3-Dimensional Matching**

**INSTANCE:** Disjoint sets $X$, $Y$, and $Z$, each of size $n$, and a set $T \subseteq X \times Y \times Z$ of triples

**QUESTION:** Is there a set of $n$ triples in $T$ such that each element of $X \cup Y \cup Z$ is contained in exactly one of these triples?
3-Dimensional Matching

- **3-Dimensional Matching** is a harder version of **Bipartite Matching**.

**INSTANCE:** Disjoint sets $X$, $Y$, and $Z$, each of size $n$, and a set $T \subseteq X \times Y \times Z$ of triples

**QUESTION:** Is there a set of $n$ triples in $T$ such that each element of $X \cup Y \cup Z$ is contained in exactly one of these triples?

- Easy to show $3$-**Dimensional Matching** $\leq_P$ **Set Cover** and $3$-**Dimensional Matching** $\leq_P$ **Set Packing**.
3-Dimensional Matching is \( \mathcal{NP} \)-Complete

- Why is the problem in \( \mathcal{NP} \)?
3-Dimensional Matching is \( \mathcal{NP} \)-Complete

- Why is the problem in \( \mathcal{NP} \)?
- Show that \( 3\text{-SAT} \leq_p 3\text{-DIMENSIONAL MATCHING} \).
- Strategy:
  - Start with an input to \( 3\text{-SAT} \) with \( n \) variables and \( k \) clauses.
  - Create a gadget for each variable \( x_i \) that encodes the choice of truth assignment to \( x_i \).
  - Add gadgets that encode constraints imposed by clauses.
3-SAT ≤ₚ 3-Dimensional Matching: Variables

- Each $x_i$ corresponds to a variable gadget $i$ with $2k$ core elements
  $A_i = \{a_{i,1}, a_{i,2}, \ldots a_{i,2k}\}$ and $2k$ tips
  $B_i = \{b_{i,1}, b_{i,2}, \ldots b_{i,2k}\}$.
- For each $1 \leq j \leq 2k$, variable gadget $i$ includes a triple $t_{ij} = (a_{i,j}, a_{i,j+1}, b_{i,j})$.
- A triple (tip) is even if $j$ is even. Otherwise, the triple (tip) is odd.
- Only these triples contain elements in $A_i$.

Figure 8.9 The reduction from 3-SAT to 3-Dimensional Matching.
3-SAT $\leq_P$ 3-Dimensional Matching: Variables

- Each $x_i$ corresponds to a *variable gadget* $i$ with $2k$ *core* elements $A_i = \{a_{i,1}, a_{i,2}, \ldots a_{i,2k}\}$ and $2k$ *tips* $B_i = \{b_{i,1}, b_{i,2}, \ldots b_{i,2k}\}$.
- For each $1 \leq j \leq 2k$, variable gadget $i$ includes a triple $t_{ij} = (a_{i,j}, a_{i,j+1}, b_{i,j})$.
- A triple (tip) is *even* if $j$ is even. Otherwise, the triple (tip) is *odd*.
- Only these triples contain elements in $A_i$.
- In any perfect matching, we can cover the elements in $A_i$.

Figure 8.9 The reduction from 3-SAT to 3-Dimensional Matching.
3-SAT $\leq_P$ 3-Dimensional Matching: Variables

- Each $x_i$ corresponds to a **variable gadget** $i$ with $2k$ **core** elements $A_i = \{a_{i,1}, a_{i,2}, \ldots a_{i,2k}\}$ and $2k$ **tips** $B_i = \{b_{i,1}, b_{i,2}, \ldots b_{i,2k}\}$.
- For each $1 \leq j \leq 2k$, variable gadget $i$ includes a triple $t_{ij} = (a_{i,j}, a_{i,j+1}, b_{i,j})$.
- A triple (tip) is **even** if $j$ is even. Otherwise, the triple (tip) is **odd**.
- Only these triples contain elements in $A_i$.

In any perfect matching, we can cover the elements in $A_i$ either using all the even triples in gadget $i$ or all the odd triples in the gadget.

- Even triples used, odd tips free $\equiv x_i = 0$; odd triples used, even tips free $\equiv x_i = 1$.

Figure 8.9 The reduction from 3-SAT to 3-Dimensional Matching.
3-SAT $\leq_p$ 3-Dimensional Matching: Clauses

- Consider the clause $C_1 = x_1 \lor \overline{x_2} \lor x_3$.

Figure 8.9 The reduction from 3-SAT to 3-Dimensional Matching.
**3-SAT \( \leq^P \) 3-Dimensional Matching: Clauses**

- Consider the clause \( C_1 = x_1 \lor \overline{x}_2 \lor x_3 \).
- \( C_1 \) says “The matching on the cores of the gadgets should leave the even tips of gadget 1 free; or it should leave the odd tips of gadget 2 free; or it should leave the even tips of gadget 3 free.”

*Figure 8.9 The reduction from 3-SAT to 3-Dimensional Matching.*
3-SAT \leq_P \text{3-Dimensional Matching: Clauses}

- Consider the clause $C_1 = x_1 \lor \overline{x_2} \lor x_3$.
- $C_1$ says “The matching on the cores of the gadgets should leave the even tips of gadget 1 free; or it should leave the odd tips of gadget 2 free; or it should leave the even tips of gadget 3 free.”

- **Clause gadget** $j$ for clause $C_j$ contains two core elements $P_j = \{p_j, p'_j\}$ and three triples:
  - $C_j$ contains $x_i$: add triple $(p_j, p'_j, b_{i,2j})$.
  - $C_j$ contains $\overline{x_i}$: add triple $(p_j, p'_j, b_{i,2j-1})$.

Figure 8.9 The reduction from 3-SAT to 3-Dimensional Matching.
3-SAT \leq_P 3-Dimensional Matching: Example

The clause elements can only be matched if some variable gadget leaves the corresponding tip free.

Figure 8.9 The reduction from 3-SAT to 3-Dimensional Matching.
3-SAT $\leq_P$ 3-Dimensional Matching: Proof

- Satisfying assignment $\rightarrow$ matching.
3-SAT \leq_p 3-Dimensional Matching: Proof

- Satisfying assignment \(\rightarrow\) matching.
  - Make appropriate choices for the core of each variable gadget.
  - At least one free tip available for each clause gadget, allowing core elements of clause gadgets to be covered.
3-SAT $\leq_P$ 3-Dimensional Matching: Proof

- Satisfying assignment $\rightarrow$ matching.
  - Make appropriate choices for the core of each variable gadget.
  - At least one free tip available for each clause gadget, allowing core elements of clause gadgets to be covered.
  - We have not covered all the tips!
3-SAT \( \leq_p \) 3-Dimensional Matching: Proof

- Satisfying assignment \( \rightarrow \) matching.
  - Make appropriate choices for the core of each variable gadget.
  - At least one free tip available for each clause gadget, allowing core elements of clause gadgets to be covered.
  - We have not covered all the tips!
  - Add \((n - 1)k\) cleanup gadgets to allow the remaining \((n - 1)k\) tips to be covered: cleanup gadget \(i\) contains two core elements \(Q = \{q_i, q'_i\}\) and triple \((q_i, q'_i, b)\) for every tip \(b\) in variable gadget \(i\).
3-SAT $\leq_P$ 3-Dimensional Matching: Proof

- Satisfying assignment $\rightarrow$ matching.
  - Make appropriate choices for the core of each variable gadget.
  - At least one free tip available for each clause gadget, allowing core elements of clause gadgets to be covered.
  - We have not covered all the tips!
  - Add $(n - 1)k$ cleanup gadgets to allow the remaining $(n - 1)k$ tips to be covered: cleanup gadget $i$ contains two core elements $Q = \{q_i, q'_i\}$ and triple $(q_i, q'_i, b)$ for every tip $b$ in variable gadget $i$.

- Matching $\rightarrow$ satisfying assignment.
3-SAT $\leq_p$ 3-Dimensional Matching: Proof

- Satisfying assignment $\rightarrow$ matching.
  - Make appropriate choices for the core of each variable gadget.
  - At least one free tip available for each clause gadget, allowing core elements of clause gadgets to be covered.
  - We have not covered all the tips!
  - Add $(n - 1)k$ cleanup gadgets to allow the remaining $(n - 1)k$ tips to be covered: cleanup gadget $i$ contains two core elements $Q = \{q_i, q'_i\}$ and triple $(q_i, q'_i, b)$ for every tip $b$ in variable gadget $i$.

- Matching $\rightarrow$ satisfying assignment.
  - Matching chooses all even $a_{ij}$ ($x_i = 0$) or all odd $a_{ij}$ ($x_i = 1$).
3-SAT $\leq_P$ 3-Dimensional Matching: Proof

- Satisfying assignment $\rightarrow$ matching.
  - Make appropriate choices for the core of each variable gadget.
  - At least one free tip available for each clause gadget, allowing core elements of clause gadgets to be covered.
  - We have not covered all the tips!
  - Add $(n - 1)k$ cleanup gadgets to allow the remaining $(n - 1)k$ tips to be covered: cleanup gadget $i$ contains two core elements $Q = \{q_i, q'_i\}$ and triple $(q_i, q'_i, b)$ for every tip $b$ in variable gadget $i$.

- Matching $\rightarrow$ satisfying assignment.
  - Matching chooses all even $a_{ij} (x_i = 0)$ or all odd $a_{ij} (x_i = 1)$.
  - Is clause $C_j$ satisfied?
3-SAT \( \leq_P \) 3-Dimensional Matching: Proof

- Satisfying assignment \( \rightarrow \) matching.
  - Make appropriate choices for the core of each variable gadget.
  - At least one free tip available for each clause gadget, allowing core elements of clause gadgets to be covered.
  - We have not covered all the tips!
  - Add \((n - 1)k\) cleanup gadgets to allow the remaining \((n - 1)k\) tips to be covered: cleanup gadget \(i\) contains two core elements \(Q = \{q_i, q'_i\}\) and triple \((q_i, q'_i, b)\) for every tip \(b\) in variable gadget \(i\).

- Matching \( \rightarrow \) satisfying assignment.
  - Matching chooses all even \(a_{ij}\) \((x_i = 0)\) or all odd \(a_{ij}\) \((x_i = 1)\).
  - Is clause \(C_j\) satisfied? Core in clause gadget \(j\) is covered by some triple \(\Rightarrow\) other element in the triple must be a tip element from the correct odd/even set in the three variable gadgets corresponding to a term in \(C_j\).
Did we create an input to 3-DIMENSIONAL MATCHING?
3-SAT $\leq_P$ 3-Dimensional Matching: Finale

- Did we create an input to 3-DIMENSIONAL MATCHING?
- We need three sets $X$, $Y$, and $Z$ of equal size.
3-SAT $\leq_p$ 3-Dimensional Matching: Finale

- Did we create an input to 3-Dimensional Matching?
- We need three sets $X$, $Y$, and $Z$ of equal size.
- How many elements do we have?
  - $2nk \ a_{ij}$ elements.
  - $2nk \ b_{ij}$ elements.
  - $k \ p_j$ elements.
  - $k \ p'_j$ elements.
  - $(n - 1)k \ q_i$ elements.
  - $(n - 1)k \ q'_i$ elements.
3-SAT \leq_P 3-Dimensional Matching: Finale

- Did we create an input to 3-DIMENSIONAL MATCHING?
- We need three sets \( X \), \( Y \), and \( Z \) of equal size.
- How many elements do we have?
  - \( 2nk \ a_{ij} \) elements.
  - \( 2nk \ b_{ij} \) elements.
  - \( k \ p_j \) elements.
  - \( k \ p'_j \) elements.
  - \( (n - 1)k \ q_i \) elements.
  - \( (n - 1)k \ q'_i \) elements.
- \( X \) is the union of \( a_{ij} \) with even \( j \), the set of all \( p_j \) and the set of all \( q_i \).
- \( Y \) is the union of \( a_{ij} \) with odd \( j \), the set if all \( p'_j \) and the set of all \( q'_i \).
- \( Z \) is the set of all \( b_{ij} \).
3-SAT \leq_p 3-Dimensional Matching: Finale

- Did we create an input to 3-DIMENSIONAL MATCHING?
- We need three sets \( X \), \( Y \), and \( Z \) of equal size.
- How many elements do we have?
  - \( 2nk \ a_{ij} \) elements.
  - \( 2nk \ b_{ij} \) elements.
  - \( k \ p_j \) elements.
  - \( k \ p_j' \) elements.
  - \( (n - 1)k \ q_i \) elements.
  - \( (n - 1)k \ q_i' \) elements.
- \( X \) is the union of \( a_{ij} \) with even \( j \), the set of all \( p_j \) and the set of all \( q_i \).
- \( Y \) is the union of \( a_{ij} \) with odd \( j \), the set if all \( p_j' \) and the set of all \( q_i' \).
- \( Z \) is the set of all \( b_{ij} \).
- Each triple contains exactly one element from \( X \), \( Y \), and \( Z \).
Any map can be coloured with four colours (Appel and Haken, 1976).

Colouring maps
Any map can be coloured with four colours (Appel and Hakken, 1976).
Given an undirected graph $G(V, E)$, a *k-colouring* of $G$ is a function $f : V \rightarrow \{1, 2, \ldots, k\}$ such that for every edge $(u, v) \in E$, $f(u) \neq f(v)$. 
Given an undirected graph $G(V, E)$, a $k$-colouring of $G$ is a function $f : V \rightarrow \{1, 2, \ldots, k\}$ such that for every edge $(u, v) \in E$, $f(u) \neq f(v)$.

**Graph Colouring ($k$-Colouring)**

**INSTANCE:** An undirected graph $G(V, E)$ and an integer $k > 0$.

**QUESTION:** Does $G$ have a $k$-colouring?
Applications of Graph Colouring

1. Job scheduling: assign jobs to $n$ processors under constraints that certain pairs of jobs cannot be scheduled at the same time.

2. Compiler design: assign variables to $k$ registers but two variables being used at the same time cannot be assigned to the same register.

3. Wavelength assignment: assign one of $k$ transmitting wavelengths to each of $n$ wireless devices. If two devices are close to each other, they must get different wavelengths.
2-Colouring

- How hard is 2-COLOURING?
2-Colouring

- How hard is 2-Colouring?
- Claim: A graph is 2-colourable if and only if it is bipartite.
2-Colouring

- How hard is 2-COLOURING?
- Claim: A graph is 2-colourable if and only if it is bipartite.
- Testing 2-colourability is possible in $O(|V| + |E|)$ time.
2-Colouring

- How hard is 2-COLOURING?
- Claim: A graph is 2-colourable if and only if it is bipartite.
- Testing 2-colourability is possible in $O(|V| + |E|)$ time.
- What about 3-COLOURING? Is it easy to exhibit a certificate that a graph cannot be coloured with three colours?

![Graph diagram](image)

**Figure 8.10** A graph that is not 3-colorable.
3-Colouring is $\mathcal{NP}$-Complete

- Why is 3-Colouring in $\mathcal{NP}$?
3-Colouring is $\mathcal{NP}$-Complete

- Why is 3-Colouring in $\mathcal{NP}$?
- $3$-SAT $\leq_p$ 3-Colouring.
### 3-SAT $\leq_p$ 3-Colouring: Encoding Variables

<table>
<thead>
<tr>
<th>3-SAT</th>
<th>3-Colouring</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean variables</td>
<td></td>
</tr>
<tr>
<td>True or False</td>
<td></td>
</tr>
<tr>
<td>Clauses</td>
<td></td>
</tr>
<tr>
<td>Is there a satisfying assignment?</td>
<td>Does a 3-colouring exist?</td>
</tr>
</tbody>
</table>
### 3-SAT $\leq_P$ 3-Colouring: Encoding Variables

<table>
<thead>
<tr>
<th>3-SAT</th>
<th>3-Colouring</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean variables</td>
<td>Nodes</td>
</tr>
<tr>
<td><em>True or False</em></td>
<td>Colours called <em>True</em>, <em>False</em>, and <em>Base</em></td>
</tr>
<tr>
<td>Clauses</td>
<td></td>
</tr>
<tr>
<td>Is there a satisfying assignment?</td>
<td>Does a 3-colouring exist?</td>
</tr>
</tbody>
</table>
### 3-SAT \( \leq_p \) 3-Colouring: Encoding Variables

<table>
<thead>
<tr>
<th>3-SAT</th>
<th>3-Colouring</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean variables</td>
<td>Nodes</td>
</tr>
<tr>
<td>\textit{True or False}</td>
<td>Colours called \textit{True, False, and Base}</td>
</tr>
<tr>
<td>Clauses</td>
<td>“Gadget”</td>
</tr>
<tr>
<td>Is there a satisfying assignment?</td>
<td>Does a 3-colouring exist?</td>
</tr>
</tbody>
</table>
3-SAT $\leq_P$ 3-Colouring: Encoding Variables

<table>
<thead>
<tr>
<th>3-SAT</th>
<th>3-Colouring</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean variables</td>
<td>Nodes</td>
</tr>
<tr>
<td><em>True or False</em> Clauses</td>
<td>Colours called <em>True, False</em>, and <em>Base</em></td>
</tr>
<tr>
<td>Is there a satisfying assignment?</td>
<td>Does a 3-colouring exist?</td>
</tr>
</tbody>
</table>

Figure 8.11 The beginning of the reduction for 3-Colouring.
### 3-SAT $\leq_P$ 3-Colouring: Encoding Variables

<table>
<thead>
<tr>
<th>3-SAT</th>
<th>3-Colouring</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean variables</td>
<td>Nodes</td>
</tr>
<tr>
<td><em>True or False</em></td>
<td>Colours called <em>True</em>, <em>False</em>, and <em>Base</em></td>
</tr>
<tr>
<td>Clauses</td>
<td>&quot;Gadget&quot;</td>
</tr>
<tr>
<td>Is there a satisfying assignment?</td>
<td>Does a 3-colouring exist?</td>
</tr>
</tbody>
</table>

- $x_i$ corresponds to node $v_i$ and $\overline{x_i}$ corresponds to node $\overline{v_i}$.

**Figure 8.11** The beginning of the reduction for 3-Colouring.
### 3-SAT \( \leq_p \) 3-Colouring: Encoding Variables

<table>
<thead>
<tr>
<th>3-SAT</th>
<th>3-Colouring</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean variables</td>
<td>Nodes</td>
</tr>
<tr>
<td>True or False</td>
<td>Colours called True, False, and Base</td>
</tr>
<tr>
<td>Clauses</td>
<td>“Gadget”</td>
</tr>
<tr>
<td>Is there a satisfying assignment?</td>
<td>Does a 3-colouring exist?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( x_i )</th>
<th>Corresponds to node ( v_i ) and ( \overline{x_i} ) corresponds to node ( \overline{v_i} ).</th>
</tr>
</thead>
<tbody>
<tr>
<td>( v_i )</td>
<td>In any 3-Colouring, nodes ( v_i ) and ( \overline{v_i} ) get a colour different from Base.</td>
</tr>
<tr>
<td>( \text{True colour} )</td>
<td>Colour assigned to the True node; False colour: colour assigned to the False node.</td>
</tr>
<tr>
<td>( \text{Set } x_i \text{ to 1 iff } v_i \text{ gets the True colour.} )</td>
<td></td>
</tr>
</tbody>
</table>

![Diagram](image)

**Figure 8.11** The beginning of the reduction for 3-Coloring.
3-SAT $\leq_P$ 3-Colouring: Encoding Clauses

Consider the clause $C_1 = x_1 \lor \bar{x}_2 \lor x_3$. If all of $v_1$, $v_2$, or $v_3$ get the False colour, then the top node in the subgraph cannot be coloured in a 3-colouring. If at least one of $v_1$, $v_2$, or $v_3$ does not get the False colour, then the top node in the subgraph can be coloured in a 3-colouring. Claim: Graph is 3-colourable iff input to 3-SAT is satisfiable.
Consider the clause $C_1 = x_1 \lor \overline{x_2} \lor x_3$.

Attach a six-node subgraph for this clause to the rest of the graph.

The top node can only be colored if one of $v_1$, $v_2$, or $v_3$ does not get the $False$ color.

**Figure 8.12** Attaching a subgraph to represent the clause $x_1 \lor \overline{x_2} \lor x_3$. 

**Claim:** If all of $v_1$, $v_2$, or $v_3$ get the $False$ color, then the top node in the subgraph cannot be colored in a 3-colouring.

**Claim:** If at least one of $v_1$, $v_2$, or $v_3$ does not get the $False$ color, then the top node in the subgraph can be colored in a 3-colouring.

**Claim:** Graph is 3-colourable iff input to 3-SAT is satisfiable.
Consider the clause $C_1 = x_1 \lor \overline{x_2} \lor x_3$.

- Attach a six-node subgraph for this clause to the rest of the graph.
- Attach a copy of six-node subgraph similarly for every other clause.

**Figure 8.12** Attaching a subgraph to represent the clause $x_1 \lor \overline{x_2} \lor x_3$. 

The top node can only be colored if one of $v_1$, $\overline{v_2}$, or $v_3$ does not get the False color.
3-SAT \( \leq_p \) 3-Colouring: Encoding Clauses

- Consider the clause \( C_1 = x_1 \lor \overline{x_2} \lor x_3 \).
- Attach a six-node subgraph for this clause to the rest of the graph.
- Attach a copy of six-node subgraph similarly for every other clause.

Claim: If all of \( v_1 \), \( \overline{v_2} \), or \( v_3 \) get the False colour, then the top node in the subgraph cannot be coloured in a 3-colouring.

**Figure 8.12** Attaching a subgraph to represent the clause \( x_1 \lor \overline{x_2} \lor x_3 \).
**3-SAT \( \leq_P \) 3-Colouring: Encoding Clauses**

- Consider the clause \( C_1 = x_1 \lor \overline{x_2} \lor x_3 \).
- Attach a six-node subgraph for this clause to the rest of the graph.
- Attach a copy of six-node subgraph similarly for every other clause.

**Claim:** If all of \( v_1, \overline{v_2}, \) or \( v_3 \) get the *False* colour, then the top node in the subgraph cannot be coloured in a 3-colouring.

**Claim:** If at least one of \( v_1, \overline{v_2}, \) or \( v_3 \) does not get the *False* colour, then the top node in the subgraph can be coloured in a 3-colouring.

**Figure 8.12** Attaching a subgraph to represent the clause \( x_1 \lor \overline{x_2} \lor x_3 \).
3-SAT $\leq_P$ 3-Colouring: Encoding Clauses

- Consider the clause $C_1 = x_1 \lor \overline{x_2} \lor x_3$.
- Attach a six-node subgraph for this clause to the rest of the graph.
- Attach a copy of six-node subgraph similarly for every other clause.

Claim: If all of $v_1$, $\overline{v_2}$, or $v_3$ get the False colour, then the top node in the subgraph cannot be coloured in a 3-colouring.

Claim: If at least one of $v_1$, $\overline{v_2}$, or $v_3$ does not get the False colour, then the top node in the subgraph can be coloured in a 3-colouring.

Claim: Graph is 3-colourable iff input to 3-SAT is satisfiable.

Figure 8.12 Attaching a subgraph to represent the clause $x_1 \lor \overline{x_2} \lor x_3$. 
Subset Sum

**Subset Sum**

**INSTANCE:** A set of $n$ natural numbers $w_1, w_2, \ldots, w_n$ and a target $W$.

**QUESTION:** Is there a subset of $\{w_1, w_2, \ldots, w_n\}$ whose sum is $W$?
**Subset Sum**

**Instance:** A set of $n$ natural numbers $w_1, w_2, \ldots, w_n$ and a target $W$.

**Question:** Is there a subset of $\{w_1, w_2, \ldots, w_n\}$ whose sum is $W$?

- **Subset Sum** is a special case of the **Knapsack Problem** (see Chapter 6.4 of the textbook).
Subset Sum

**Subset Sum**

**INSTANCE:** A set of $n$ natural numbers $w_1, w_2, \ldots, w_n$ and a target $W$.  
**QUESTION:** Is there a subset of \{ $w_1, w_2, \ldots, w_n$ \} whose sum is $W$?

- **Subset Sum** is a special case of the **Knapsack Problem** (see Chapter 6.4 of the textbook).
- There is a dynamic programming algorithm for **Subset Sum** that runs in $O(nW)$ time.
Subset Sum

INSTANCE: A set of $n$ natural numbers $w_1, w_2, \ldots, w_n$ and a target $W$.

QUESTION: Is there a subset of \{w_1, w_2, \ldots, w_n\} whose sum is $W$?

- **Subset Sum** is a special case of the Knapsack Problem (see Chapter 6.4 of the textbook).
- There is a dynamic programming algorithm for **Subset Sum** that runs in $O(nW)$ time. This algorithm’s running time is exponential in the size of the input.
Subset Sum

Subset Sum

INSTANCE: A set of $n$ natural numbers $w_1, w_2, \ldots, w_n$ and a target $W$.

QUESTION: Is there a subset of $\{w_1, w_2, \ldots, w_n\}$ whose sum is $W$?

- Subset Sum is a special case of the Knapsack Problem (see Chapter 6.4 of the textbook).

- There is a dynamic programming algorithm for Subset Sum that runs in $O(nW)$ time. This algorithm’s running time is exponential in the size of the input.

- Claim: Subset Sum is $\mathcal{NP}$-Complete, 3-Dimensional Matching $\leq_P$ Subset Sum.
Subset Sum

**INSTANCE:** A set of \( n \) natural numbers \( w_1, w_2, \ldots, w_n \) and a target \( W \).

**QUESTION:** Is there a subset of \( \{w_1, w_2, \ldots, w_n\} \) whose sum is \( W \)?

- **Subset Sum** is a special case of the **Knapsack Problem** (see Chapter 6.4 of the textbook).

- There is a dynamic programming algorithm for **Subset Sum** that runs in \( O(nW) \) time. This algorithm’s running time is exponential in the size of the input.

- Claim: **Subset Sum** is \( \mathcal{NP} \)-Complete, 3-Dimensional Matching \( \leq_P \) **Subset Sum**.

- Caveat: Special case of **Subset Sum** in which \( W \) is bounded by a polynomial function of \( n \) is not \( \mathcal{NP} \)-Complete (read pages 494–495 of your textbook).
Examples of Hard Computational Problems
Bejeweled, Candy Crush and other Match-Three Games are (NP-)Hard

Luciano Gualà, Stefano Leucci, Emanuele Natale

(Submitted on 24 Mar 2014)

The twentieth century has seen the rise of a new type of video games targeted at a mass audience of "casual" gamers. Many of these games require the player to swap items in order to form matches of three and are collectively known as \textbf{tile-matching match-three games}. Among these, the most influential one is arguably \textbf{Bejeweled} in which the matched items (gems) pop and the above gems fall in their place. Bejeweled has been ported to many different platforms and influenced an incredible number of similar games. Very recently one of them, named \textbf{Candy Crush Saga} enjoyed a huge popularity and quickly went viral on social networks. We generalize this kind of games by only parameterizing the size of the board, while all the other elements (such as the rules or the number of gems) remain unchanged. Then, we prove that answering many natural questions regarding such games is actually \textbf{NP-Hard}. These questions include determining if the player can reach a certain score, play for a certain number of turns, and others.
Examples of Hard Computational Problems

**Fig. 1** A Typical Minesweeper Position

**Fig. 2** Impossible Minesweeper position.
Examples of Hard Computational Problems

<table>
<thead>
<tr>
<th>Strategy</th>
<th>3-SAT</th>
<th>Sequencing Problems</th>
<th>Partitioning Problems</th>
<th>Other Problems</th>
</tr>
</thead>
</table>

**Examples of Hard Computational Problems**

**Fig. 1** A Typical Minesweeper Position

**Fig. 2** Impossible Minesweeper Position

---

**MINESWEEPER IS NP-COMPLETE**

---

T. M. Murali

April 21, 28, 2021

---

RICHARD KAYE

---

NP-Complete Problems
Examples of Hard Computational Problems

Tetris is Hard, Even to Approximate

Erik D. Demaine, Susan Hohenberger, David Liben-Nowell

(Submitted on 21 Oct 2002)

In the popular computer game of Tetris, the player is given a sequence of tetromino pieces and must pack them into a rectangular gameboard initially occupied by a given configuration of filled squares; any completely filled row of the gameboard is cleared and all pieces above it drop by one row. We prove that in the offline version of Tetris, it is NP-complete to maximize the number of cleared rows, maximize the number of tetrises (quadruples of rows simultaneously filled and cleared), minimize the maximum height of an occupied square, or maximize the number of pieces placed before the game ends. We furthermore show the extreme inapproximability of the first and last of these objectives to within a factor of $p^\epsilon(1-\epsilon)$, when given a sequence of $p$ pieces, and the inapproximability of the third objective to within a factor of $(2-\epsilon)$, for any $\epsilon>0$. Our results hold under several variations on the rules of Tetris, including different models of rotation, limitations on player agility, and restricted piece sets.
More Examples of Hard Computational Problems

(from Kevin Wayne’s slides at Princeton University)

- Aerospace engineering: optimal mesh partitioning for finite elements.
- Biology: protein folding.
- Chemical engineering: heat exchanger network synthesis.
- Civil engineering: equilibrium of urban traffic flow.
- Economics: computation of arbitrage in financial markets with friction.
- Electrical engineering: VLSI layout.
- Environmental engineering: optimal placement of contaminant sensors.
- Financial engineering: find minimum risk portfolio of given return.
- Game theory: find Nash equilibrium that maximizes social welfare.
- Genomics: phylogeny reconstruction.
- Mechanical engineering: structure of turbulence in sheared flows.
- Medicine: reconstructing 3-D shape from biplane angiocardioigram.
- Operations research: optimal resource allocation.
- Physics: partition function of 3-D Ising model in statistical mechanics.
- Politics: Shapley-Shubik voting power.
- Pop culture: Minesweeper consistency.
- Statistics: optimal experimental design.