P and NP

Sam McCauley

May 5, 2025

Welcome Back!

- PS7 and Midterm 2 back by Thursday
- Last problem set is this week
- *Last call:* please email me if you want to take the final early
    - If you are not sure (e.g. you're on a sports team and don't know if you'll qualify for a tournament) please let me know
- Questions?

Shifting Focus

Algorithmic Design Paradigms

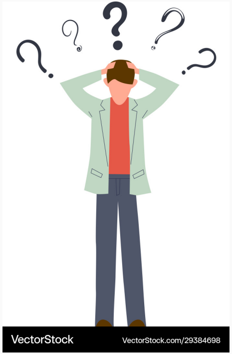
- Greedy Algorithms
- Divide and Conquer
- Dynamic Programming
- Network Flow
- $\Leftarrow$ we are here!

Where we are

- We saw increasingly powerful techniques to solve computational problems
- Are there *limits*? Can we keep coming up with increasingly powerful techniques to eventually solve any problem?
- Answer: We don’t know.

We don't know

- Most other sciences have been around for hundreds of years
- Many of their fundamental questions have been either fully addressed, or are at least well-understood
- Not really the case in computer science. I don't know if:
  - Network flows can be solved on a flow network with m edges in $O(m)$ time (best we used: $O(nm)$)
  - Edit distance can be solved for two length-n strings in $O(n)$ time (we saw $O(n^2)$)
  - Knapsack with n items can be solved in $O(n)$ time, no matter how large C is



Shifting Focus

- Rest of the class: *lower bounds:* what problems are impossible to solve efficiently?

- Most of these lower bounds are *conditional:* I can only say that they are probably impossible to solve efficiently

Polynomial Time

- What problems can a computer solve in *polynomial time*?
  - As we saw earlier: I mean polynomial in the size of the input
- What problems can a computer (probably) not solve in polynomial time?
- (Pseudopolynomial does not count: we will see that Knapsack probably cannot be solved in polynomial time)

Technical Setup

- Focus on *decision problems*—problems with a “yes” or “no” ansewr
  - Does this directed graph have a topological order?
  - Is this graph bipartite?
  - Do these two strings have edit distance at most 10?
  - Does this flow network have a maximum flow of at least 20?
- Most computational problems have a decision analog like this
- If you want the exact solution, can binary search for the optimal value

P and NP

Class P

- Definition: P is the class of decision problems that can be solved in polynomial time in the size of the input
- Some problems in P:
  - Edit distance
  - Max flow
  - Bipartite matching
  - Knapsack?
    - We have not seen a polynomial-time algorithm! (And we won't.)
    - We can't say that Knapsack is in P. And soon we will say: it is probably not in P.

Class NP

- Definition: Class of problems that can be *verified* in polynomial time

- (Does not stand for "not polynomial" or anything like that.)

- More formally: if I give you helpful information, say a proposed solution, you can *check* that it is correct in polynomial time

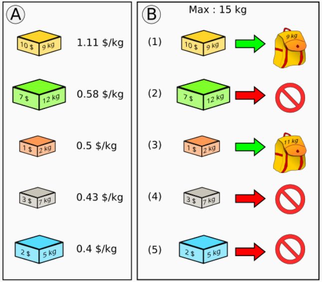
Class NP Example

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

- Sudoku may be hard to solve(?)
- But if I give you the solution, it's easy to *verify*
- Sudoku is in NP!

Class NP Example 2



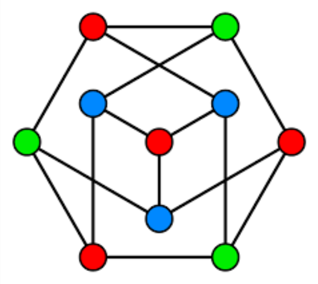
- The dynamic program for knapsack is pseudopolynomial
- But if I give you the solution, it's easy to *verify*
- Knapsack is in NP!

Class NP

- Class of problems that can be *verified* in polynomial time
- More formal definition (you do not need to know): there exists a polynomial-time “certificate” for every input such that there exists a polynomial-time algorithm that can solve the problem correctly given both the input and the certificate

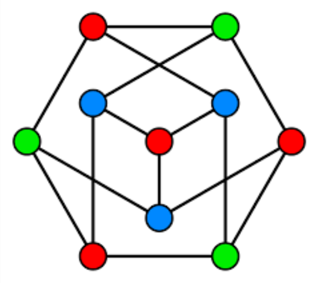
Examples of New Problems in NP

Graph 3-Coloring



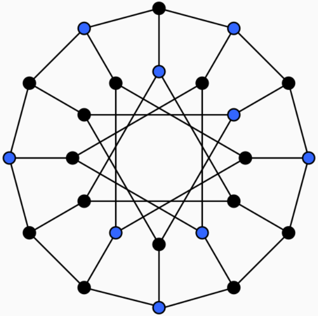
- **Graph Coloring:** Given a graph G , is it possible to color the vertices of G using only three colors, such that no edge has both end points colored with the same color
- Graph coloring is in NP:
  - Given a solution, we can check that only 3 colors are used in $O(n)$ time, and verify that each edge has differently-colored endpoints in $O(m)$ time

Graph 3-Coloring



- **Graph Coloring:** Given a graph G , is it possible to color the vertices of G using only three colors, such that no edge has both end points colored with the same color
- What problem does this remind you of?
- Answer: coloring the vertices with 2 colors is the same as the graph being *bipartite*. Remember: can solve that in $O(n+m)$ time using BFS! Does not work for 3-coloring however

Independent Set



- For a graph G and an integer k, is there a set $S \subseteq V$ of k vertices such that no two are adjacent? (In other words, for any $(u, v) \in E$, either $u \notin S$ or $v \notin S$.)
- **In pairs:** why is this problem in NP?

Testing your Intuition

- Not all problems can be easily verified (Not all problems are in NP)
- Classic example: I give you some code I wrote and I asked you “does this program loop infinitely”?
- You can give an input where it seems to keep looping. But I *can’t* verify it in polynomial time.

This is not a “probably” statement; it is mathmatically impossible to verify. You’ll explore this in CSCI 361.

Quick Question

- If a problem is in P, does that mean it is also in NP?
- In P: can solve in polynomial time
- In NP: can verify in polynomial time
- Answer: yes! If a problem can be solved in polynomial time, we can verify it in polynomial time
  - Intuitively: to check the solution, we just solve the problem and double-check that the solution matches
  - More formally: can just use an empty certificate

P vs NP

P vs NP

- We know that every problem in P is also in NP
- Is the *reverse* true? If a problem can be verified efficiently, does that mean it can be efficiently solved in the first place?
- Or: do there exist problems that can be verified quickly, but are *impossible* to solve quickly?
- This is what it means to ask if $P = NP$

The answer to $P = NP$ has extensive real-world implications, both good and bad, either way.

What happens if $P = NP$

Some good things:

- We can solve most real-world problems quickly
- Can lay out chips optimally, pack trucks optimally, schedule shipping optimally, with minimal computational cost

Some bad things:

- (Public key) cryptography does not exist

What happens if $P \neq NP$

Some good things:

- Can encrypt messages; hide information
- No longer need to look for polynomial-time solutions to some problems

Some bad things:

- Some problems we cannot solve efficiently without massive computational cost

Million Dollar Question

BUSINESS INSIDER

BUSINESS INSIDER

If you can solve this math problem you'll get a $1 million prize — and change internet security as we know it

By Andy Kiersz

P vs NP

- Possibly the second biggest open problem in computer science
- One of the biggest open problems in math as well
- We are not even close to solving it!

Proving that Problems are Hard to Solve

Goal

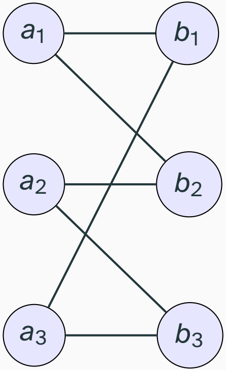
- We want to show that some problems are probably not in P
- (We won't know for sure: after all it's possible that $P = NP$)
- We can show that a problem is efficient to solve using an *algorithm*
- Plan: we will show that a problem is probably not efficient to solve using a reduction
- Let's review reductions

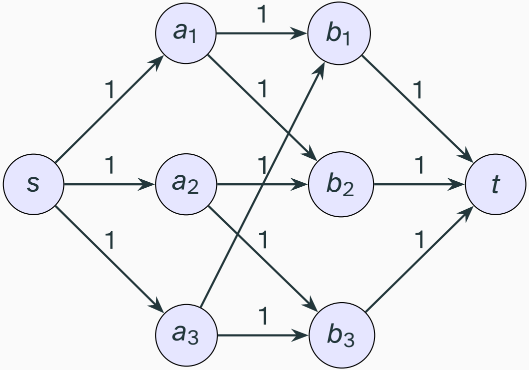
Recall: median finding

- With this algorithm I can make the following claim:
- "If I can sort in $O(f(n))$ time, then I can find the median in $O(f(n)+1)$ time"

```
Sort A
if n is even:
    return A[n/2] + A[n/2+1]
else:
    return A[(n+1)/2]
```

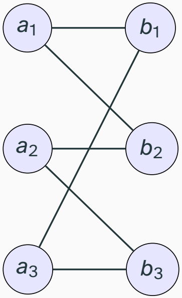
Recall: Bipartite Matching

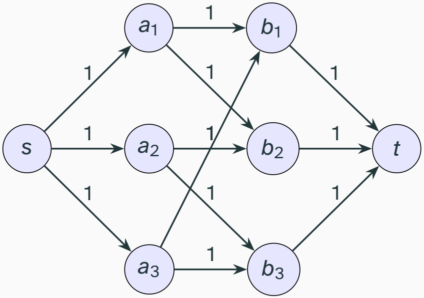




- Recall: given a bipartite graph G, find the largest subset of edges M such that no two edges in M share an endpoint

Recall: Bipartite Matching






```
Create a flow network G′ as follows:
    add vertices s, t to G′
    add edges from s to all vertices in A
    add edges from all vertices in B to t
    all edges directed to the right
    all edges have capacity 1
Value of best flow in G′ is size of best matching in G
```

Recall: Bipartite Matching

- Recall: given a bipartite graph G, find the largest subset of edges M such that no two edges in M share an endpoint
- With the algorithm below I can make the following claim:
- “If I can solve network flow in $O(f(m))$ time, then I can solve bipartite matching in $O(m + f(m))$ time”

```
Create a flow network G′ as follows:
    add vertices s, t to G′
    add edges from s to all vertices in A
    add edges from all vertices in B to t
    all edges directed to the right
    all edges have capacity 1
Value of best flow in G′ is size of best matching in G
```

Recall: Bipartite Matching

- We have: "If I can solve network flow in $O(f(m))$ time, then I can solve bipartite matching in $O(m + f(m))$ time"

- How can I rephrase this as a *lower bound*?

- If it is *impossible* to solve bipartite matching in $f(m)$ time, for some $f(m) > m$, then it is *impossible* to solve network flow in $O(f(m))$ time
  - If I could, it would contradict our statement above!

Lower Bounds via Reductions

- Reductions: create an algorithm for a problem using a different problem
- Strategy: let's say we can solve problem X using an algorithm for problem Y. Then it's impossible for Y to be *faster* to solve than X
- Conclusion: Y takes at least as long to solve as X.

NP-hard Problems

Plan for Lower Bounds

- We will define a set of problems that are “NP-hard”
- Idea: NP-hard problems are (probably) not in P: are probably not possible to solve in polynomial time
- Plan: we will use reductions! If we can use X to solve Y, and problem Y is not in P, then problem X is also not in P

Starting Point

- Reductions only show that one problem is *as hard* as another
- For this to work: we need to start with a problem that is (probably) hard to solve efficiently

Satisfiability

Satisfiability

$$\phi = \overbrace{(\overline{x}_1 \vee x_2 \vee x_3)}^{\textit{clause}} \wedge (x_1 \vee \overline{x}_2 \vee x_4) \wedge (\overline{x}_1 \vee \overline{x}_3 \vee x_4) \wedge (x_2 \vee x_3 \vee \overline{x}_4)$$

- The classic problem in NP
- Many variations of this problem; we'll look at one called 3-SAT
- **3-SAT**: given a formula ϕ, where ϕ consists of:
    - m "clauses;" each clause is the "or" of exactly 3 literals
    - Each clause has an "and" between it (so *every* clause must evaluate to true)

3-SAT rephrased

$$\phi = (\overline{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x}_2 \vee x_4) \wedge (\overline{x}_1 \vee \overline{x}_3 \vee x_4) \wedge (x_2 \vee x_3 \vee \overline{x}_4)$$

- *n* variables, *m* clauses
- Each clause has 3 literals (a variable, or the "not" of the variable)
- Clause is true if *at least* one literal in the clause is true
- Every clause must evaluate to true

3-SAT rephrased

$$\phi = (\overline{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x}_2 \vee x_4) \wedge (\overline{x}_1 \vee \overline{x}_3 \vee x_4) \wedge (x_2 \vee x_3 \vee \overline{x}_4)$$

- One solution:
  - x_1 = false
  - x_2 = false
  - x_3 = true
  - x_4 = true

NP-Hard Problems

Cook-Levin Theorem

- If 3-SAT can be solved in polynomial time, then *any problem in NP* can be solved in polynomial time

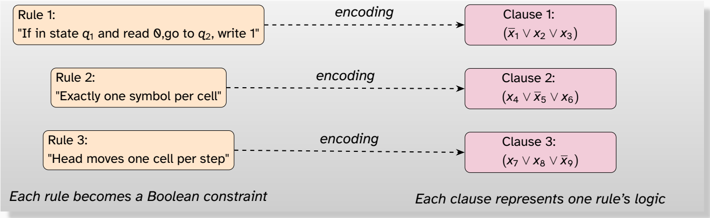
- In other words:

  - If 3-SAT can be solved in polynomial time, then $P = NP$

  - If 3-SAT cannot be solved in polynomial time, then $P \neq NP$

- How could one possibly prove such a general statement???

Cook-Levin Theorem: Very Brief Intuition



(You do not need to know this for the final.)

- Any computer program can be written down as a 3-SAT formula
- Long story short: given any instance of a problem in NP, we can create a 3-SAT formula that is satisfiable if and only if there is a solution to the original problem
- In other words: it's an explicit *reduction* from any problem in NP to 3-sat

NP-hard definition

A problem is NP-hard if:

- For any problem $Y \in NP$, we can reduce Y to X in polynomial time
- *Therefore:* if X can be solved in polynomial time, then *any problem in NP* can be solved in polynomial time
- In other words: X can be solved in polynomial time if and only if $P = NP$
- Cook-Levin theorem: 3-SAT is NP-hard

What Does This Mean

- When I said "probably" before, what I really meant was "if $P \neq NP$"
- So I would have said: it is "probably" not possible to solve 3-SAT in polynomial time
- From now on we'll be more formal: we will say "3-SAT is NP-hard"
- If you want to intuitively think of "NP-hard" as meaning "cannot be solved in polynomial time" I think that's fine.
- But: please know the real definition, and bear in mind that the true meaning is more subtle than that

NP-Complete

- A problem is **NP-Complete** if it is both NP-hard, and in NP
- So: 3SAT is NP-Complete
- We’ll see next week: Knapsack is NP-Complete
- NP-Complete problems are the hardest problems in NP: if any of them can be solved in polynomial time, then all problems in NP can be solved in polynomial time

Proving that Problems are NP-Hard

Proving a Problem is NP-Hard

We use the notation $X \leq_P Y$ to denote that we can reduce X to Y in polynomial time.

- In other words: given an instance a of X, in polynomial time we can define an instance a' of Y such that the answer to a is "yes" if and only if the answer to a' is "yes"

We will use the following fact:

- If $X \leq_P Y$, and X is NP-hard, then Y is NP-hard
- Let's explain why intuitively on the board

More Formal Proof

Theorem

If $X \leq_P Y$, and X is NP-hard, then Y is NP-hard

Proof Summary. Since X is NP-hard, for any problem Z in NP, $Z \leq_P X$: we can reduce Z to X in polynomial time.

We can also reduce X to Y in polynomial time.

By applying both reductions one after the other, we reduce Z to Y in polynomial time. Therefore, Y is NP-hard.

Plan from Here on Out

Theorem

If $X \leq_P Y$, and X is NP-hard, then Y is NP-hard

- We will be showing that a bunch of problems are NP-hard
- Plan: if we can reduce 3-SAT to a problem X, then X is NP-hard. Then if we can reduce X to Y, we must have that Y is NP-hard and so on
- First we will reduce problems to each other; we'll reduce 3-SAT to one of them later
- We'll eventually prove all of them are NP-hard
- But the 3-SAT reduction is a more difficult reduction, so I want to get some reduction practice first