Introduction to Algorithms and Data Structures

NP-completeness: A closer look
The class \( \text{NP} \)

- Stands for “\textit{non deterministic polynomial time}”.

- Problems that can be solved in polynomial time by a non-deterministic Turing machine.

- More intuitive definition:
  
  - Problems such that, \textit{if a solution is given}, it can be \textit{checked} that it is indeed a solution in polynomial time.

  - \textit{Efficiently verifiable}.
The class NP
The class NP

Interval Scheduling
The class **NP**

- Interval Scheduling
- Weighted Interval Scheduling
The class NP

- Interval Scheduling
- Weighted Interval Scheduling
- Shortest Paths in Graphs
The class NP

- Interval Scheduling
- Weighted Interval Scheduling
- Shortest Paths in Graphs
- Minimum-cost Paths in Graphs
The class NP

Interval Scheduling
Weighted Interval Scheduling

Shortest Paths in Graphs
Minimum-cost Paths in Graphs
Testing Bipartiteness
The class NP

Interval Scheduling
Weighted Interval Scheduling
Shortest Paths in Graphs
Minimum-cost Paths in Graphs
Testing Bipartiteness
Topological Sort
The class **NP**

- Interval Scheduling
- Weighted Interval Scheduling
- Shortest Paths in Graphs
- Minimum-cost Paths in Graphs
- Testing Bipartiteness
- Topological Sort
- Subset Sum
The class **NP**

- Interval Scheduling
- Weighted Interval Scheduling
- Subset Sum
- Knapsack
- Shortest Paths in Graphs
- Minimum-cost Paths in Graphs
- Testing Bipartiteness
- Topological Sort
3 SAT

- A CNF formula with \( m \) clauses and \( k \) literals.

\[ \phi = (x_1 \lor x_5 \lor x_3) \land (x_2 \lor x_6 \lor \neg x_5) \land \ldots \land (x_3 \lor x_8 \lor x_{12}) \]

- (“An AND of ORs”).

- Each clause has three literals.

- Truth assignment: A value in \( \{0, 1\} \) for each variable \( x_i \).

- Satisfying assignment: A truth assignment which makes the formula evaluate to 1 (= true).

- Computational problem 3SAT: Decide if the input formula \( \phi \) has a satisfying assignment.
The class NP

- Interval Scheduling
- Weighted Interval Scheduling
- Shortest Paths in Graphs
- Minimum-cost Paths in Graphs
- Testing Bipartiteness
- Topological Sort
- Subset Sum
- Knapsack
The class **NP**

- Interval Scheduling
- Weighted Interval Scheduling
- Subset Sum
- Knapsack
- Shortest Paths in Graphs
- Minimum-cost Paths in Graphs
- Testing Bipartiteness
- Topological Sort
- 3SAT
Vertex Cover
decision version

• **Definition:** A vertex cover $C$ of a graph $G=(V, E)$ is a subset of the nodes such that every edge $e$ in the graph has at least one endpoint in $C$.

• **Definition:** A minimum vertex cover is a vertex cover of the smallest possible size.

• **Vertex Cover**
  **Input:** A graph $G=(V, E)$ and a number $k$
  **Output:** Is there a vertex cover of size $\leq k$?
The class **NP**

- Interval Scheduling
- Weighted Interval Scheduling
- Subset Sum
- Knapsack
- Shortest Paths in Graphs
- Minimum-cost Paths in Graphs
- Testing Bipartiteness
- Topological Sort
- 3SAT
The class NP

- Interval Scheduling
- Weighted Interval Scheduling
- Shortest Paths in Graphs
- Minimum-cost Paths in Graphs
- Testing Bipartiteness
- Topological Sort
- Subset Sum
- Knapsack
- 3SAT
- Vertex Cover
Other problems in NP
Other problems in NP

- **Independent Set in graph G**: A set of nodes in the graph, such that there is no edge between any two nodes in the set.
Other problems in NP

- **Independent Set in graph G**: A set of nodes in the graph, such that there is no edge between any two nodes in the set.

- **Maximum Independent Set**
  Given a graph G, find an independent set of maximum size.
Other problems in \textbf{NP}

- **Independent Set in graph** $G$: A set of nodes in the graph, such that there is no edge between any two nodes in the set.

- **Maximum Independent Set**
  Given a graph $G$, find an independent set of maximum size.

- **Maximum Independent Set (decision version)**
  Given a graph $G$, and an integer $k$, is there an independent set of size at least $k$?
Other problems in NP
Other problems in NP

- **Set Packing**
  Given a set $U$ of elements, a collection $S_1, \ldots, S_m$ of subsets of $U$ and a number $k$, does there exist a collection of at least $k$ of these sets such that no two of them intersect?
Other problems in NP

- **Set Packing**
  Given a set $U$ of elements, a collection $S_1, \ldots, S_m$ of subsets of $U$ and a number $k$, does there exist a collection of at least $k$ of these sets such that no two of them intersect?

- **Set Cover**
  Given a set $U$ of elements, a collection $S_1, \ldots, S_m$ of subsets of $U$ and a number $k$, does there exist a collection of at most $k$ of these sets whose union is equal to $U$?
Other problems in NP
Other problems in NP

- 3-Dimensional Matching
  Given disjoint sets $X$, $Y$ and $Z$ each of size $n$, and given a set $T$ (which is a subset of $X \times Y \times Z$) of ordered triples, does there exist a set of $n$ triples in $T$, so that each element of $X \cup Y \cup Z$ is contained in exactly in one of these triples?
Other problems in NP
Other problems in NP

- k-Colouring of a graph $G$: A function $f: V \rightarrow \{1, \ldots, k\}$ so that for every edge $(u, v)$ we have that $f(u) \neq f(v)$. 
Other problems in \textbf{NP}

- \textbf{k-Colouring of a graph }G: A function \( f: V \rightarrow \{1, \ldots, k\} \) so that for every edge \((u, v)\) we have that \( f(u) \neq f(v) \).

- \textbf{3-Colouring}
  Given a graph \( G \), does it have a 3-Colouring?
Other problems in NP
Other problems in NP

- Hamiltonian cycle in a directed graph $G$: A cycle in a directed graph that visits each vertex \textit{exactly once}.
Other problems in $\text{NP}$

- **Hamiltonian cycle in a directed graph $G$**: A cycle in a directed graph that visits each vertex *exactly once*.

- **Hamiltonian path in a directed graph $G$**: A path in a directed graph that contains each vertex *exactly once*. 
Other problems in $\mathbf{NP}$

- **Hamiltonian cycle in a directed graph $G$**: A cycle in a directed graph that visits each vertex \textit{exactly once}.

- **Hamiltonian path in a directed graph $G$**: A path in a directed graph that contains each vertex \textit{exactly once}.

- **Hamiltonian Cycle**
  Given a directed graph $G$, does it have a Hamiltonian Cycle?
Other problems in NP

- **Hamiltonian cycle in a directed graph G**: A cycle in a directed graph that visits each vertex *exactly once*.

- **Hamiltonian path in a directed graph G**: A path in a directed graph that contains each vertex *exactly once*.

- **Hamiltonian Cycle**
  Given a directed graph G, does it have a Hamiltonian Cycle?

- **Hamiltonian Path**
  Given a directed graph G, does it have a Hamiltonian Path?
Other problems in \textbf{NP}

- \textbf{Hamiltonian cycle in a directed graph} \(G\): A cycle in a directed graph that visits each vertex \textit{exactly once}.

- \textbf{Hamiltonian path in a directed graph} \(G\): A path in a directed graph that contains each vertex \textit{exactly once}.

- \textbf{Hamiltonian Cycle}
  Given a directed graph \(G\), does it have a Hamiltonian Cycle?

- \textbf{Hamiltonian Path}
  Given a directed graph \(G\), does it have a Hamiltonian Path?

- \textbf{Traveling Salesman}
  (def Kleinberg and Tardos, p. 474).
The class **NP**

- Interval Scheduling
- Weighted Interval Scheduling
- Subset Sum
- Knapsack
- Shortest Paths in Graphs
- Minimum-cost Paths in Graphs
- Testing Bipartiteness
- Topological Sort
- 3SAT
- Vertex Cover
The class NP

Interval Scheduling
Weighted Interval Scheduling

Shortest Paths in Graphs
Minimum-cost Paths in Graphs
Testing Bipartiteness
Topological Sort

Vertex Cover
Independent Set

Subset Sum
Knapsack

3SAT
The class NP
The class $\textbf{NP}$

- Interval Scheduling
- Weighted Interval Scheduling
- Subset Sum
- Knapsack

- Shortest Paths in Graphs
- Minimum-cost Paths in Graphs
- Testing Bipartiteness
- Topological Sort

- 3SAT

- Vertex Cover
- Independent Set

- Set Cover
- Set Packing
The class $\mathbf{NP}$
The class \textbf{NP}
The class NP

- Interval Scheduling
- Weighted Interval Scheduling
- Subset Sum
- Knapsack
- Hamiltonian Cycle
- 3SAT
- Shortest Paths in Graphs
- Minimum-cost Paths in Graphs
- Testing Bipartiteness
- Topological Sort
- 3-Colouring
- Vertex Cover
- Independent Set
- Set Cover
- Set Packing
- 3D-Matching
The class NP

- Interval Scheduling
- Weighted Interval Scheduling
- Shortest Paths in Graphs
- Minimum-cost Paths in Graphs
- Testing Bipartiteness
- Topological Sort
- Vertex Cover
- Independent Set
- Set Cover
- Set Packing
- 3D-Matching
- Subset Sum
- Knapsack
- 3SAT
- Hamiltonian Cycle
- Hamiltonian Path
- 3-Colouring
The class NP
A problem $B$ is $\text{NP}$-complete if

- *It is in $\text{NP}$.*
  - i.e., it has a polynomial-time verifiable solution.

- *It is $\text{NP}$-hard.*
  - i.e., every problem in $\text{NP}$ can be efficiently reduced to it.
NP-completeness

- Interval Scheduling
- Weighted Interval Scheduling
- Subset Sum
- Knapsack
- Hamiltonian Cycle
- Hamiltonian Path
- Traveling Salesman
- Shortest Paths in Graphs
- Minimum-cost Paths in Graphs
- Testing Bipartiteness
- Topological Sort
- 3SAT
- Vertex Cover
- Independent Set
- Set Cover
- Set Packing
- 3D-Matching
- 3-Colouring
NP-completeness

Interval Scheduling
Weighted Interval Scheduling

Subset Sum
Knapsack

Hamiltonian Cycle
Hamiltonian Path
Traveling Salesman

Shortest Paths in Graphs
Minimum-cost Paths in Graphs
Testing Bipartiteness
Topological Sort

3SAT

Vertex Cover
Independent Set
Set Cover
Set Packing
3D-Matching

3-Colouring
NP-completeness

- Interval Scheduling
- Weighted Interval Scheduling
- Subset Sum
- Knapsack
- Hamiltonian Cycle
- Hamiltonian Path
- Traveling Salesman
- Shortest Paths in Graphs
- Minimum-cost Paths in Graphs
- Testing Bipartiteness
- Topological Sort
- Vertex Cover
- Independent Set
- Set Cover
- Set Packing
- 3D-Matching
- 3-Colouring
NP-completeness

Interval Scheduling
Weighted Interval Scheduling

Shortest Paths in Graphs
Minimum-cost Paths in Graphs
Testing Bipartiteness
Topological Sort

Vertex Cover
Independent Set

Set Cover
Set Packing
3D-Matching

Subset Sum
Knapsack

3SAT
SAT

Hamiltonian Cycle
Hamiltonian Path
Traveling Salesman

3-Colouring
NP-completeness

Interval Scheduling
Weighted Interval Scheduling
Subset Sum
Knapsack
Hamiltonian Cycle
Hamiltonian Path
Traveling Salesman
Shortest Paths in Graphs
Minimum-cost Paths in Graphs
Testing Bipartiteness
Topological Sort
3SAT
SAT
3-Colouring
Vertex Cover
Independent Set
Set Cover
Set Packing
3D-Matching
NP-completeness

- Interval Scheduling
- Weighted Interval Scheduling
- Subset Sum
- Knapsack
- Hamiltonian Cycle
- Hamiltonian Path
- Traveling Salesman
- Shortest Paths in Graphs
- Minimum-cost Paths in Graphs
- Testing Bipartiteness
- Topological Sort
- Vertex Cover
- Independent Set
- Set Cover
- Set Packing
- 3D-Matching
- 3-Colouring
- 3SAT
- SAT
NP-completeness

- Interval Scheduling
- Weighted Interval Scheduling
- Subset Sum
- Knapsack
- Hamiltonian Cycle
- Hamiltonian Path
- Traveling Salesman
- Shortest Paths in Graphs
- Minimum-cost Paths in Graphs
- Testing Bipartiteness
- Topological Sort
- 3SAT
- 3-Colouring
- SAT
- 3SAT
- Vertex Cover
- Independent Set
- Set Cover
- Set Packing
- 3D-Matching
NP-completeness

- Interval Scheduling
- Weighted Interval Scheduling
- Shortest Paths in Graphs
- Minimum-cost Paths in Graphs
- Testing Bipartiteness
- Topological Sort
- Subset Sum
- Knapsack
- Vertex Cover
- Independent Set
- Set Cover
- Set Packing
- 3D-Matching
- 3SAT
- SAT
- 3-Colouring
- Hamiltonian Cycle
- Hamiltonian Path
- Traveling Salesman
NP-completeness

The Cook-Levin Theorem (1971, 1973)
The Cook-Levin Theorem (1971, 1973)

The proof uses a generic argument that if a problem has a solution which can be verified in polynomial-time, then it reduces in polynomial time to the SAT problem.
The Cook-Levin Theorem (1971, 1973)

The proof uses a generic argument that if a problem has a solution which can be verified in polynomial-time, then it reduces in polynomial time to the SAT problem.
NP-completeness
NP-completeness

• What does the NP-completeness of SAT mean?
What does the NP-completeness of SAT mean?

It means that it is \textit{at least as hard to solve} as any other problem in NP.
NP-completeness

• What does the NP-completeness of SAT mean?

• It means that it is *at least as hard to solve* as any other problem in NP.

• In particular, if we had a polynomial-time algorithm for solving SAT, *we could solve any other problem in NP*, via the reduction (the arrow).
Wooclap!
NP-completeness
NP-completeness

• What does the NP-completeness of SAT mean?

  • It means that it is *at least as hard to solve* as any other problem in NP.

  • In particular, if we had a polynomial-time algorithm for solving SAT, *we could solve any other problem in NP*, via the reduction (the arrow).

  • At this stage, that doesn’t necessarily say much.
NP-completeness
NP-completeness

• Some time passes, and we tried and tried to find a polynomial-time algorithm for SAT (or 3SAT) and we are still looking for one…
NP-completeness

• Some time passes, and we tried and tried to find a polynomial-time algorithm for SAT (or 3SAT) and we are still looking for one…

• This seems to suggest that SAT might be in some sense harder to solve than e.g., Interval Scheduling or Testing Bipartiteness.
• Some time passes, and we tried and tried to find a polynomial-time algorithm for SAT (or 3SAT) and we are still looking for one…

• This seems to suggest that SAT might be in some sense *harder to solve* than e.g., Interval Scheduling or Testing Bipartiteness.

• We know of course that it is at least as hard to solve, by virtue of being NP-complete.
NP-completeness

• Some time passes, and we tried and tried to find a polynomial-time algorithm for SAT (or 3SAT) and we are still looking for one…

• This seems to suggest that SAT might be in some sense harder to solve than e.g., Interval Scheduling or Testing Bipartiteness.

• We know of course that it is at least as hard to solve, by virtue of being NP-complete.

• But this seems to suggest that some problems in NP are harder than others.
NP-completeness
After a while, we gave up on SAT and decided to try to solve our new favourite problem, Vertex Cover, in polynomial time.
NP-completeness

• After a while, we gave up on SAT and decided to try to solve our new favourite problem, Vertex Cover, in polynomial time.

• We tried hard and we failed… We are still looking for a polynomial-time algorithm.
NP-completeness

- After a while, we gave up on SAT and decided to try to solve our new favourite problem, Vertex Cover, in polynomial time.

- We tried hard and we failed... We are still looking for a polynomial-time algorithm.

- Hmm, maybe Vertex Cover is also harder to solve than, say, Interval Scheduling or Testing Bipartiteness...
NP-completeness

Interval Scheduling
Weighted Interval Scheduling
Subset Sum
Knapsack
Hamiltonian Cycle
Hamiltonian Path
Traveling Salesman

Shortest Paths in Graphs
Minimum-cost Paths in Graphs
Testing Bipartiteness
Topological Sort
3SAT
SAT
3-Colouring

Vertex Cover
Independent Set
Set Cover
Set Packing
3D-Matching
NP-completeness

- Interval Scheduling
  - Weighted Interval Scheduling
- Subset Sum
  - Knapsack
- Shortest Paths in Graphs
  - Minimum-cost Paths in Graphs
  - Testing Bipartiteness
  - Topological Sort
- Hamiltonian Cycle
  - Hamiltonian Path
  - Traveling Salesman
- 3-Colouring
- 3SAT
- SAT
- Vertex Cover
  - Independent Set
- Set Cover
  - Set Packing
  - 3D-Matching
NP-completeness

Interval Scheduling
Weighted Interval Scheduling

Subset Sum
Knapsack

Shortest Paths in Graphs
Minimum-cost Paths in Graphs
Testing Bipartiteness
Topological Sort

3SAT

Hamiltonian Cycle
Hamiltonian Path
Traveling Salesman

3-Colouring

Vertex Cover
Independent Set

Set Cover
Set Packing
3D-Matching
NP-completeness

- Interval Scheduling
- Weighted Interval Scheduling
- Shortest Paths in Graphs
- Minimum-cost Paths in Graphs
- Testing Bipartiteness
- Topological Sort
- Vertex Cover
- Independent Set
- Set Cover
- Set Packing
- 3D-Matching
- 3-Colouring
- Subset Sum
- Knapsack
- 3SAT
- SAT
- Hamiltonian Cycle
- Hamiltonian Path
- Traveling Salesman
NP-completeness

Interval Scheduling
Weighted Interval Scheduling

Subset Sum
Knapsack

Hamiltonian Cycle
Hamiltonian Path
Traveling Salesman

3SAT

SAT

3-Colouring

Shortest Paths in Graphs
Minimum-cost Paths in Graphs
Testing Bipartiteness
Topological Sort

Vertex Cover
Independent Set

Set Cover
Set Packing
3D-Matching
NP-completeness
**NP-completeness**

- Vertex Cover is NP-complete.
**NP-completeness**

- Vertex Cover is NP-complete.
- This means that it is at least as hard as any problem in NP, including SAT.
NP-completeness

• Vertex Cover is NP-complete.

• This means that it is at least as hard as any problem in NP, including SAT.

• But we really tried to solve SAT in polynomial-time… No wonder we failed to solve Vertex Cover too!
NP-completeness
NP-completeness

- Ok, let’s try to solve Independent Set in polynomial time then.
NP-completeness

• Ok, let’s try to solve Independent Set in polynomial time then.

• Arghh, we can’t solve that either!
NP-completeness

Interval Scheduling
Weighted Interval Scheduling

Shortest Paths in Graphs
Minimum-cost Paths in Graphs
Testing Bipartiteness
Topological Sort

Subset Sum
Knapsack

3SAT

Vertex Cover
Independent Set

Hamiltonian Cycle
Hamiltonian Path
Traveling Salesman

3D-Matching

Set Cover
Set Packing

3-Colouring
NP-completeness
NP-completeness

- Interval Scheduling
- Weighted Interval Scheduling
- Subset Sum
- Knapsack
- Hamiltonian Cycle
- Hamiltonian Path
- Traveling Salesman
- Shortest Paths in Graphs
- Minimum-cost Paths in Graphs
- Testing Bipartiteness
- Topological Sort
- 3SAT
- Vertex Cover
- Independent Set
- SAT
- Set Cover
- Set Packing
- 3D-Matching
- 3-Colouring
NP-completeness
NP-completeness
• Independent Set is NP-complete.
Independent Set is NP-complete.

This means that it is at least as hard as any problem in NP, including SAT and Vertex Cover.
• Independent Set is NP-complete.

• This means that it is at least as hard as any problem in NP, including SAT and Vertex Cover.

• But we really tried to solve SAT and Vertex Cover in polynomial-time… No wonder we failed to solve Independent Set too.
NP-completeness

Interval Scheduling
Weighted Interval Scheduling
Subset Sum
Knapsack

Shortest Paths in Graphs
Minimum-cost Paths in Graphs
Testing Bipartiteness
Topological Sort

3SAT
SAT

Vertex Cover
Independent Set
Set Cover
Set Packing
3D-Matching

Hamiltonian Cycle
Hamiltonian Path
Traveling Salesman

3-Colouring
NP-completeness
NP-completeness

Interval Scheduling
Weighted Interval Scheduling

Subset Sum
Knapsack

3SAT

3-Colouring

Vertex Cover
Independent Set

Set Cover
Set Packing
3D-Matching

Shortest Paths in Graphs
Minimum-cost Paths in Graphs
Testing Bipartiteness
Topological Sort

Hamiltonian Cycle
Hamiltonian Path
Traveling Salesman
NP-completeness

Interval Scheduling
Weighted Interval Scheduling

Shortest Paths in Graphs
Minimum-cost Paths in Graphs
Testing Bipartiteness
Topological Sort

Vertex Cover
Independent Set

Set Cover
Set Packing
3D-Matching

3-Colouring

Subset Sum
Knapsack

3SAT

Hamiltonian Cycle
Hamiltonian Path
Traveling Salesman

SAT

3D-Matching
NP-completeness

- Interval Scheduling
- Weighted Interval Scheduling

- Subset Sum
- Knapsack

- Shortest Paths in Graphs
- Minimum-cost Paths in Graphs
- Testing Bipartiteness
- Topological Sort

- 3SAT
- SAT

- Hamiltonian Cycle
- Hamiltonian Path
- Traveling Salesman

- Vertex Cover
- Independent Set

- 3D-Matching
- Set Cover
- Set Packing

- 3-Colouring
NP-completeness
NP-completeness

Interval Scheduling
Weighted Interval Scheduling

Subset Sum
Knapsack

Hamiltonian Cycle
Hamiltonian Path
Traveling Salesman

Shortest Paths in Graphs
Minimum-cost Paths in Graphs
Testing Bipartiteness
Topological Sort

3SAT
SAT

Vertex Cover
Independent Set

Set Cover
Set Packing
3D-Matching

3-Colouring
NP-completeness

Interval Scheduling
Weighted Interval Scheduling

Subset Sum
Knapsack

Hamiltonian Cycle
Hamiltonian Path
Traveling Salesman

3SAT
SAT

3-Colouring

Vertex Cover
Independent Set

Set Cover
Set Packing
3D-Matching

Shortest Paths in Graphs
Minimum-cost Paths in Graphs
Testing Bipartiteness
Topological Sort
NP-completeness

Interval Scheduling
Weighted Interval Scheduling

Subset Sum
Knapsack

Hamiltonian Cycle
Hamiltonian Path
Traveling Salesman

3SAT
SAT

3-Colouring

Vertex Cover
Independent Set

Set Cover
Set Packing
3D-Matching

Shortest Paths in Graphs
Minimum-cost Paths in Graphs
Testing Bipartiteness
Topological Sort
NP-completeness

- Interval Scheduling
- Weighted Interval Scheduling
- Subset Sum
- Knapsack
- Hamiltonian Cycle
- Hamiltonian Path
- Traveling Salesman
- 3SAT
- SAT
- 3-Colouring
- 3D-Matching
- Shortest Paths in Graphs
- Minimum-cost Paths in Graphs
- Testing Bipartiteness
- Topological Sort
- Vertex Cover
- Independent Set
- Set Cover
- Set Packing
- 3D-Matching
NP-completeness
NP-completeness

- 3SAT, Vertex Cover, Independent Set, Subset Sum, Knapsack, Hamiltonian Path, Hamiltonian Cycle, Traveling Salesman, 3-Colouring, Set Cover, Set Packing, 3D-Matching are all NP-complete.
NP-completeness

• 3SAT, Vertex Cover, Independent Set, Subset Sum, Knapsack, Hamiltonian Path, Hamiltonian Cycle, Traveling Salesman, 3-Colouring, Set Cover, Set Packing, 3D-Matching are all NP-complete.

• Actually, this is only a very small subset of NP-complete problems.
NP-completeness

- 3SAT, Vertex Cover, Independent Set, Subset Sum, Knapsack, Hamiltonian Path, Hamiltonian Cycle, Traveling Salesman, 3-Colouring, Set Cover, Set Packing, 3D-Matching are all NP-complete.

- Actually, this is only a very small subset of NP-complete problems.
  - Hundreds of other meaningful problems are NP-complete.
NP-completeness

• 3SAT, Vertex Cover, Independent Set, Subset Sum, Knapsack, Hamiltonian Path, Hamiltonian Cycle, Traveling Salesman, 3-Colouring, Set Cover, Set Packing, 3D-Matching are all NP-complete.

• Actually, this is only a very small subset of NP-complete problems.
  • Hundreds of other meaningful problems are NP-complete.

• We don’t know how to solve any one of those in polynomial-time.
The effect of NP-hardness
The effect of NP-hardness

• Imagine that you have a new favourite problem P.
The effect of **NP**-hardness

- Imagine that you have a new favourite problem $P$.
- You try to design a polynomial-time algorithm for it but you find it hard to do so.
The effect of NP-hardness

• Imagine that you have a new favourite problem $P$.

• You try to design a polynomial-time algorithm for it but you find it hard to do so.

• Then you discover that it can be reduced to one of all of these NP-complete problems.
The effect of NP-hardness

- Imagine that you have a new favourite problem $P$.

- You try to design a polynomial-time algorithm for it but you find it hard to do so.

- Then you discover that it can be reduced to one of all of these NP-complete problems.

- This means that if you succeeded in your quest, you would solve all of these problems in polynomial-time.
The effect of **NP**-hardness

- Imagine that you have a new favourite problem $P$.

- You try to design a polynomial-time algorithm for it but you find it hard to do so.

- Then you discover that it can be reduced to one of all of these NP-complete problems.

- This means that if you succeeded in your quest, you would solve all of these problems in polynomial-time.

- That would mean that you are smarter than generations of researchers and pretty much anyone else that has studied computer science ever.
Imagine that you have a new favourite problem $\text{P}$.

You try to design a polynomial-time algorithm for it but you find it hard to do so.

Then you discover that it can be reduced to one of all of these NP-complete problems.

This means that if you succeeded in your quest, you would solve all of these problems in polynomial-time.

That would mean that you are smarter than generations of researchers and pretty much anyone else that has studied computer science ever.

I don’t know about you, but I would probably be convinced that I am not going to come up with a polynomial-time algorithm!
Wooclap!
Reduction strategies
Reduction strategies

• For now, we’ll tell you what to reduce from.
Reduction strategies

• For now, we’ll tell you what to reduce from.

• And the reduction will be relatively simple.
Reduction strategies

• For now, we’ll tell you what to reduce from.

• And the reduction will be relatively simple.

• In general, the idea is to find a problem that looks similar to the one we are trying to prove NP-hardness for.
Reduction strategies

• For now, we’ll tell you what to reduce from.

• And the reduction will be relatively simple.

• In general, the idea is to find a problem that looks similar to the one we are trying to prove NP-hardness for.

• Try to think of reductions you have seen in the past.
Reduction strategies

- For now, we’ll tell you what to reduce from.
- And the reduction will be relatively simple.
- In general, the idea is to find a problem that looks similar to the one we are trying to prove NP-hardness for.
- Try to think of reductions you have seen in the past.
  - This takes time!
NP-completeness, a taxonomy

Packing problems
- Independent Set
- Set Packing

Covering problems
- Vertex Cover
- Set Cover

Partitioning problems
- 3D-Matching
- Graph Colouring

Sequencing problems
- Hamiltonian Cycle
- Hamiltonian Path
- Traveling Salesman

Numerical problems
- Subset Sum
- Knapsack

Constraint Satisfaction problems
- 3 SAT
NP-completeness
NP-completeness

• So when a problem is NP-complete, this means:
**NP-completeness**

• So when a problem is NP-complete, this means:
  
  • That it is in NP, and it is at least as hard to solve as any other problem in NP.
So when a problem is NP-complete, this means:

- That it is in NP, and it is at least as hard to solve as any other problem in NP.
- That it is unlikely that we solve it in polynomial time, as that would imply that we solve all the NP-complete problems.
So when a problem is NP-complete, this means:

- That it is in NP, and it is at least as hard to solve as any other problem in NP.

- That it is unlikely that we solve it in polynomial time, as that would imply that we solve all the NP-complete problems.

- That it is not solvable in polynomial time assuming $P \neq NP$. 
Wooclap!
NP-hardness is a worst-case impossibility
NP-hardness is a worst-case impossibility

- Let’s recall the NP-hardness proof for Vertex Cover.
NP-hardness is a worst-case impossibility

- Let’s recall the NP-hardness proof for Vertex Cover.

- If I could decide Vertex Cover on this graph, I could decide 3SAT.
NP-hardness is a worst-case impossibility
NP-hardness is a worst-case impossibility

- What about this graph? Can I decide Vertex Cover on this graph?
NP-hardness is a worst-case impossibility

- What about this graph? Can I decide Vertex Cover on this graph?

```
Choose one leave one
```

- “Choose one leave one” finds a minimum vertex cover.
NP-hardness is a worst-case impossibility
NP-hardness is a worst-case impossibility

- An NP-hardness result does not mean that (unless P=NP) we cannot solve the problem in polynomial time for any instance.
NP-hardness is a worst-case impossibility

• An NP-hardness result does not mean that (unless P=NP) we cannot solve the problem in polynomial time for any instance.

• It means that we cannot solve it in polynomial time for every instance.
NP-hardness is a worst-case impossibility

- An NP-hardness result does not mean that (unless P=NP) we cannot solve the problem in polynomial time for any instance.

- It means that we cannot solve it in polynomial time for every instance.

- For all we know, every other instance besides those used in the reduction could be easy to solve.
NP-hardness is a worst-case impossibility

- For all we know, every other instance besides those used in the reduction could be easy to solve.
NP-hardness is a worst-case impossibility

• For all we know, every other instance besides those used in the reduction could be easy to solve.

• Usually not the case! In practice usually we don’t have good ways of solving NP-hard problems.
NP-hardness is a worst-case impossibility

• For all we know, every other instance besides those used in the reduction could be easy to solve.

• Usually not the case! In practice usually we don’t have good ways of solving NP-hard problems.

• Still, sometimes we can provably design polynomial algorithms on certain input structures.
**NP-hardness is a worst-case impossibility**

- For all we know, every other instance besides those used in the reduction could be easy to solve.

- Usually not the case! In practice usually we don’t have good ways of solving NP-hard problems.

- Still, sometimes we can provably design polynomial algorithms on certain *input structures*.

- For example, a minimum Vertex Cover on *trees* can be found in polynomial time using Dynamic Programming.
Wooclap!
NP-hardness ≠ Exponential Time
NP-hardness $\neq$ Exponential Time

- This is true even if $P \neq NP$. 

NP-hardness \neq \text{Exponential Time}

- This is true even if $P \neq NP$.

- Roughgarden: Acceptable Inaccuracy #3.
NP-hardness $\not= \text{ Exponential Time}$

- This is true even if $P \neq NP$.
- Roughgarden: Acceptable Inaccuracy #3.
- Subexponential time: $n^{O(\lg n)}$ or $2^{O(\sqrt{n})}$. 
NP-hardness ≠ Exponential Time

- This is true even if $P \neq NP$.

- Roughgarden: Acceptable Inaccuracy #3.

- Subexponential time: $n^{O(\lg n)}$ or $2^{O(\sqrt{n})}$.

- There are NP-complete problems that can be solved in subexponential time.
NP-hardness $\neq$ Exponential Time

- This is true even if $P \neq NP$.
- Roughgarden: *Acceptable Inaccuracy #3.*
- Subexponential time: $n^{O(\lg n)}$ or $2^{O(\sqrt{n})}$.
- There are NP-complete problems that can be solved in subexponential time.
- Exponential Time Hypothesis (ETH): SAT requires exponential time to be solved.
NP-hardness $\not\equiv$ Exponential Time

• This is true even if $P \neq NP$.

• Roughgarden: Acceptable Inaccuracy #3.

• Subexponential time: $n^{O(\lg n)}$ or $2^{O(\sqrt{n})}$.

• There are NP-complete problems that can be solved in subexponential time.

• Exponential Time Hypothesis (ETH): SAT requires exponential time to be solved.

  • ETH $\Rightarrow P \neq NP$
NP-hardness vs NP-completeness
NP-hardness vs NP-completeness

• Every NP-complete problem is NP-hard.
NP-hardness vs NP-completeness

• Every NP-complete problem is NP-hard.

• Is every NP-hard problem NP-complete?
NP-hardness vs NP-completeness

• Every NP-complete problem is NP-hard.
• Is every NP-hard problem NP-complete?
• Are there problems that are NP-hard but not in NP?
Totally Quantified Boolean Formula (TQBF)
Totally Quantified Boolean Formula (TQBF)

• A CNF formula with \( m \) clauses and \( k \) literals, and a set of quantifiers.

\[ Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \phi(x_1, x_2, \ldots, x_n) \]

where \( Q_i \in \{ \forall, \exists \} \)
Totally Quantified Boolean Formula (TQBF)

• A CNF formula with $m$ clauses and $k$ literals, and a set of quantifiers.

$$Q_1x_1Q_2x_2\ldots Q_nx_n\phi(x_1, x_2, \ldots, x_n)$$

where $Q_i \in \{\forall, \exists\}$

• For example, we may have

$$\exists x_1 \forall x_2 \exists x_3, \ldots, \forall x_n\phi(x_1, x_2, x_3, \ldots, x_n)$$
Totally Quantified Boolean Formula (TQBF)

- A CNF formula with \( m \) clauses and \( k \) literals, and a set of quantifiers.

\[
Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \phi(x_1, x_2, \ldots, x_n)
\]

where \( Q_i \in \{ \forall, \exists \} \)

- For example, we may have

\[
\exists x_1 \forall x_2 \exists x_3, \ldots, \forall x_n \phi(x_1, x_2, x_3, \ldots, x_n)
\]

- We read “Does there exists \( x_1 \) such that for every \( x_2 \) there exists \( x_3 \) such that ... such that for every \( x_n \), the formula \( \phi \) is satisfiable?”
Or maybe a game of chess
Or maybe a game of chess

- Does white have a winning strategy?
Or maybe a game of chess

• Does white have a winning strategy?

• Does there exist a move for white, such that for every move of black, there exists a move for white, such that for every move of black, … , such that for every move of black, white wins?
Totally Quantified Boolean Formula (TQBF)

- A CNF formula with \( m \) clauses and \( k \) literals, and a set of quantifiers.

\[
Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \phi(x_1, x_2, \ldots, x_n)
\]

where \( Q_i \in \{ \forall, \exists \} \)

- For example, we may have
  \[
  \exists x_1 \ \forall x_2 \ \exists x_3, \ldots, \forall x_n \phi(x_1, x_2, x_3, \ldots, x_n)
  \]
Totally Quantified Boolean Formula (TQBF)

- A CNF formula with \( m \) clauses and \( k \) literals, and a set of quantifiers.

\[
Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \phi(x_1, x_2, \ldots, x_n)
\]

where \( Q_i \in \{ \forall, \exists \} \)

- For example, we may have

\[
\exists x_1 \forall x_2 \exists x_3, \ldots, \forall x_n \phi(x_1, x_2, x_3, \ldots, x_n)
\]

- TQBF is NP-hard. Why?
NP-completeness

- Interval Scheduling
- Weighted Interval Scheduling
- Subset Sum
- Knapsack
- Hamiltonian Cycle
- Hamiltonian Path
- Traveling Salesman
- Shortest Paths in Graphs
- Minimum-cost Paths in Graphs
- Testing Bipartiteness
- Topological Sort
- 3SAT
- SAT
- 3-Colouring
- Vertex Cover
- Independent Set
- Set Cover
- Set Packing
- 3D-Matching
- TQBF
NP-completeness

- Interval Scheduling
- Weighted Interval Scheduling
- Subset Sum
- Knapsack
- Hamiltonian Cycle
- Hamiltonian Path
- Traveling Salesman
- 3D-Matching
- 3-Colouring
- TQBF
- Shortest Paths in Graphs
- Minimum-cost Paths in Graphs
- Testing Bipartiteness
- Topological Sort
- 3SAT
- SAT
- Vertex Cover
- Independent Set
- Set Cover
- Set Packing
- 3D-Matching
NP-completeness

Interval Scheduling
Weighted Interval Scheduling

Subset Sum
Knapsack

Hamiltonian Cycle
Hamiltonian Path
Traveling Salesman

Shortest Paths in Graphs
Minimum-cost Paths in Graphs
Testing Bipartiteness
Topological Sort

3SAT
SAT

Vertex Cover
Independent Set

Set Cover
Set Packing
3D-Matching

3-Colouring

TQBF
Totally Quantified Boolean Formula (TQBF)

• A CNF formula with \textit{m clauses} and \textit{k literals}, and a set of quantifiers.

\[ Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \phi(x_1, x_2, \ldots, x_n) \]

where \( Q_i \in \{ \forall, \exists \} \)

• For example, we may have \( \exists x_1 \forall x_2 \exists x_3, \ldots, \forall x_n \phi(x_1, x_2, x_3, \ldots, x_n) \)

• We read “Does there exists \( x_1 \) such that for every \( x_2 \) there exists \( x_3 \) such that … such that for every \( x_n \), the formula \( \phi \) is satisfiable?”
Totally Quantified Boolean Formula (TQBF)

- A CNF formula with \( m \) clauses and \( k \) literals, and a set of quantifiers.

\[ Q_1 x_1 Q_2 x_2 \cdots Q_n x_n \phi(x_1, x_2, \ldots, x_n) \]

where \( Q_i \in \{ \forall, \exists \} \)

- For example, we may have \( \exists x_1 \forall x_2 \exists x_3, \ldots, \forall x_n \phi(x_1, x_2, x_3, \ldots, x_n) \)

- We read “Does there exists \( x_1 \) such that for every \( x_2 \) there exists \( x_3 \) such that … such that for every \( x_n \), the formula \( \phi \) is satisfiable?”

- For a given \( x_1 \) we have to check the values of \( x_2 \) for all possible values of the remaining \( x_4, x_6 \), etc. Does not seem to be doable in polynomial time.
An interesting but important note
An interesting but important note

• We cannot categorically say that TQBF is not efficiently verifiable (i.e., that it is not in NP).
An interesting but important note

• We cannot categorically say that TQBF is not efficiently verifiable (i.e., that it is not in NP).

• In the same way as we cannot categorically say that SAT is not efficiently solvable (i.e., that it is not in P).
An interesting but important note

• We cannot categorically say that TQBF is not efficiently verifiable (i.e., that it is not in NP).

• In the same way as we cannot categorically say that SAT is not efficiently solvable (i.e., that it is not in P).

• But we have reasons to believe that SAT is not in P.
An interesting but important note

• We cannot categorically say that TQBF is not efficiently verifiable (i.e., that it is not in NP).

• In the same way as we cannot categorically say that SAT is not efficiently solvable (i.e., that it is not in P).

• But we have reasons to believe that SAT is not in P.

• Because then we would be able to solve all NP-complete problems.
An interesting but important note

• We cannot categorically say that TQBF is not efficiently verifiable (i.e., that it is not in NP).

• In the same way as we cannot categorically say that SAT is not efficiently solvable (i.e., that it is not in P).

• But we have reasons to believe that SAT is not in P.
  • Because then we would be able to solve all NP-complete problems.

• Similarly we have reasons to believe that TQBF is not in NP.
NP

Interval Scheduling
Weighted Interval Scheduling

Subset Sum
Knapsack

Shortest Paths in Graphs
Minimum-cost Paths in Graphs
Testing Bipartiteness
Topological Sort

Vertex Cover
Independent Set

3SAT

SAT

Hamiltonian Cycle
Hamiltonian Path
Traveling Salesman

3-Colouring

Set Cover
Set Packing
3D-Matching

TQBF
PSPACE

Interval Scheduling
Weighted Interval Scheduling

Shortest Paths in Graphs
Minimum-cost Paths in Graphs
Testing Bipartiteness
Topological Sort

Vertex Cover
Independent Set

Set Cover
Set Packing
3D-Matching

3SAT
SAT

Hamiltonian Cycle
Hamiltonian Path
Traveling Salesman

3-Colouring

TQBF

TQBF is PSPACE-complete
Another NP-hard problem that is not in NP
Another NP-hard problem that is not in NP

- **Informally:** Given the description of an arbitrary computer program and an input to the program, determine if the program will terminate or not.
Another NP-hard problem that is not in NP

- **Informally:** Given the description of an arbitrary computer program and an input to the program, determine if the program will terminate or not.

- This is the **Halting Problem**, which is NP-hard but it is undecidable.
Another NP-hard problem that is not in NP

• **Informally:** Given the description of an arbitrary computer program and an input to the program, determine if the program will terminate or not.

• This is the **Halting Problem**, which is NP-hard but it is **undecidable**.
  
  • i.e., it cannot be solved in any amount of time on any computer.
Another NP-hard problem that is not in NP

- **Informally:** Given the description of an arbitrary computer program and an input to the program, determine if the program will terminate or not.

- This is the **Halting Problem**, which is NP-hard but it is **undecidable**.
  - i.e., it cannot be solved in any amount of time on any computer.

- More about that later!
Are all NP-complete problems equally hard?
Are all NP-complete problems equally hard?

We solved those in pseudopolynomial time.
Are all NP-complete problems equally hard?

We solved those in pseudopolynomial time. Could they be “easier” than SAT in some sense?
Strong vs Weak NP-hardness
Strong vs Weak NP-hardness

- A problem $P$ is strongly NP-hard if it remains NP-hard even when the numerical parameters in the input are given in unary representation.
Strong vs Weak NP-hardness

- A problem $P$ is strongly NP-hard if it remains NP-hard even when the numerical parameters in the input are given in unary representation.

- Otherwise, it is weakly NP-hard.
Strong vs Weak NP-hardness

- A problem $P$ is **strongly NP-hard** if it remains NP-hard even when the numerical parameters in the input are given in unary representation.

- Otherwise, it is **weakly NP-hard**.

- Weakly NP-hard problems admit pseudopolynomial algorithms.
Are all NP-complete problems equally hard?

- Interval Scheduling
- Weighted Interval Scheduling
- Subset Sum
- Knapsack
- Hamiltonian Cycle
- Hamiltonian Path
- Traveling Salesman
- Shortest Paths in Graphs
- Minimum-cost Paths in Graphs
- Testing Bipartiteness
- Topological Sort
- Vertex Cover
- Independent Set
- Set Cover
- Set Packing
- 3D-Matching
- 3SAT
- SAT
- 3-Colouring
Are all NP-complete problems equally hard?

Another way to compare: Approximate Solutions
Are all NP-complete problems equally hard?

Another way to compare: **Approximate Solutions**
More about that over the next two lectures!