Today’s Keywords

Reductions
NP-Completeness
P vs. NP

CLRS Readings: Chapter 34
HW9, HW10C due Thursday, December 5, 11pm
• Graphs, Reductions
• Written (LaTeX)
Monday, December 9, 7pm in Olsson 120

• Practice exam coming soon
• Review session likely the weekend before
• SDAC: Please sign-up for a time on December 9
A language $\mathcal{L} \in \text{NP}$ if there exists a deterministic polynomial-time “verifier” $\mathcal{R}$ such that

$$x \in \mathcal{L} \Leftrightarrow \exists w \in \{0,1\}^{\text{poly}(|x|)}: \mathcal{R}(x, w) = 1$$

A language $\mathcal{L} \in \text{P}$ if there exists a deterministic polynomial-time “solver” $\mathcal{R}$ such that

$$x \in \mathcal{L} \Leftrightarrow \mathcal{R}(x) = 1$$

If we can decide a problem in polynomial time, we can verify a solution to the problem in polynomial time:

$$\text{P} \subseteq \text{NP}$$

**Biggest open problem in computer science:** is this containment strict?

$$\text{P} = \text{NP} \text{ or } \text{P} \neq \text{NP}$$
Understanding the Landscape of NP

Question: What are the hard problems in NP?
  • Can we systematically characterize these?
  • Can we use insights from one problem to help solve another problem?

Strategy: Identify problems at least as “hard” as NP
  • If any of these “hard” problems can be solved in polynomial time, then all NP problems can be solved in polynomial time

A problem (or language) $B$ is NP-hard
  • $\forall A \in \text{NP}, A \leq_p B$
  • $A \leq_p B$ means $A$ reduces to $B$ in polynomial time
NP-Hardness

any NP problem

Solution for $A$

Map instances of problem $A$ to instances of $B$

polynomial time

Map solutions of problem $B$ to solutions of $A$

polynomial time

Solution for $B$

NP-hard problem

$A \leq_p B$: there is a polynomial-time reduction from $A$ to $B$
**NP-Hardness**

Any NP problem can be mapped to instances of another NP problem, say B, in polynomial time. Similarly, solutions of problem B can be mapped back to solutions of problem A in polynomial time. This is shown by the diagram:

- **Map instances of problem A to instances of B** in polynomial time.
- **Map solutions of problem B to solutions of A** in polynomial time.

**Very powerful:** if we can solve even one NP-hard problem in polynomial time, we can solve all of them!
**Question:** What are the **hardest** problems in NP?

- By definition, an efficient algorithm for an NP-hard problem implies an efficient algorithm for **every** NP problem
- **Answer:** the ones that are NP-hard (if there are any)

\[
\text{NP-complete} = \text{NP} \cap \text{NP-hard}
\]

“Complete” for NP in the sense that a solution to one implies a solution to **all**

- To show \( P = \text{NP} \), just need a **single** polynomial-time algorithm for a single NP-complete (or NP-hard) problem
- To show \( P \neq \text{NP} \), just need a **single** lower-bound that some NP problem cannot be solved in polynomial time
Question: What are the hardest problems in NP?

• By definition, an efficient algorithm for an NP-hard problem implies an efficient algorithm for every NP problem
• Answer: the ones that are NP-hard (if there are any)

\[ \text{NP-complete} = \text{NP} \cap \text{NP-hard} \]

“Complete” for NP in the sense that a solution to one implies a solution to all

• To show \( P = \text{NP} \), need a single NP-complete problem
• To show \( P \neq \text{NP} \), just need a single lower-bound that some NP problem cannot be solved in polynomial time

Not only do our existing techniques for proving lower bounds not work here, we are able to prove that most of our techniques will always fail...
NP-Completeness

NP-complete = NP ∩ NP-hard

To prove that a problem (or language) is NP-complete:

• Show it is in NP (i.e., construct a polynomial-time verifier)
• Show it is NP-hard (i.e., show every problem in NP reduces to it)

But there are a lot of problems in NP...
NP-Completeness

NP-complete = NP ∩ NP-hard

To prove that a problem (or language) is NP-complete:

• Show it is in NP (i.e., construct a polynomial-time verifier)
• Show it is NP-hard (i.e., show every problem in NP reduces to it)
  • Sufficient to show that another NP-hard problem reduces to it
  • Suppose $C$ is NP-hard and $C \leq_p B$; then for all $A \in NP$
    \[ A \leq_p C \leq_p B \Rightarrow A \leq_p C \]
• Challenge: coming up with a first NP-hard problem
3-SAT (Satisfiability)

Shown to be NP-hard by Cook and Levin (independently)

Given a 3-CNF formula (logical AND of clauses, each an OR of 3
variables), is there an assignment of true/false to each variable to make
the formula true (i.e., satisfy the formula)?

\[(x \lor y \lor z) \land (x \lor \bar{y} \lor y) \land (u \lor y \lor \bar{z}) \land (z \lor \bar{x} \lor u) \land (\bar{x} \lor \bar{y} \lor \bar{z})\]

Clause

Variables

\[x = \text{true} \]
\[y = \text{false} \]
\[z = \text{false} \]
\[u = \text{true} \]
$k$-Independent Set is NP-Complete

1. Show that it belongs to NP
2. Show it is NP-Hard
   • Show $3$-SAT $\leq_p k$-Independent Set
**Show:** For any graph $G$:

- There is a short witness (i.e., proof) that $G$ has a $k$-independent set
- The proof can be checked efficiently (in polynomial time)

**Witness for $G$:** $S = \{A, C, E, G, H, J\}$
(nodes in the $k$-independent set)

**Checking the witness:**
- Check that $|S| = k$
- Check that every edge is incident on at most one node in $S$

**Total time:** $O(|E| + |V|) = \text{poly}(|V| + |E|)$
$k$-Independent Set is NP-Complete

1. Show that it belongs to NP

2. Show it is NP-Hard
   - Show $3$-SAT $\leq_p k$-Independent Set
3-SAT \leq_p k\text{-Independent Set}

3-SAT

\((x \lor y \lor z) \land (x \lor \overline{y} \lor y) \land (u \lor y \lor \overline{z})\)

\(x = \text{true}\)
\(y = \text{false}\)
\(z = \text{false}\)
\(u = \text{true}\)

\(k\)-independent set

Map instances of problem \(A\) to instances of \(B\)

\text{polynomial time}

Map solutions of problem \(B\) to solutions of \(A\)

\text{polynomial time reduction}
For each clause, construct a triangle graph with its three variables as nodes.
Add an edge between each node and its negation.

Let $k = \text{number of clauses}$

Claim. There is a $k$-independent set in this graph if and only if there is a satisfying assignment.
Suppose there is a $k$-independent set $S$ in this graph $G$

- By construction of $G$, at most one node from each triangle is in $S$
- Since $|S| = k$ and there are $k$ triangles, each triangle contributes one node
- If a variable $x$ is selected in one triangle, then $\bar{x}$ is never selected in another triangle (since each variable is connected to its negation)
- There are no contradicting assignments, so can set variable chosen in each triangle to “true”; satisfying assignment by construction
3-SAT $\leq_P k$-Independent Set

\[(x \lor y \lor z) \land (x \lor \overline{y} \lor y) \land (u \lor y \lor \overline{z}) \land (z \lor \overline{x} \lor u) \land (\overline{x} \lor \overline{y} \lor \overline{z})\]

Suppose there is a satisfying assignment to the formula

- At least one variable in each clause must be true
- Add the node to that variable to the set $S$
- There are $k$ clauses, so set $S$ has exactly $k$ nodes
- If we use $x$ in any clause, we will never use $\overline{x}$, so there are no edges among the nodes in $S$

$x = \text{true}$

$y = \text{false}$

$z = \text{false}$

$u = \text{true}$
3-SAT $\leq_p k$-Independent Set

3-SAT

$$(x \lor y \lor z) \land (x \lor \overline{y} \lor y) \land (u \lor y \lor z)$$

$x = \text{true}$
$y = \text{false}$
$z = \text{false}$
$u = \text{true}$

Map instances of problem $A$ to instances of $B$

polynomial time

$k$-independent set

Map solutions of problem $B$ to solutions of $A$

polynomial time

polynomial-time reduction
$k$-Independent Set is NP-Complete

1. Show that it belongs to NP
2. Show it is NP-Hard
   - Show $3$-SAT $\leq_p k$-independent set
Max Independent Set $\leq_p$ Min Vertex Cover

$k$-independent set

\[ O(1) \text{ time} \]

Map instances of problem $A$ to instances of $B$

$k$-vertex cover

\[ O(|V|) \text{ time} \]

Map solutions of problem $B$ to solutions of $A$

Reduction
**k-Vertex Cover is NP-Complete**

1. Show that it belongs to NP
   - Given a candidate cover, check that every edge is covered

2. Show it is NP-Hard
   - Show $k$-independent set $\leq_p k$-vertex cover
$k$-Clique Problem

**Clique:** A complete subgraph

**$k$-Clique problem:** given a graph $G$ and a number $k$, is there a clique of size $k$?
$k$-Clique is NP-Complete

1. Show that it belongs to NP
   • Give a polynomial time verifier

2. Show it is NP-Hard
   • Give a reduction from a known NP-Hard problem
   • We will show $3$-SAT $\leq_p k$-clique
**Show:** For any graph $G$:

- There is a short witness (i.e., proof) that $G$ has a $k$-clique
- The proof can be checked efficiently (in polynomial time)

Suppose $k = 4$

**Witness for $G$:** $S = \{B, D, E, F\}$ (nodes in the $k$-clique)

**Checking the witness:**

- Check that $|S| = k$
- Check that every pair of nodes in $S$ share an edge

Total time: $O(|V|^2) = \text{poly}(|V| + |E|)$
$k$-Clique is NP-Complete

1. Show that it belongs to NP
   • Give a polynomial time verifier

2. Show it is NP-Hard
   • Give a reduction from a known NP-Hard problem
   • We will show $3$-SAT $\leq_p k$-clique
3-SAT $\leq_p k$-Clique

3-SAT

$$(x \lor y \lor z) \land (x \lor \overline{y} \lor y) \land (u \lor y \lor \overline{z})$$

$x = \text{true}$

$y = \text{false}$

$z = \text{false}$

$u = \text{true}$

$k$-clique

Map instances of problem $A$ to instances of $B$

polynomial time

Map solutions of problem $B$ to solutions of $A$

polynomial time

polynomial-time reduction
For each clause, introduce a node for each of its three variables
Add an edge from each node to all non-contradictory nodes in the other clauses (i.e., to all nodes that is not the negation of its own variable)
Let $k = \text{number of clauses}$

**Claim.** There is a $k$-clique in this graph if and only if there is a satisfying assignment
3-SAT $\leq_p k$-Clique

\[(x \lor y \lor z) \land (x \lor \bar{y} \lor y) \land (u \lor y \lor \bar{z}) \land (z \lor \bar{x} \lor u) \land (\bar{x} \lor \bar{y} \lor \bar{z})\]

Suppose there is a \textit{k-clique} in this graph

- There are no edges between nodes for variables in the same clause, so \textit{k-clique} must contain one node from each clause
- Nodes in clique cannot contain variable and its negation
- Nodes in clique must then correspond to a satisfying assignment
3-SAT \leq_p k\text{-Clique}

\[(x \lor y \lor z) \land (x \lor \bar{y} \lor y) \land (u \lor y \lor \bar{z}) \land (z \lor \bar{x} \lor u) \land (\bar{x} \lor \bar{y} \lor \bar{z})\]

Suppose there is a **satisfying assignment** to the formula

- For each clause, choose one node whose value is true
- There are \(k\) clauses, so this yields a collection of \(k\) nodes
- Since the assignment is consistent, there is an edge between every pair of nodes, so this constitutes a \(k\)-clique
3-SAT $\leq_p k$-Clique

3-SAT

$$(x \lor y \lor z) \land (x \lor \overline{y} \lor y) \land (u \lor y \lor \overline{z})$$

Map instances of problem $A$ to instances of $B$ in polynomial time

Map solutions of problem $B$ to solutions of $A$ in polynomial time

$x = \text{true}$

$y = \text{false}$

$z = \text{false}$

$u = \text{true}$

$k$-clique

polynomial-time reduction
$k$-Clique is NP-Complete

1. Show that it belongs to NP
   • Give a polynomial time verifier

2. Show it is NP-Hard
   • Give a reduction from a known NP-Hard problem
   • We will show 3-SAT $\leq_p k$-clique
Bonus Material: Coping with NP-Hardness

Material from subsequent slides will **not** be on the exam
Many optimization problems that come up in practice are NP-complete

What do we do?

**Approach 1:** Find an algorithm that gives nearly-optimal solutions
Greedy Vertex Cover

**Goal:** Find a set of nodes such that every edge is incident on one of the nodes

**Greedy choice:** Node with highest degree (e.g., node that covers the most edges)
Greedy Vertex Cover

**Goal:** Find a set of nodes such that every edge is incident on one of the nodes

**Greedy choice:** Node with highest degree (e.g., node that covers the most edges)
**Greedy Vertex Cover**

**Goal:** Find a set of nodes such that every edge is incident on one of the nodes

**Greedy choice:** Node with highest degree (e.g., node that covers the most edges)

Greedy approach?
Greedy Vertex Cover

Goal: Find a set of nodes such that every edge is incident on one of the nodes

Greedy approach?

Greedy choice: Node with highest degree (e.g., node that covers the most edges)
Greedy Vertex Cover

**Goal:** Find a set of nodes such that every edge is incident on one of the nodes

**Greedy choice:** Node with highest degree (e.g., node that covers the most edges)

**Greedy approach?**
Greedy Vertex Cover

**Goal:** Find a set of nodes such that every edge is incident on one of the nodes

**Greedy choice:** Node with highest degree (e.g., node that covers the most edges)
Greedy Vertex Cover

Goal: Find a set of nodes such that every edge is incident on one of the nodes

Greedy choice: Node with highest degree (e.g., node that covers the most edges)
**Goal:** Find a set of nodes such that every edge is incident on one of the nodes

**Greedy choice:** Node with highest degree (e.g., node that covers the most edges)
Greedy Vertex Cover

Goal: Find a set of nodes such that every edge is incident on one of the nodes

Greedy choice: Node with highest degree (e.g., node that covers the most edges)
**Greedy Vertex Cover**

**Goal:** Find a set of nodes such that every edge is incident on one of the nodes.

**Greedy approach?**

**Greedy choice:** Node with highest degree (e.g., node that covers the most edges)
**Greedy Vertex Cover**

**Goal:** Find a set of nodes such that every edge is incident on one of the nodes

**Greedy approach?**

**Greedy choice:** Node with highest degree (e.g., node that covers the most edges)
**Goal:** Find a set of nodes such that every edge is incident on one of the nodes

**Greedy choice:** Node with highest degree (e.g., node that covers the most edges)

**Size of vertex cover:** 5

In this case, actually **optimal**!
Greedy Vertex Cover

But not always optimal...

Graph $G$  
Optimal  
Greedy
Greedy Vertex Cover

But is it “good enough?”

How do we measure good enough?

Let $\text{OPT}(G)$ denote the size of the minimum vertex cover in $G$ and $|A(G)|$ be the size of the cover output by algorithm $A$

Define the approximation factor of $A$ to be

$$\text{ApproxFactor}(A) = \frac{|A(G)|}{\text{OPT}}$$

The larger this value is, the worse the quality of the approximation

(\textbf{Goal:} as close to 1 as possible)
But is it “good enough?”

How do we measure good enough?

Let $\text{OPT}(G)$ denote the size of the minimum vertex cover in $G$ and $|A(G)|$ be the size of the cover output by algorithm $A$

Define the approximation factor of $A$ to be

$$\text{ApproxFactor}(A) = \frac{|A(G)|}{\text{OPT}}$$

**Theorem.** The greedy algorithm for vertex cover achieves an approximation factor of $\Omega(\log|V|)$

Not that great... quality of solution is worse for large instances
Approximate Vertex Cover

Goal: Obtain a 2-approximation (i.e., vertex cover that is at most twice as large as the optimal)
Approximate Vertex Cover

**Goal:** Obtain a 2-approximation (i.e., vertex cover that is at most twice as large as the optimal)

Consider an edge $e = (u, v) \in E$
- Optimal vertex covering must contain either $u$ or $v$
- **Our approach:** take both of them!
  - Add $u, v$ to cover
  - Remove all edges incident on $u$ and $v$
  - Repeat until no edges remain
Approximate Vertex Cover

**Goal:** Obtain a 2-approximation (i.e., vertex cover that is at most twice as large as the optimal)

Consider an edge $e = (u, v) \in E$

- Optimal vertex covering must contain either $u$ or $v$
- **Our approach:** take both of them!
  - Add $u, v$ to cover
  - Remove all edges incident on $u$ and $v$
  - Repeat until no edges remain
Approximate Vertex Cover

**Goal:** Obtain a 2-approximation (i.e., vertex cover that is at most twice as large as the optimal)

Consider an edge \( e = (u, v) \in E \)

- Optimal vertex covering must contain either \( u \) or \( v \)
- **Our approach:** take **both** of them!
  - Add \( u, v \) to cover
  - Remove all edges incident on \( u \) and \( v \)
  - Repeat until no edges remain
**Goal:** Obtain a 2-approximation (i.e., vertex cover that is at most twice as large as the optimal)

Consider an edge $e = (u, v) \in E$

- Optimal vertex covering must contain either $u$ or $v$
- **Our approach:** take both of them!
  - Add $u, v$ to cover
  - Remove all edges incident on $u$ and $v$
  - Repeat until no edges remain
Approximate Vertex Cover

**Goal:** Obtain a 2-approximation (i.e., vertex cover that is at most twice as large as the optimal)

Consider an edge $e = (u, v) \in E$
- Optimal vertex covering must contain either $u$ or $v$
- **Our approach:** take both of them!
  - Add $u, v$ to cover
  - Remove all edges incident on $u$ and $v$
  - Repeat until no edges remain
Approximate Vertex Cover

Goal: Obtain a 2-approximation (i.e., vertex cover that is at most twice as large as the optimal)

Consider an edge $e = (u, v) \in E$

- Optimal vertex covering must contain either $u$ or $v$
- **Our approach:** take both of them!
  - Add $u, v$ to cover
  - Remove all edges incident on $u$ and $v$
  - Repeat until no edges remain
**Approximate Vertex Cover**

**Goal:** Obtain a 2-approximation (i.e., vertex cover that is at most twice as large as the optimal)

Consider an edge $e = (u, v) \in E$

- Optimal vertex covering must contain either $u$ or $v$
- **Our approach:** take both of them!
  - Add $u, v$ to cover
  - Remove all edges incident on $u$ and $v$
  - Repeat until no edges remain
Approximate Vertex Cover

**Goal:** Obtain a 2-approximation (i.e., vertex cover that is at most twice as large as the optimal)

Consider an edge $e = (u, v) \in E$

- Optimal vertex covering must contain either $u$ or $v$
- **Our approach:** take both of them!
  - Add $u, v$ to cover
  - Remove all edges incident on $u$ and $v$
  - Repeat until no edges remain

Size of vertex cover: 6
Size of optimal vertex cover: 5
Consider an edge $e = (u, v) \in E$

- Optimal vertex covering must contain either $u$ or $v$
- Our approach: take both of them!
  - Add $u, v$ to cover
  - Remove all edges incident on $u$ and $v$
  - Repeat until no edges remain

**Theorem.** The approximate algorithm for vertex cover achieves an approximation factor of 2
Coping with NP-Hardness

Many optimization problems that come up in practice are NP-complete

What do we do?

**Approach 1:** Find an algorithm that gives nearly-optimal solutions

**Question:** Can we do better than a 2-approximation?

Slightly... there is an algorithm that achieves a \(2 - O(1/\sqrt{\log|V|})\) approximation

**Open Problem:** Obtain a \((2 - \epsilon)\)-approximation for constant \(\epsilon > 0\)

**Question:** What’s the best we could hope for? Can we have a 1.00001-approximation?

Unlikely, computing a \(\sqrt{2} \approx 1.41\) approximation is NP-hard (Khot-Minzer-Safra, 2018)

**Earlier lower bounds:** \(7/6 \approx 1.17\) (Håstad, 1997), \(10\sqrt{5} - 21 \approx 1.36\) (Dinur-Safra, 2005)
Many optimization problems that come up in practice are NP-complete

What do we do?

**Approach 1:** Find an algorithm that gives nearly-optimal solutions

**Question:** Can we do better than a 2-approximation?

A slightly better than 2-approximation is

\[ \left(1 - \frac{1}{\log|V|}\right) \]

**Open Problem:** Obtain a \((2 - \varepsilon)\)-approximation for constant \(\varepsilon > 0\)

**Question:** What’s the best we could hope for? Can we have a 1.00001-approximation?

Likely, computing a \(\sqrt{2} \approx 1.41\) approximation is NP-hard (Khot-Minzer-Safra, 2018)

**Earlier lower bounds:** 7/6 \(\approx 1.17\) (Håstad, 1997), \(10\sqrt{5} - 21 \approx 1.36\) (Dinur-Safra, 2005)
Coping with NP-Hardness

Many optimization problems that come up in practice are NP-complete

What do we do?

**Approach 1:** Find an algorithm that gives nearly-optimal solutions

**Approach 2:** For small instances, solve using brute force or dynamic programming
Can also improve (expected) run-time using heuristics
Coping with NP-Hardness

Many optimization problems that come up in practice are NP-complete

What do we do?

**Approach 1:** Find an algorithm that gives nearly-optimal solutions

**Approach 2:** For small instances, solve using brute force or dynamic programming
   Can also improve (expected) run-time using heuristics

**Approach 3:** Special cases of the problems can be tractable
When the graph is a tree, vertex cover can be solved using **dynamic programming**:

- Consider the root node
- Either it is part of the cover or all of its children are part of the cover

Solve vertex cover on subtrees and take the minimum
Coping with NP-Hardness

Many optimization problems that come up in practice are NP-complete

What do we do?

**Approach 1:** Find an algorithm that gives nearly-optimal solutions

**Approach 2:** For small instances, solve using brute force or dynamic programming
   Can also improve (expected) run-time using heuristics

**Approach 3:** Special cases of the problems can be tractable
   (see also parameterized complexity)