Lecture 1: *Computational Complexity 101*

Advanced Algorithms

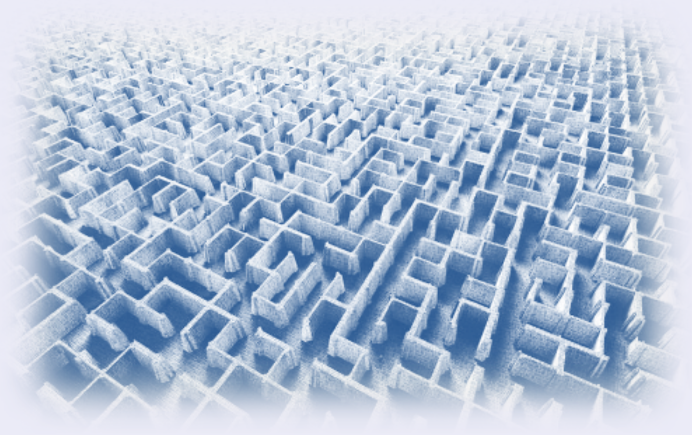
Sid Chi-Kin Chau

Australian National University

sid.chau@anu.edu.au

October 9, 2022

⋘Computational Complexity 101⋙



Algorithms vs. Heuristics

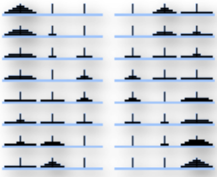
Definition (Algorithms)

- Based on good understanding of performance and optimality with rigorous analysis
- Supported by theoretical evidence and universal results
- E.g., approximation algorithms with proven approximation ratios

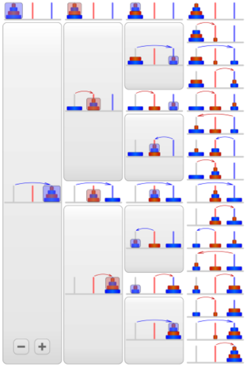
Definition (Heuristics)

- Based on mostly guess work
- Empirically driven, but with no universal result and not backed by rigorous analysis
- E.g., meta-heuristics (e.g. genetic algorithms)

How to solve Towers of Hanoi? Is it Hard?



Towers of Hanoi
by divide-and-conquer



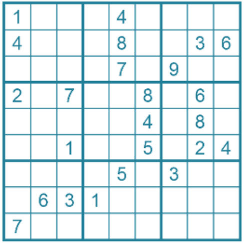
- Let Hanoi$[n, r]$ be the algorithm that moves the smallest n disks from the left rod to rod $r \in \{\text{middle}, \text{right}\}$

Algorithm Hanoi$[n, r]$

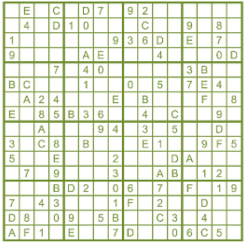
- If $n > 1$ then
  - Hanoi$[n-1, \{\text{middle}, \text{right}\} \backslash \{r\}]$
  - Move the remaining n-th disk to rod r
  - Hanoi$[n-1, r]$
- Else
  - Move the smallest disk to rod r

- Running time of Hanoi$[n, r]$ is $\Omega(2^n)$
- Is Towers of Hanoi a hard problem to solve?

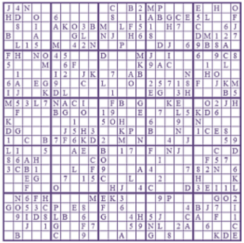
How to solve Sudoku? Is it Hard?



3^2x3^2 Sudoku



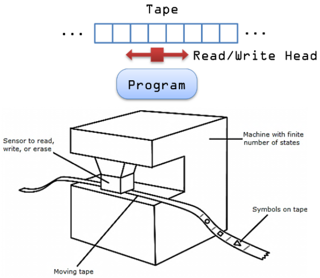
4^2x4^2 Sudoku



5^2x5^2 Sudoku

- Can you solve a $3^2 \times 3^2$ Sudoku? How about solving a $n^2 \times n^2$ Sudoku?
- Can you check a given $3^2 \times 3^2$ Sudoku solution? How about checking a $n^2 \times n^2$ Sudoku solution?

Mathematical Formalization of Algorithms



A Conceptual Turing Machine

Figure: Watch a real-life Turing machine: 🎬 `https://youtu.be/E3keLeMwfHY`

- Algorithms are represented by mechanical operations (i.e. program) of a universal problem solver (i.e. Turing machine)
  - An algorithm is a mapping from a tape of input sequence of symbols (i.e. problem instance) to an output sequence of symbols (i.e. answer)
  - Universal Turing Machine - Can simulate any programs and the tape is also consisted of an input program

Abstract Representation of Problems

Definition (Problems, Instances, Languages)

- We fix a certain encoding scheme $\text{Enc}(\cdot)$ that maps a linguistic representation of a sentence to a binary representation (e.g. ASCII)
- A problem is represented by a subset of all (finite or infinite) binary strings $\subseteq \{0,1\}^*$
  - An instance of a problem is translated to a binary string by $\text{Enc}(\cdot)$, denoted by $\mathcal{I} \in \{0,1\}^*$
  - Multiple instances may have the same answer
  - The simplest of answer is binary (yes/no) for a decision problem
  - A decision problem can be represented by all the "yes" instances, denoted by a subset $\mathcal{L} \subseteq \{0,1\}^*$
  - $\mathcal{L}$ is also called a "language"

- ⚠ Caveat: Don't need to pay attention to a particular encoding scheme
  - We focus on an abstract representation of a problem

Example: Decision Problem

Example (Problem: isPrime(X))

- Enc("isPrime(23)") $\mapsto$ 101000100100
- Enc("isPrime(25)") $\mapsto$ 000010110011
- Enc("isPrime(27)") $\mapsto$ 1100010110011
- Enc("isPrime(29)") $\mapsto$ 0000000100010
- $\cdots\cdots\cdots$

- Problem isPrime is represented by $\mathcal{L}_{\textsf{isPrime}} = \{101000100100, 0000000100010, \ldots\}$
- Let $\overline{\mathcal{L}}_{\textsf{isPrime}} \triangleq \{0,1\}^* \backslash \mathcal{L}_{\textsf{isPrime}} = \{000010110011, 1100010110011, \ldots\}$

- Why do we need an abstract representation of a problem?
  - Need a universal formalism, independent of any linguistic/programming languages

Abstract Representation of Problems

- We can classify problems by their abstract representation
  - **Which problems are easy or hard?**
  - Which problems need a lot of memory space?
  - Which problems can be solved by a quantum computer?

Definition

- Denote a realization of Turing Machine by $\mathcal{M}$, which implements an algorithm for problem $\mathcal{L}$
  - Running time of $\mathcal{M}$ should be polynomial in the input size ($|\mathcal{I}|$, or the $\#$bits to represent $\mathcal{I}$)
- If you claim that you know an answer of a decision problem (yes/no), then you should be able to present a proof
  - Denote an instance of $\mathcal{L}$ by $\mathcal{I}$, a witness (a proof of yes) by w
  - ⚠ Caveat: A witness does not need to be a solution for $\mathcal{I}$
  - $\mathcal{M}(\mathcal{I}, w)$ should return TRUE, if w is a witness for $\mathcal{I}$
  - Example: $\mathcal{M}(\text{isComposite}(21), 3 \times 7)$ returns TRUE

What are Hard Problems?

Definition (Class NP)

- Let $|\mathcal{I}|$ = #bits to represent $\mathcal{I}$ and $|w|$ = #bits to represent w
- Define a class of problems called NP: For all $\mathcal{L} \in$ NP, there exist a polynomial-time bound $\mathcal{M}$ and a polynomial function $p(\cdot)$, such that
  - If $\mathcal{I} \in \mathcal{L}$, then there exists a witness w where $|w| \leq p(|\mathcal{I}|)$, such that $\mathcal{M}(\mathcal{I}, w)$ returns TRUE, and
  - If $\mathcal{I} \notin \mathcal{L}$, then for any witness w where $|w| \leq p(|\mathcal{I}|)$, $\mathcal{M}(\mathcal{I}, w)$ returns FALSE

- NP stands for Non-deterministic Polynomial-time
- NP are the problems that can be verified efficiently when given a proof
- Question: Is $\mathcal{L}_{\text{isPrime}} \in$ NP? Is $\mathcal{L}_{\text{isComposite}} \in$ NP?

What are Hard Problems?

Definition (Class co-NP)

- Define a class of problems called co-NP: For all $\mathcal{L} \in$ co-NP, there exist a polynomial-time bound $\mathcal{M}$ and a polynomial function $p(\cdot)$, such that
  - If $\mathcal{I} \notin \mathcal{L}$, then there exists a witness w where $|w| \le p(|\mathcal{I}|)$, such that $\mathcal{M}(\mathcal{I}, w)$ returns TRUE, and
  - If $\mathcal{I} \in \mathcal{L}$, then for any witness w where $|w| \le p(|\mathcal{I}|)$, $\mathcal{M}(\mathcal{I}, w)$ returns FALSE

- co-NP are problems that can be verified efficiently when given a counter-example
- $\mathcal{L} \in$ co-NP $\iff \overline{\mathcal{L}} \in$ NP
- Question: Is $\mathcal{L}_{\text{isPrime}} \in$ co-NP? Is $\mathcal{L}_{\text{isComposite}} \in$ co-NP?

What are Hard Problems?

Definition (Class P)

- Define a class of problems called P: For all $\mathcal{L} \in \text{P}$, there exists a polynomial-time bound $\mathcal{M}$, such that
  - If $\mathcal{I} \notin \mathcal{L}$, then without any witness, $\mathcal{M}(\mathcal{I}, ?)$ returns TRUE, and
  - If $\mathcal{I} \in \mathcal{L}$, then without any witness, $\mathcal{M}(\mathcal{I}, ?)$ returns FALSE

- P are problems that can generate a proof or a counter-example efficiently
- $\text{P} \subseteq \text{co-NP} \cap \text{NP}$
- Question: Is $\mathcal{L}_{\text{isPrime}} \in \text{P}$?
- What are Hard Problems?
  - Conventional wisdom of computer scientists is that any problems that are not in P are hard
  - Otherwise, it is not time efficient to solve the problem (i.e. generating a proof or a counter-example)
  - Is that true? Million-dollar question that makes you immortal: $\text{P} \stackrel{?}{=} \text{NP}$

Solve New Problems from Known Problems

Definition (Reduction)

- Consider two decision problems $\mathcal{L}_1, \mathcal{L}_2 \in \mathsf{NP}$
  - There may exist a polynomial-time bound $\mathcal{M}$ such that it maps an instance $\mathcal{I}_1 \in \mathcal{L}_1$ to an instance $\mathcal{I}_2 \in \mathcal{L}_2$
- A *polynomial-time reduction* from $\mathcal{L}_1$ to $\mathcal{L}_2$ if there exists a polynomial-time bound $\mathcal{M}$, such that
  - If $\mathcal{I}_1 \in \mathcal{L}_1$, then $\mathcal{M}(\mathcal{I}_1) \in \mathcal{L}_2$
- If $\mathcal{L}_1$ can be polynomial-time reduced to $\mathcal{L}_2$, we write $\mathcal{L}_1 \preceq \mathcal{L}_2$

- Intuitively, if we can solve every $\mathcal{I}_2 \in \mathcal{L}_2$, then we can also solve every $\mathcal{I}_1 \in \mathcal{L}_1$ (but not necessarily vice versa). Namely, $\mathcal{L}_2$ is harder than $\mathcal{L}_1$

Example: Reduction from HamCyc to TSP

Definition

- TSP(C) (Travel Salesman Problem):
  - Given complete graph $\mathcal{G}$ of n vertices & non-negative edge costs
  - Decide if the minimum cost cycle that visits every vertex exactly once has a cost $\leq C$
- HamCyc (Hamiltonian Cycle Problem):
  - Given a graph $\mathcal{G}'$ (may be not complete) of n vertices,
  - Decide if there is a cycle that visits every vertex exactly once

- We can show HamCyc $\preceq$ TSP$(c \cdot n)$ for any constant $c > 1$
  - Given $\mathcal{G}'$, we construct $\mathcal{G}$ with the same set of vertices in $\mathcal{G}'$
    - ★ If $e \in \mathcal{G}'$, then $c_{\mathcal{G}}(e) = 1$, otherwise, $c_{\mathcal{G}}(e) = c \cdot n$
  - This is a reduction, because that
    - ★ If $\mathcal{G}'$ has a Hamiltonian cycle, then there exists a Hamiltonian cycle in $\mathcal{G}$ with a cost $\leq c \cdot n$
    - ★ If $\mathcal{G}'$ has no Hamiltonian cycle, then any Hamiltonian cycle in $\mathcal{G}$ will have a cost $> c \cdot n$

Are all Hard Problems equally Hard?

Definition (NP-hard and NP-complete)

- Define a class of problems called NP-hard:
  - If $\mathcal{L}' \preceq \mathcal{L}$ for any problem $\mathcal{L}' \in$ NP, then $\mathcal{L} \in$ NP-hard
- Define a class of problems called NP-complete :
  - If $\mathcal{L} \in$ NP-hard and $\mathcal{L} \in$ NP, then $\mathcal{L} \in$ NP-complete

- NP-complete $=$ NP $\cap$ NP-hard
- NP-hard refers to the problems that are at least as hard as any problem in NP
- NP-complete refers to the hardest problem in NP
- Which is the hardest problem in NP?

Hardness: Are all hard problems equally hard?

Definition (SAT (Boolean Satisfiability Problem))

- A SAT is to decide if a given Boolean expression (that combines Boolean variables with Boolean operators) is satisfiable (i.e. there exists an assignment of truth values to the variables to make entire expression true)

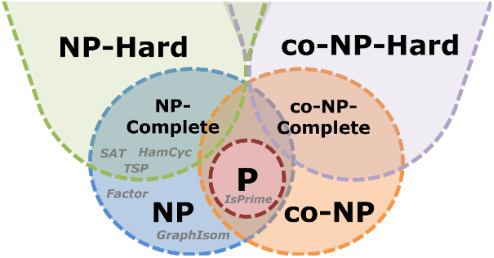
- E.g., decide if $\neg x_1 \vee (x_2 \wedge x_4) \vee (x_1 \wedge \neg x_3 \wedge x_4 \wedge x_5)$ is satisfiable

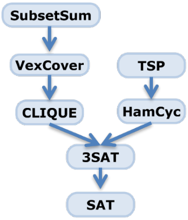
Theorem (Cook's Theorem)

- SAT *is* NP-complete

- Cook's theorem implies that all NP-hard problems in NP are as hard as SAT

Hierarchy of Hardness



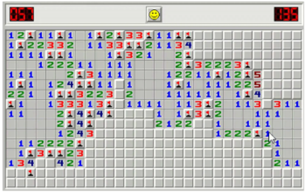


- ⚠ Unknown: P $\subsetneq$ co-NP $\cap$ NP? P$\neq$co-NP-complete$\neq$NP-complete? P$=$co-NP$=$NP?
- Chains of reductions for NP-complete problems
  - E.g., HamCyc $\preceq$ TSP and SAT $\preceq$ HamCyc
  - Hence, HamCyc and TSP are also NP-complete

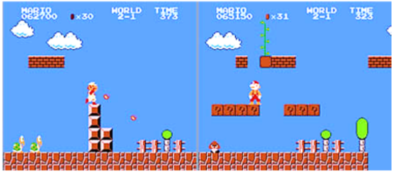
NP-Complete Problems

- 3SAT:
  - 3-literal satisfiability problem, where each clause is limited to at most three literals
  - E.g., $(\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee \neg x_2 \vee x_4)$
- CLIQUE:
  - Decide if there exists a clique (a complete sub-graph) of a certain size in a given graph
- VexCover:
  - Decide if there exists a subset of vertices of a certain size that include at least one endpoint of every edge of a given graph
- SubsetSum:
  - Decide if there exists a subset of numbers that sum to a certain value in a given set of numbers
- and many real-life NP-complete problems ...

Hard Recreational Games 🎮



Minesweeper



Super Mario

- Sudoku is NP-complete
- Minesweeper is NP-complete
  - Decide if there exist consistent mine locations for the uncovered cells given a number of covered cells in a Minesweeper game
- SuperMario is NP-complete
  - Decide if there exists a game play to win a Super Mario game
  - Watch a proof for the NP-Hardness of SuperMario: 🎬 `https://youtu.be/oS8m9fSk-Wk`

Hard Problems: So What? 🤷‍♀️

Optimization Problems

- Often, we don't need to solve the hard problems **exactly**

Definition (NP-optimization)

- An optimization problem $\mathcal{X}$ has an objective function $f(\cdot)$, which maps every instance $\mathcal{I}$ and solution (or witness) $\mathcal{S}$ to a numerical value $f(\mathcal{I}, \mathcal{S})$
  - E.g., In TSP, $\mathcal{I}$ is a graph $\mathcal{G}$, $\mathcal{S}$ is a cycle in $\mathcal{G}$, $f(\cdot)$ measures the total cost of $\mathcal{S}$
- A minimization (or maximization) problem $\mathcal{X}$ is optimization problem $\mathcal{X}$ to find $\mathcal{S}$ such that $f(\mathcal{S}, \mathcal{I})$ is minimized (or maximized) for a given instance $\mathcal{I}$
- A minimization (or maximization) problem $\mathcal{X}$ is NP-optimization, if deciding there exists $\mathcal{S}$ such that $f(\mathcal{S}, \mathcal{I}) \leq C$ (or $f(\mathcal{S}, \mathcal{I}) \geq C$) is NP-hard

- We also call a NP-optimization problem NP-hard
  - Finding an optimal solution is harder than deciding if a solution of a certain value exists

Approximation Algorithms

- Finding an optimal solution to a NP-hard problem is difficult. How about approximation?

Definition (Approximation Ratio for Minimization Problem)

- Consider an NP-hard minimization problem $\mathcal{X}$ with an instance denoted by $\mathcal{I}$
  - Let $\mathsf{Opt}(\mathcal{I})$ be an optimal solution, and objective function be $f\big(\mathsf{Opt}(\mathcal{I})\big)$
  - Consider a polynomial-time algorithm $\mathcal{A}$ that produces a solution $\mathcal{A}(\mathcal{I})$
- Define minimization approximation ratio: $\alpha_n(\mathcal{A}) = \max_{\mathcal{I}:|\mathcal{I}|\le n} \dfrac{f\big(\mathcal{A}(\mathcal{I})\big)}{f\big(\mathsf{Opt}(\mathcal{I})\big)}$

Definition (Approximation Ratio for Maximization Problem)

- Consider an NP-hard maximization problem $\mathcal{X}$ with an instance denoted by $\mathcal{I}$
  - Consider a polynomial-time algorithm $\mathcal{A}$ that produces a solution $\mathcal{A}(\mathcal{I})$
- Define maximization approximation ratio: $\alpha_n(\mathcal{A}) = \min_{\mathcal{I}:|\mathcal{I}|\le n} \dfrac{f\big(\mathcal{A}(\mathcal{I})\big)}{f\big(\mathsf{Opt}(\mathcal{I})\big)}$

Approximation Algorithms

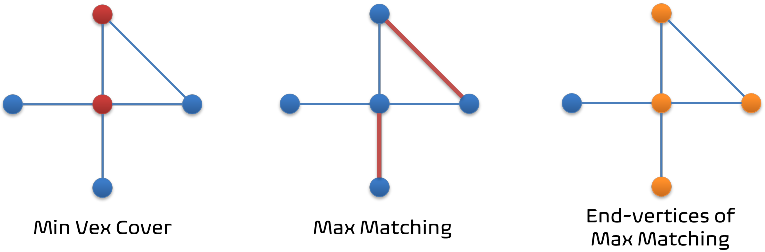
- We aim to find a polynomial-time **approximation algorithm** $\mathcal{A}$ with a good approximation ratio $\alpha_n(\mathcal{A})$ to bound the gap from an optimal solution
  - Minimization Problem: $f\big(\mathcal{A}(\mathcal{I})\big) \leq \alpha_n(\mathcal{A}) \cdot f\big(\mathsf{Opt}(\mathcal{I})\big)$
  - Maximization Problem: $f\big(\mathcal{A}(\mathcal{I})\big) \geq \alpha_n(\mathcal{A}) \cdot f\big(\mathsf{Opt}(\mathcal{I})\big)$
- $\alpha_n(\mathcal{A})$ depends on the input size $|\mathcal{I}|$
  - $\alpha_n(\mathcal{A})$ is the worst-case ratio considering all instances that are bounded by size n
- For a NP-hard problem, $\alpha_n(\mathcal{A}) = 1$ is impossible, unless P = NP
- However, can we come up with a polynomial-time algorithm $\mathcal{A}$, such that $\alpha_n(\mathcal{A})$ is sufficiently good? What is the best $\alpha_n(\mathcal{A})$ that we can achieve?

Approximation Algorithm: Vertex Cover

Example (VexCover)

- VexCover (Minimum Vertex Cover Problem):
  - Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, find a minimum subset of vertices $\tilde{\mathcal{V}} \subseteq \mathcal{V}$, such that every $e \in \mathcal{E}$ has one end-vertex in $\tilde{\mathcal{V}}$
- MaximalMatch (Maximal Matching Problem):
  - Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, find a maximal subset of edges $\tilde{\mathcal{E}} \subseteq \mathcal{E}$, such that no $e \in \tilde{\mathcal{E}}$ share the same vertex
- VexCover is NP-complete
- But MaximalMatch is easy
  - Is MaximalMatch in P?

Approximation Algorithm: Vertex Cover



Approximation Algorithm: Vertex Cover

Algorithm $\mathcal{A}_{\text{vxc}}$

- Solve MaximalMatch on $\mathcal{G}$, and the solution is denoted by $\tilde{\mathcal{E}}$
- Output the set of end-vertices in every $e \in \tilde{\mathcal{E}}$

Theorem (2-Approximability of VexCover)

$\mathcal{A}_{\text{vxc}}$ always outputs a vertex cover. The approximation ratio of $\mathcal{A}_{\text{vxc}}$ is $\alpha_n(\mathcal{A}_{\text{vxc}}) \leq 2$

Proof:

- Let Opt be a minimum set of vertex cover
- Consider an edge (u, v) in a maximal matching $\tilde{\mathcal{E}}$
- One of u, v must be in Opt, otherwise, (u, v) is not covered. Hence, $|\tilde{\mathcal{E}}| \leq |\text{Opt}|$
- The number of vertices output from $\mathcal{A}_{\text{vxc}}$ is $f(\mathcal{A}_{\text{vxc}}) \leq 2|\tilde{\mathcal{E}}|$. Hence, $f(\mathcal{A}_{\text{vxc}}) \leq 2|\text{Opt}|$

Approximation Algorithm: TSP

Example (TSP)

- Recall that HamCyc $\preceq$ TSP$(c \cdot n)$ for any constant $c > 1$

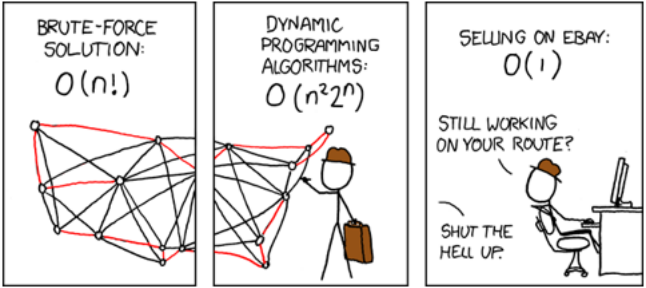
Theorem (Inapproximability of TSP)

There exists no polynomial-time $\mathcal{A}$ for TSP *such that $\alpha_n(\mathcal{A})$ is a constant c, unless* P $=$ NP

Proof:

- Let Opt be a minimum cycle in TSP
- Use contradiction – suppose $\alpha_n(\mathcal{A}) = c$, and use reduction HamCyc $\preceq$ TSP$(c \cdot n)$
- Then there exists a polynomial-time algorithm $\mathcal{A}$ that produces a cycle in TSP with $f(\mathcal{A}) \leq c \cdot f(\text{Opt})$
- By reduction HamCyc $\preceq$ TSP$(c \cdot n)$, if there exists a Hamiltonian cycle, $f(\text{Opt}) = n$
- Hence, $f(\mathcal{A}) \leq c \cdot n \iff$ there exists a Hamiltonian cycle
- This gives a polynomial-time algorithm to solve HamCyc and hence, P $=$ NP

TSP is Really Hard 🤔



- TSP is not only NP-hard, but also inapproximable within any constant approximation ratio
  - In fact, it is inapproximable within any polynomial approximation ratio
- Is it inapproximable in practice?
  - Not in specific settings, e.g. in an Euclidean space

Follow-up Questions ✋

- Is NP-hardness an accurate description of computational hardness?
  - No, but it is close. Turing Machine, with no memory access bottleneck, is not a realistic computer model
  - A more realistic model is von Neumann model, which has a memory hierarchy with different access speeds and capacities

- Why do we expect $P \neq NP$, even though we may not be able to prove it?
  - "If $P = NP$, then the world would be a profoundly different place than we usually assume it to be. There would be no special value in "creative leaps," no fundamental gap between solving a problem and recognizing the solution once it's found. Everyone who could appreciate a symphony would be Mozart; everyone who could follow a step-by-step argument would be Gauss; everyone who could recognize a good investment strategy would be Warren Buffett." - **Scott Aaronson**
  - Cryptography critically relies on efficiently verifiable but intractable problems. If $P = NP$, then there will be no asymmetric cryptography, or one-way hash function

Follow-up Questions ✋

- Why do we bother NP-hard problems, if powerful quantum computers are coming soon?
  - Only two useful quantum algorithms can solve classical (non-quantum) problems: Shor algorithm and Gover algorithm – both are insufficient to solve NP-hard problems in general
  - Quantum heuristics are applied to solve classical problems. But they are not proven to be better than classical approximation algorithms - in fact, there are a lot of skepticisms
  - Quantum computers are better for solving quantum problems

- Why do we bother approximation algorithms, if machine learning can solve many hard problems?
  - Machine learning is mostly heuristics – do not have universal results on approximation ratios
  - Many empirical results are not replicable, or are specific to particular experimental data
  - Machine learning can not be extended to a problem of arbitrary size – it needs a lot of training data, which is impractical for large problems
  - But there are provable machine learning algorithms for limited applications

References

Reference Materials

- Introduction to Algorithms (Cormen, Leiserson, Rivest, Stein), 4th ed, MIT Press
  - Chapters 34-35
- Approximation Algorithms (V. Vazirani), Springer
  - Chapter 1, Appendix A

Recommended Materials

- Survey of P $\stackrel{?}{=}$ NP (Scott Aaronson), `https://www.scottaaronson.com/papers/pnp-kindle.pdf`
- 🎬 Watch online tutorial videos: `https://youtube.com/playlist?list=PLlwsleWT767dnN25K_QgvdKkovK_t4K6-`

Related Courses in Advanced Algorithms

Related Courses in Other Universities

- Harvard (CS 224) Advanced Algorithms:
  `https://people.seas.harvard.edu/~cs224/fall14/lec.html`
- CMU (15-850) Advanced Algorithms:
  `http://www.cs.cmu.edu/~15850/`
- Princeton (COS 521) Advanced Algorithm Design:
  `https://www.cs.princeton.edu/courses/archive/fall18/cos521/`
- MIT (6.854J) Advanced Algorithms:
  `https://ocw.mit.edu/courses/6-854j-advanced-algorithms-fall-2005/`
- Stanford (CS 361B) Advanced Algorithms:
  `https://web.stanford.edu/class/cs361b/`