Complexity Classes

Are some problems more difficult than others?

Slides by: Sander Aarts

Last week we solved IPs - why continue?

Problem $\quad$ Integer Program

$$\Pi \quad \Rightarrow \quad \Pi_{\mathbf{IP}}$$

Last week we solved IPs - why continue?

Problem $\quad\quad$ Integer Program

$$\Pi \Rightarrow \Pi_{\mathbf{IP}} \Rightarrow \boxed{\text{IP Solver}}$$

Last week we solved IPs - why continue?

Problem Π $\Rightarrow$ Integer Program $\Pi_{\mathbf{IP}}$ $\Rightarrow$ $\boxed{\text{IP Solver}}$ $\Rightarrow$ Solution $\mathrm{sol}(\Pi)$

Last week we solved IPs - why continue?

Problem Π $\Rightarrow$ Integer Program $\Pi_{\mathbf{IP}}$ $\Rightarrow$ $\boxed{\text{IP Solver}}$ $\Rightarrow$ Solution $\mathrm{sol}(\Pi)$

Can all problems be solved with this pipeline?

Is this efficient?

Last week we solved IPs - why continue?

Problem		Integer Program				Solution
Π	$\rightarrow$	$\Pi_{\mathbf{IP}}$	$\Rightarrow$	IP Solver	$\Rightarrow$	$\mathrm{sol}(\Pi)$

Can all problems be solved with this pipeline?

Is this efficient?

Last week we solved IPs - why continue?

Problem		Integer Program				Solution
Π	$\Rightarrow$	$\Pi_{\mathbf{IP}}$	$\Rightarrow$	IP Solver	$\Rightarrow$	$\mathrm{sol}(\Pi)$

Can all problems be solved with this pipeline?

Is this efficient?

Outline: Complexity Classes

- Introduction: Why care about complexity?
- Decision Problems
  - Definition
  - Example: Minimum Cost Set Cover
- Difficulty Classes
  - P
  - NP
  - Co-NP

Learning objectives:
You are able to
- describe a decision problem
- describe a polynomial time checkable certificate
- state the definitions of three complexity classes

Introduction: Why care about complexity?

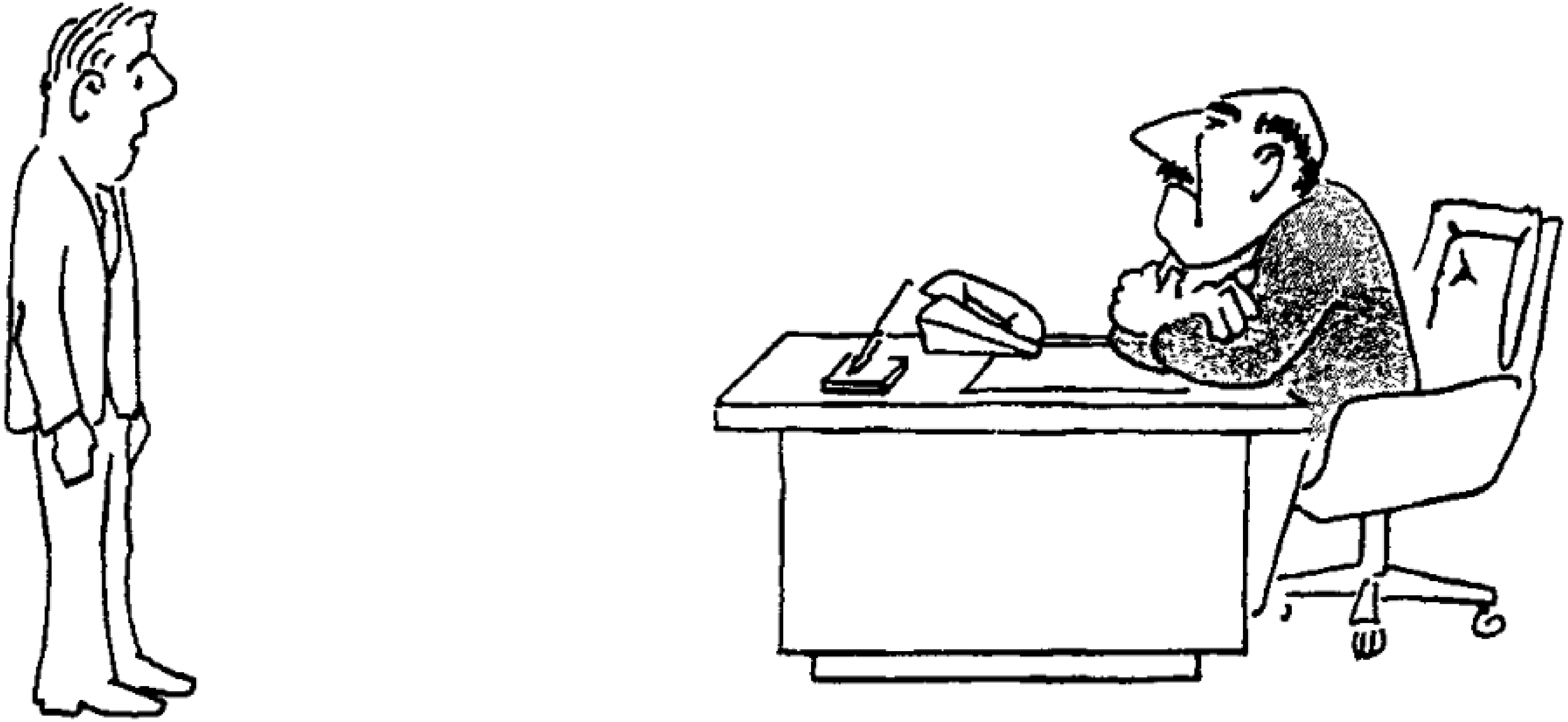
From Garey and Johnson's book *Computers and Intractability* (1979):

Your boss hands you a problem and tells you to find an **efficent** algorithm.

The problem is **tricky**. You work days and nights.

A week later, you have not found a good algorithm. You tell your boss...

Introduction: Why care about hardness?



"I can't find an efficient algorithm, I guess I'm just too dumb."

You can tell from the cast of characters it is 1979...

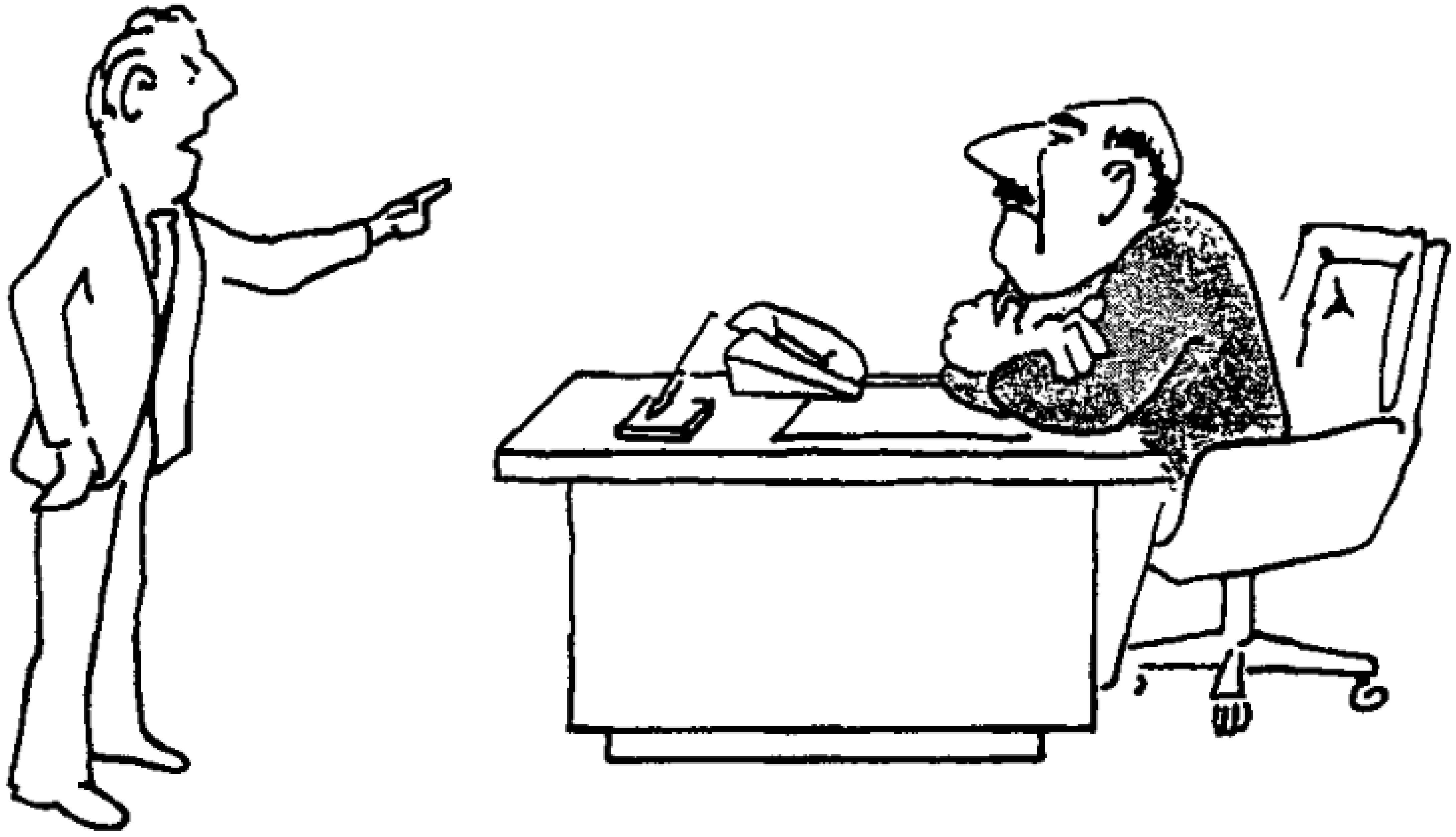
Source: *Garey, Michael R.; Johnson, D. S. (1979), Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman*.

Introduction: Why care about hardness?

That feels bad.

What if you could **prove** that finding an efficient algorithm to the problem is impossible?

Introduction: Why care about hardness?



“I can’t find an efficient algorithm, because no such algorithm is possible!”

Source: Garey, Michael R.; Johnson, D. S. *(1979), Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman*.

Introduction: Why care about hardness?

That feels
much better!

Introduction: Why care about hardness?

That feels
much better!

However, **nobody knows how** to prove this.

But we can prove something like this...

Introduction: Why care about hardness?

“I can't find an efficient algorithm, but neither can all these famous people.”

Source: Garey, Michael R.; Johnson, D. S. *(1979), Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman*.

Introduction: Why care about hardness?

Almost as good.

This type of proofs relate to a problem's **complexity class.**

Introduction: Why care about hardness?

Almost as good.

”The set of problems not solvable by famous people”.

This type of proofs relate to a problem's **complexity class.**

Introduction: Why care about hardness?

Almost as good.

This type of proofs relate to a problem's **complexity class.**

Today's goal: Learn about complexity classes and understand their algorithmic implications.

Why? Sets standards for what is reasonable to expect.

Outline

- Introduction: Why care about Hardness?

- Decision Problems
  - Definition
  - Example: Set Cover
- Difficulty Classes
  - P
  - NP
  - Co-NP

Decision Problems: example

A **decision problem** is a problem $\prod$ with binary output.

I.e. the solution is either **True** or **False**.

Decision Problems: example

A **decision problem** is a problem $\prod$ with binary output.

I.e. the solution is either **True** or **False**.

Example (Decision Set Cover):
Input: n elements and m subsets, integer $k > 0$.
Goal: *Decide* if a set cover of size $\leq k$ exists

Decision Problems: details matter!

Small differences in the problem can mean large difference in difficulty!

Decision Problems: Many forms

Small differences in the problem can mean large difference in difficulty!

Example (Decision Set Cover):
Input: n elements and m subsets, integer $k > 0$.
Goal: Decide if a set cover of size $\leq k$ exists

Not trivial....

Decision Problems: Many forms

Small differences in the problem can mean large difference in difficulty!

Example (Decision Set Cover):
Input: n elements and m subsets, integer $k > 0$.
Goal: Decide if a set cover of size $\leq k$ exists

Not trivial....

Example (Easy Set cover):
Input: n elements and m subsets
Goal: Decide if a set cover exists

Piece of cake!

Outline

- Introduction: why care about Hardness?
- Decision Problems: formalizing a problem
  - Definition
  - Example: Set Cover
- Difficulty Classes
  - P
  - NP
  - Co-NP
  - Relationships

Complexity Classes: P

There are many complexity classes. This lecture covers the most common 3.

P (**P**olynomial Time)
This is the class of problems that can be solved in polynomial time.

Complexity Classes: P

There are many complexity classes. We cover the most common 3.

P (**P**olynomial Time)
This is the class of problems that can be solved in polynomial time.

For problems in P your boss can expect you to code an efficient algoritm.

Complexity Classes: P

There are many complexity classes. We cover the most common 3.

P (**P**olynomial Time)
This is the class of problems that can be solved in polynomial time.

For problems in P your boss can expect you to code an efficient algoritm.

Computation will eventually finish

Complexity Classes: P

There are many complexity classes. We cover the most common 3.

P (**P**olynomial Time)
This is the class of problems that can be solved in polynomial time.

Examples (Problems in P):
- Stable Matching
- Minimum Spanning Tree (MST)
- **2-Approximation** to Minimum Vertex Cover

Proving a problem is in P

How do we prove a problem is in P?

Show a polynomial time algorithm!

Proving a problem is in P

How do we prove a problem is in P?

Show a polynomial time algorithm!

Examples (we designed the algorithms):
- Stable Matching
- Minimum Spanning Tree (MST)
- **2-Approximation** to Minimum Vertex Cover

Proving a problem is in P

How do we prove a problem is in P?

Show a polynomial time algorithm!

Examples (we designed the algorithms):
- Stable Matching
- Minimum Spanning Tree (MST)
- ~~**2-Approximation**~~ to Minimum Vertex Cover

Proving a problem is in P

How do we prove a problem is in P?

Show a polynomial time algorithm!

Examples (we designed the algorithms):
- Stable Matching
- Minimum Spanning Tree (MST)
- ~~**2-Approximation**~~ to Minimum Vertex Cover

Is there a vertex cover of size k?

Outline

- Introduction: why care about Hardness?
- Decision Problems: formalizing a problem
  - Definition
  - Example: Set Cover
- Difficulty Classes
  - P
  - NP
  - Co-NP
  - Relationships

Complexity Classes: NP

NP = (**N**on-deterministic **P**olynomial Time)

Call instance I of problem a Yes-instance if the solution to the decision problem is True.

A problem is in NP if for every Yes-instance there is a *certificate* that can be verified in polynomial time.

Complexity Classes: NP

NP = (**N**on-deterministic **P**olynomial Time)

Call instance I of problem Π a Yes-instance
if the solution to the decision problem is True.

A problem is in NP if for every Yes-instance
there is a *certificate* that can be verified in
polynomial time.

Complexity Classes: NP

NP = (**N**on-deterministic **P**olynomial Time)

Call instance I of problem Π a Yes-instance
if the solution to the decision problem is True.

A problem Π is in NP if for every Yes-instance
there is a *certificate* that can be verified in
polynomial time.

Proving a problem is in NP

A problem Π is in NP if for every Yes-instance there is a *certificate* that can be verified in polynomial time.

To prove a problem is in NP we need the following:

1. A proof that every Yes-instance has a Yes-certificate

2. An algorithm that correctly verifies a given certificate in polynomial time.

Proving a problem is in NP - Example

Example (Decision Set Cover):
Input: n elements and m subsets, integer $k > 0$.
Goal: Decide if a set cover of size $\leq k$ exists

Claim: (Decision) Set Cover is in NP.

Proof. Every Yes-instance has a set cover with no more than k subsets.

Certificate: w.l.o.g let $\{S_1, S_2, \ldots, S_q\}$ be the set cover, where $q \leq k \leq m$.

To show: Can verify that all n elements are covered in polynomial time.

Go over each S_i in turn and mark every element as covered. There are at most m sets, each containing at most n elements, so this takes $O(nm)$ time.

Complexity Classes: NP - an example

Example (Decision Set Cover):
Input: n elements and m subsets, integer $k > 0$.
Goal: Decide if a set cover of size $\leq k$ exists

This exists because we assume a Yes-instance

Claim: (Decision) Set Cover is in NP.

Proof. Every Yes-instance has a set cover with no more than k subsets.

Certificate: w.l.o.g let $\{S_1, S_2, \dots, S_q\}$ be the set cover, where $q \leq k \leq m$.

To show: Can verify that all n elements are covered in polynomial time.

Go over each S_i in turn and mark every element as covered. There are at most m sets, each containing at most n elements, so this takes $O(nm)$ time.

Complexity Classes: NP - an example

Example (Decision Set Cover):
Input: n elements and m subsets, integer $k > 0$.
Goal: Decide if a set cover of size $\leq k$ exists

Claim: (Decision) Set Cover is in NP.

Proof. Every Yes-instance has a set cover with no more than k subsets.

Certificate: w.l.o.g let $\{S_1, S_2, \dots, S_q\}$ be the set cover, where $q \leq k \leq m$.

To show: Can verify that all n elements are covered in polynomial time.

Go over each S_i in turn and mark every element as covered. There are at most m sets, each containing at most n elements, so this takes $O(nm)$ time.

Given it exists, we can assume the certificate is known.

Outline

- Introduction: why care about Hardness?

- Decision Problems: formalizing a problem
  - Definition
  - Example: Set Cover

- Difficulty Classes
  - P
  - NP
  - Co-NP
  - Relationships

Complexity Classes: co-NP

Co-NP = "complement"-NP

Call instance I of problem Π a No-instance
if its solution to the decision problem is False.

A problem Π is in co-NP if for every No-instance there is a *certificate* that can be verified in polynomial time.

Proving a problem is in co-NP

The proof structure is the same as for NP

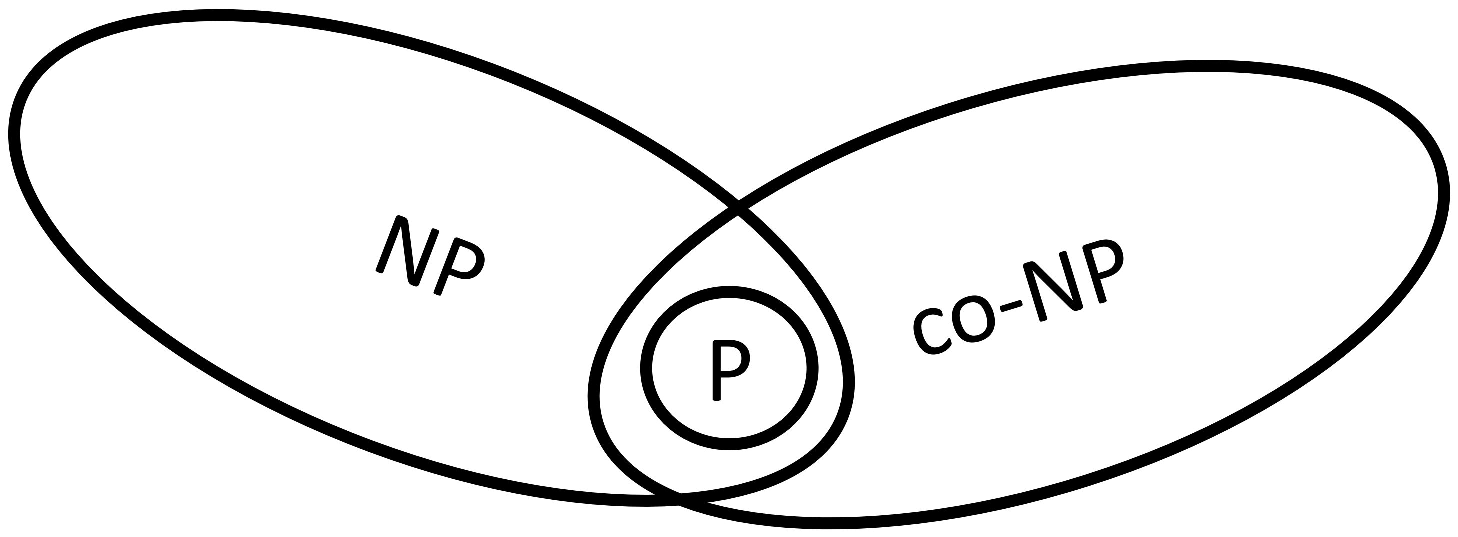
Proving a problem is in co-NP

The proof structure is the same as for NP

Example problem:
Is a number x a prime?

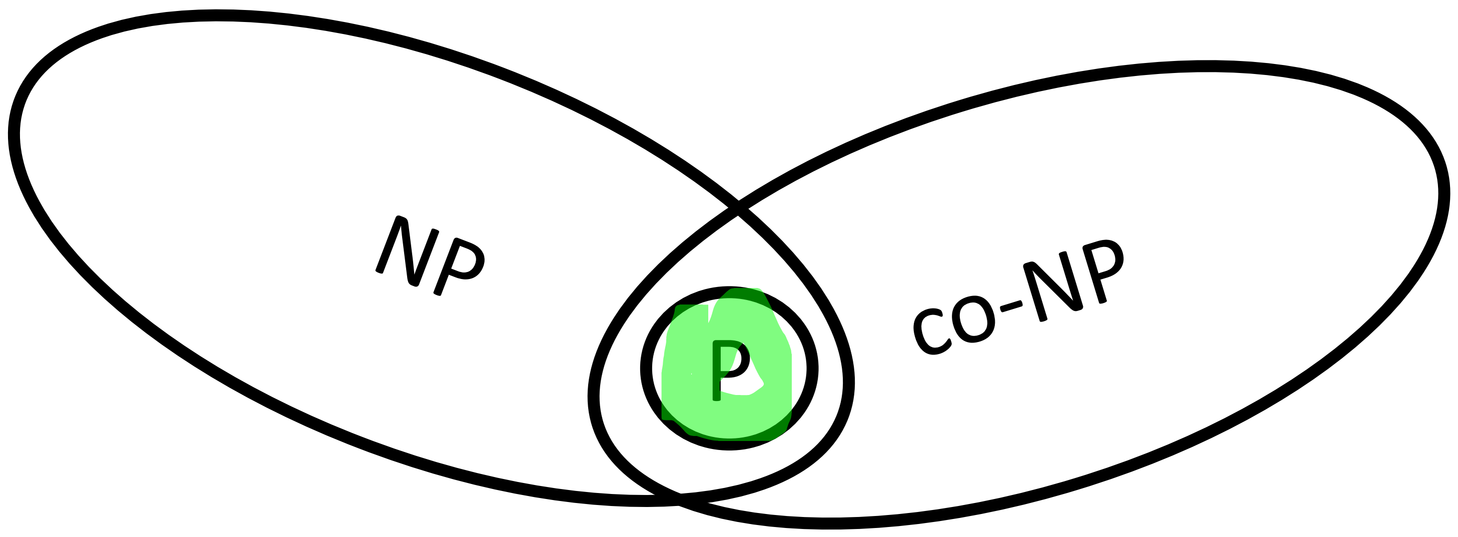
In a No-instance, provide the factors.

Complexity Classes: Relationships



This is how most people *think* the classes relate.

Complexity Classes: Relationships

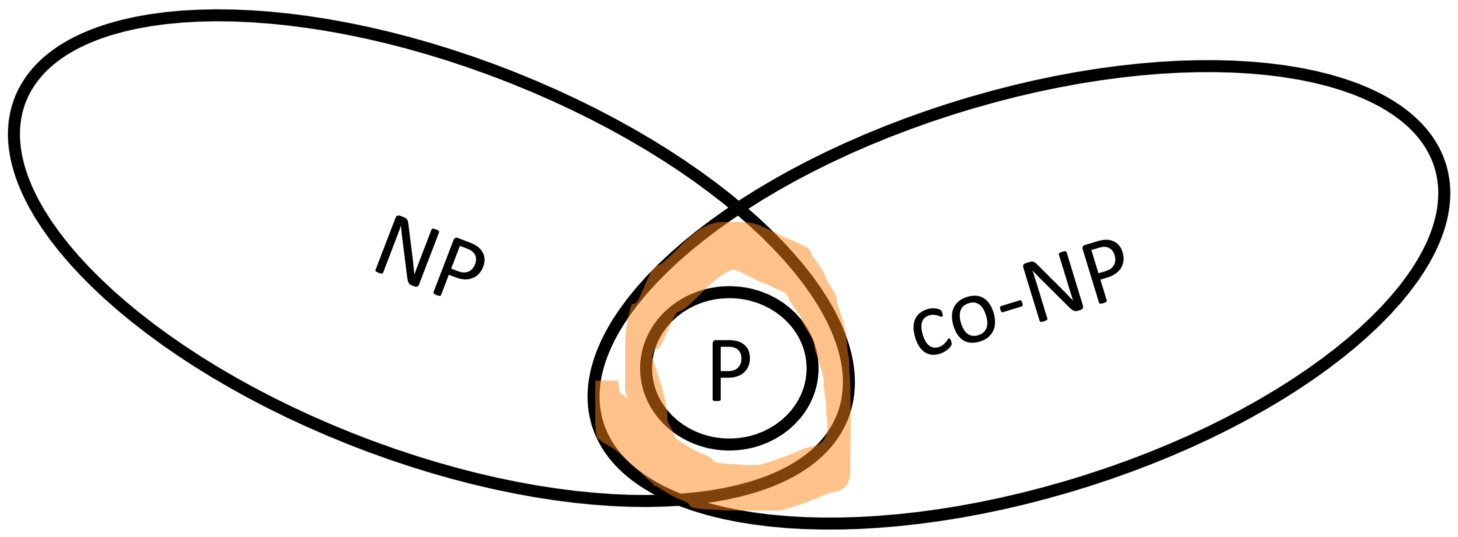


This is how most people *think* the classes relate.

Proven relationships:

P is in NP and in co-NP

Complexity Classes: Relationships



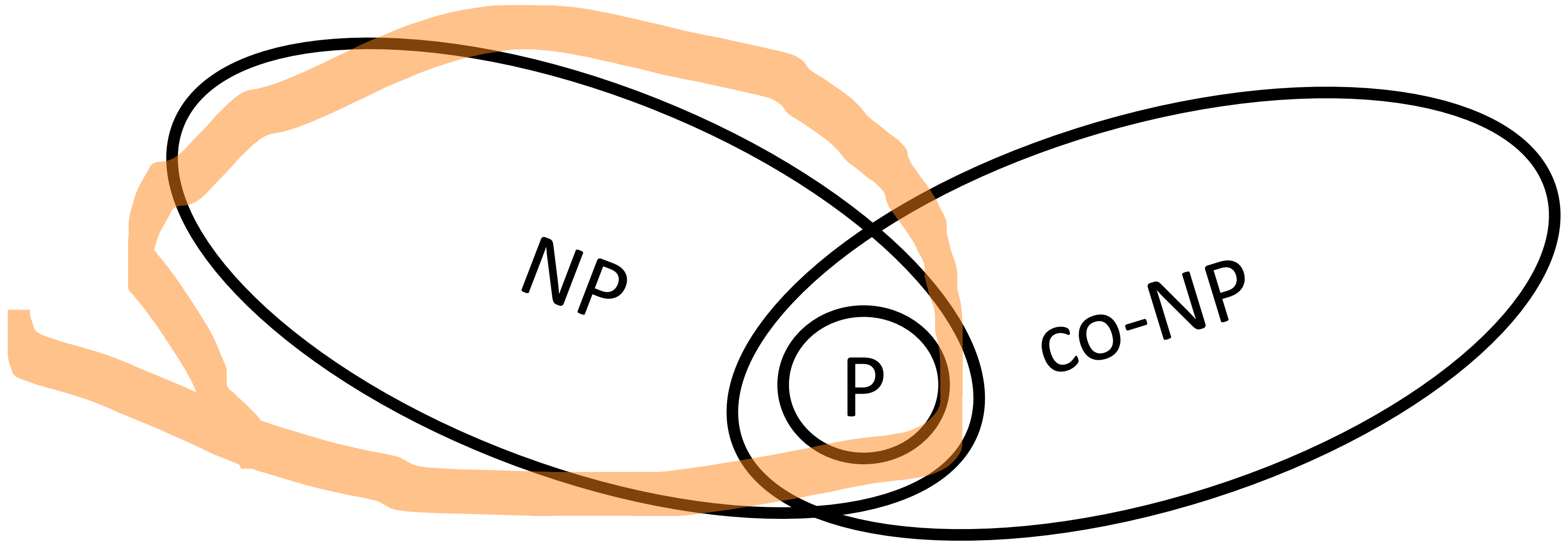
This is how most people *think* the classes relate.

There are problems that are both in co-NP and NP but not in P

Proven relationships:

P is in NP and in co-NP

Complexity Classes: Relationships



This is how most people *think* the classes relate.

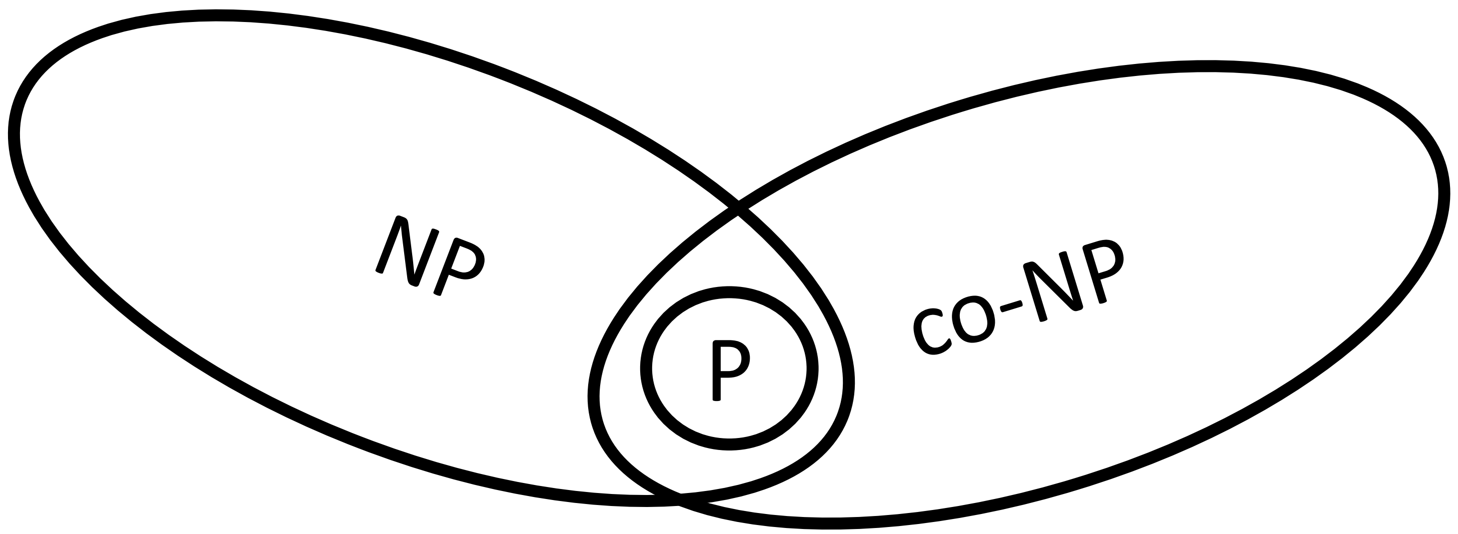
There are problems that are both in co-NP and NP but not in P

Open problems:

P $=_?$ NP ($1 million prize)

NP $=_?$ co-NP

Complexity Classes: Relationships



This is how most people *think* the classes relate.

What most think:

P ≠ NP
NP ≠ co-NP

Outline

- Introduction: why care about Hardness?

- Decision Problems: formalizing a problem
  - Definition
  - Example: Set Cover

- Difficulty Classes
  - P
  - NP
  - Co-NP
  - Relationships

Next part: Reductions

Proving one problem is harder than another.