# Introduction to Computability Complexity

## Introduction to the Course

Ugo Dal Lago



University of Bologna, Academic Year 2024/2025

Section 1

# The Logistics

# Organisation

- **Webpage**
  - At VIRTUALE: `http://virtuale.unibo.it/`
- **Email**
  - `ugo.dallago@unibo.it`
- **Office Hours**
  - *Where*: see `http://udallago.github.io`
  - *When*: there is no fixed office hours, just write me an email and we will fix an appointment.
- **Teaching Assistant**
  - Victor Arrial, `victor.arrial2@unibo.it`
  - Please contact both of us in case you need help, this way your request will be taken into account ASAP.

# Structure and Schedule

- The *schedule* of this course's lectures will vary along the coming weeks.
  - All lectures will be on Monday (2pm-4pm) or on Thursday (9am-11am).
  - We will skip some of the lectures.
  - Please keep an eye to to the online schedule.
- **Content**: the course is divided into two parts:
  1. The very basics of *computability and complexity theory*, with implications for AI.
  2. Some rudiments of *computational learning theory*.
- **Requirements**:
  - We assume all students know what a *program* is.
  - We also assume familiarity with the basics of *probability theory*, *discrete mathematics* (natural numbers, graphs, and the like) and *mathematical logic* (propositional logic and its semantics).
- All lectures will be given by Ugo Dal Lago.
- Some of the exercise sessions will be given by Victor Arrial.

# Textbooks and Exams

- **Textbooks**

  The course will be as self contained as possible, but there are plenty of nice books from which you can learn more than what we will do, if you are interested.

  - Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press. 2007.
  - Christos Papadimitriou. *Computational Complexity*. Addison-Wesley. 1994.
  - Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: from Theory to Algorithms* Cambridge University Press. 2014.
  - Michael Kearns and Umesh Vazirani. *An Introduction to Computational Learning Theory* The MIT Press. 1994.

- **Exams**

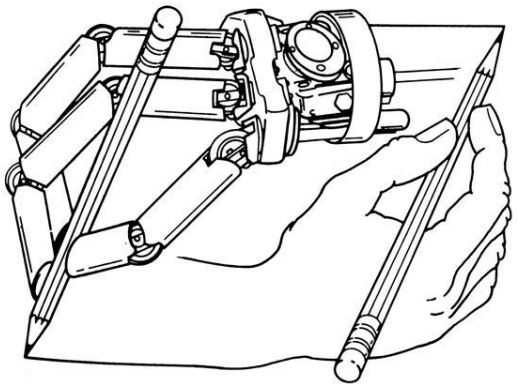
  - You need to pass a *written exam*.
  - As you know, you will get just *one* mark for the whole of this integrated course, and the three modules have the same weight.

# How to Fill any Gap?

TEXTS AND MONOGRAPHS IN COMPUTER SCIENCE
Edited by David Gries

**A BASIS FOR THEORETICAL COMPUTER SCIENCE**

**Michael A. Arbib**
**A.J. Kfoury**
**Robert N. Moll**



Springer-Verlag
New York Heidelberg Berlin

# Section 2

## What? Why?

# Is There any Limit on what AI can do?

- **AI and ML are Everywhere**.
  - The great impact to our society of technologies derived from Artificial Intellingence and Machine Learning is in front of us.
  - Learning the way tools from AI and ML can be employed in a variety of application scenarios (vision, social-media, recommendation systems, etc.) is of course important for you and your career.
  - A number of courses in the Master Degree you are enrolled in are specifically about the above.

# Is There any Limit on what AI can do?

- **AI and ML are Everywhere**.
  - The great impact to our society of technologies derived from Artificial Intellingence and Machine Learning is in front of us.
  - Learning the way tools from AI and ML can be employed in a variety of application scenarios (vision, social-media, recommendation systems, etc.) is of course important for you and your career.
  - A number of courses in the Master Degree you are enrolled in are specifically about the above.
- **This is *not* What this Module is About**!
  - The question we will try to ask you is rather the following: *is there any intrinsic limit to what we can do with AI and ML, or to the accurancy or efficiency of algorithms solving a give problem?*

# A Paradigm Shift

**"Algorithm $X$ solves Problem $Y$ Efficiently and with Great Accuracy"**

# A Paradigm Shift

**"Problem $Y$ Cannot be Solved by any Algorithm $X$".**

$$\Uparrow$$

**"Algorithm $X$ solves Problem $Y$ Efficiently and with Great Accuracy"**

# A Paradigm Shift

**"Problem $Y$ Cannot be Solved by any Algorithm $X$".**

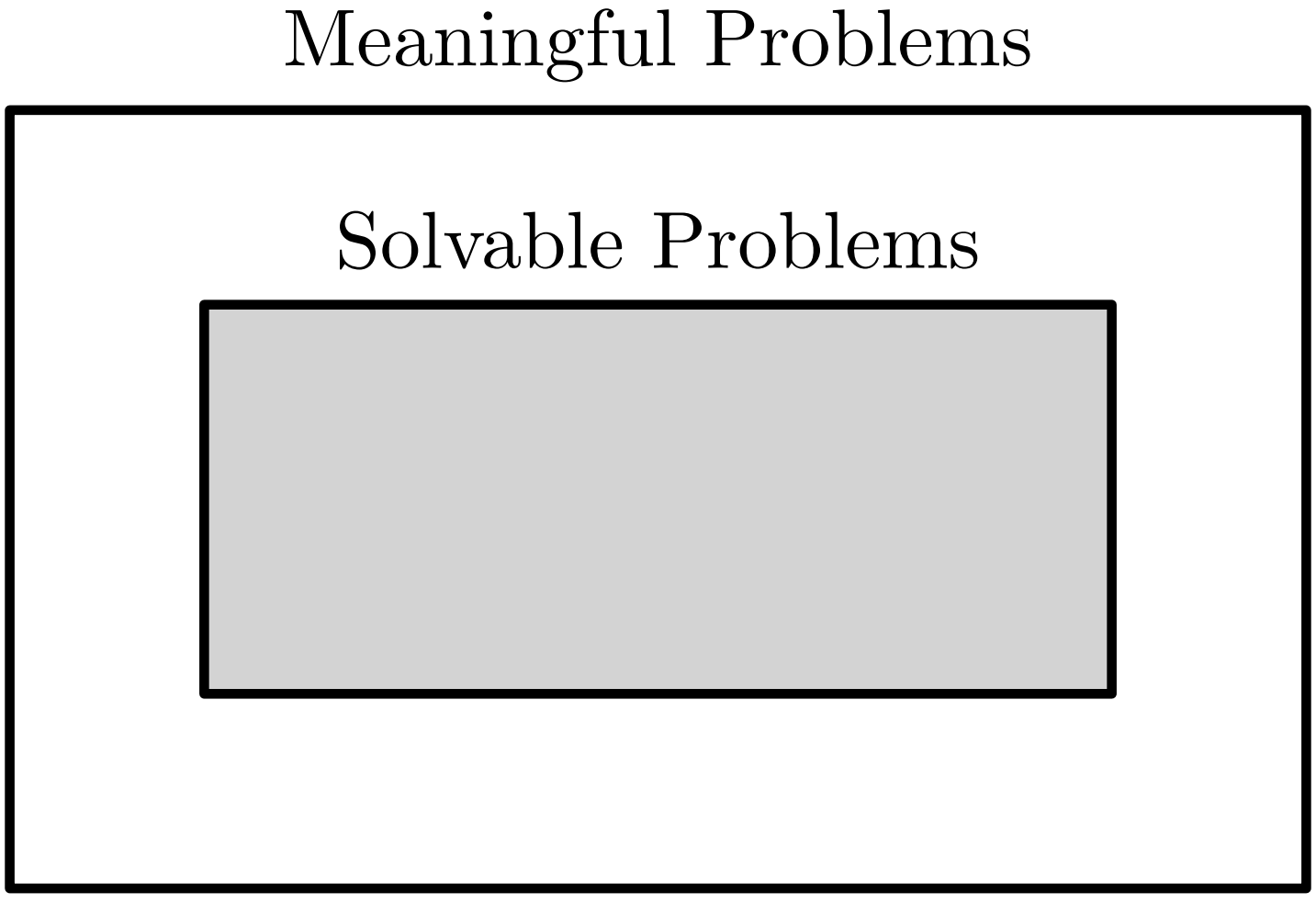
$\Uparrow$

**"Algorithm $X$ solves Problem $Y$ Efficiently and with Great Accuracy"**

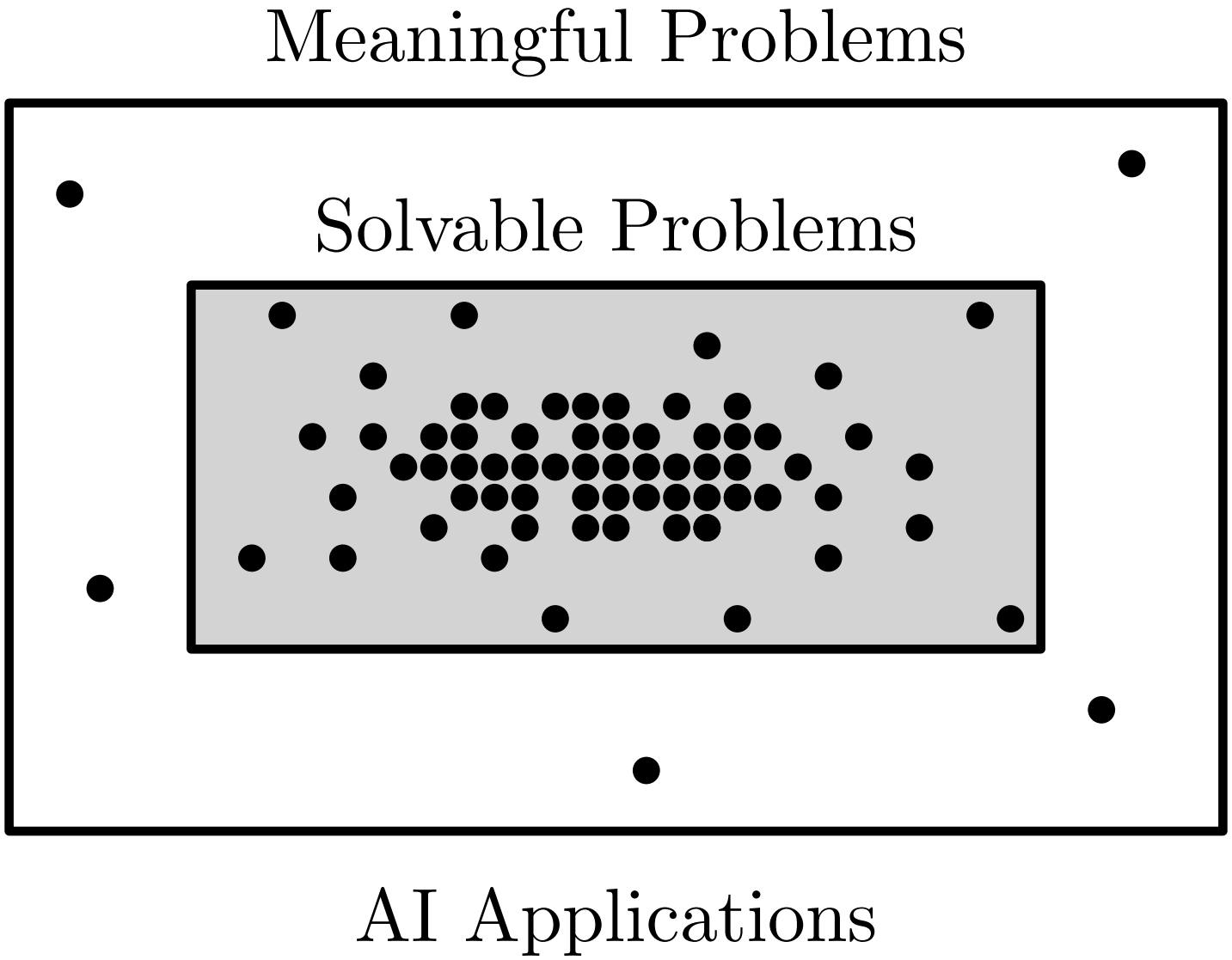
$\Downarrow$

**"Any Algorithm Solving $Y$ is bound to be Inefficient, or not Accurate".**

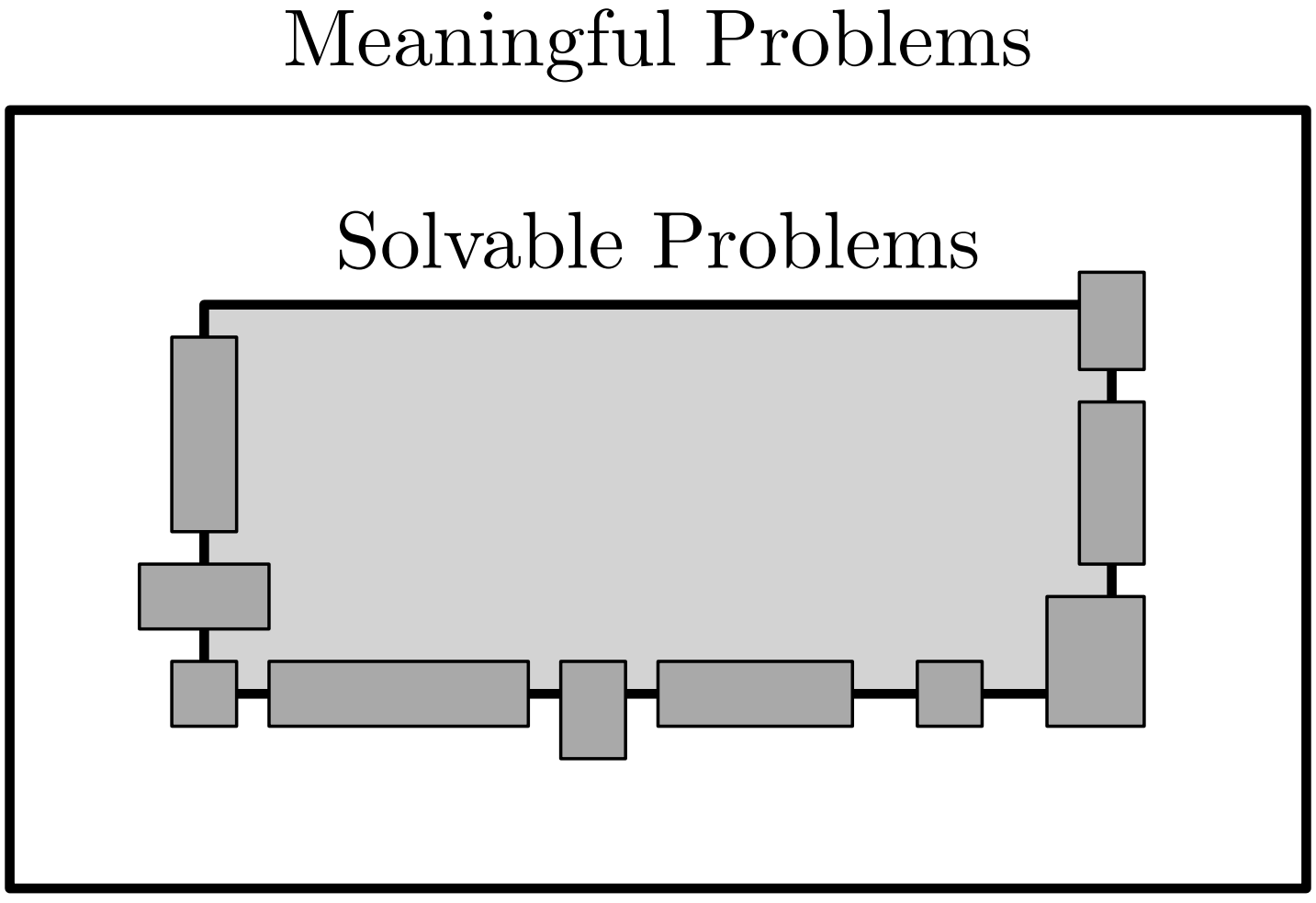
# A Paradigm Shift



# A Paradigm Shift



# A Paradigm Shift



# Why?

- Why shouldn't we just study a catalogue of algorithms, for each of them analyzing scope and performances?

# Why?

- Why shouldn't we just study a catalogue of algorithms, for each of them analyzing scope and performances?
- There are various reasons why we proceed this way.

# Why?

- Why shouldn't we just study a catalogue of algorithms, for each of them analyzing scope and performances?
- There are various reasons why we proceed this way.
  1. There are **too many interesting algorithms**.
     - There algorithms dealing with, e.g., strings, graphs, trees, and solving problems of so many different kinds.
     - The time at our disposal is rather limited, and there is nothing like a basic toolset every student should know.

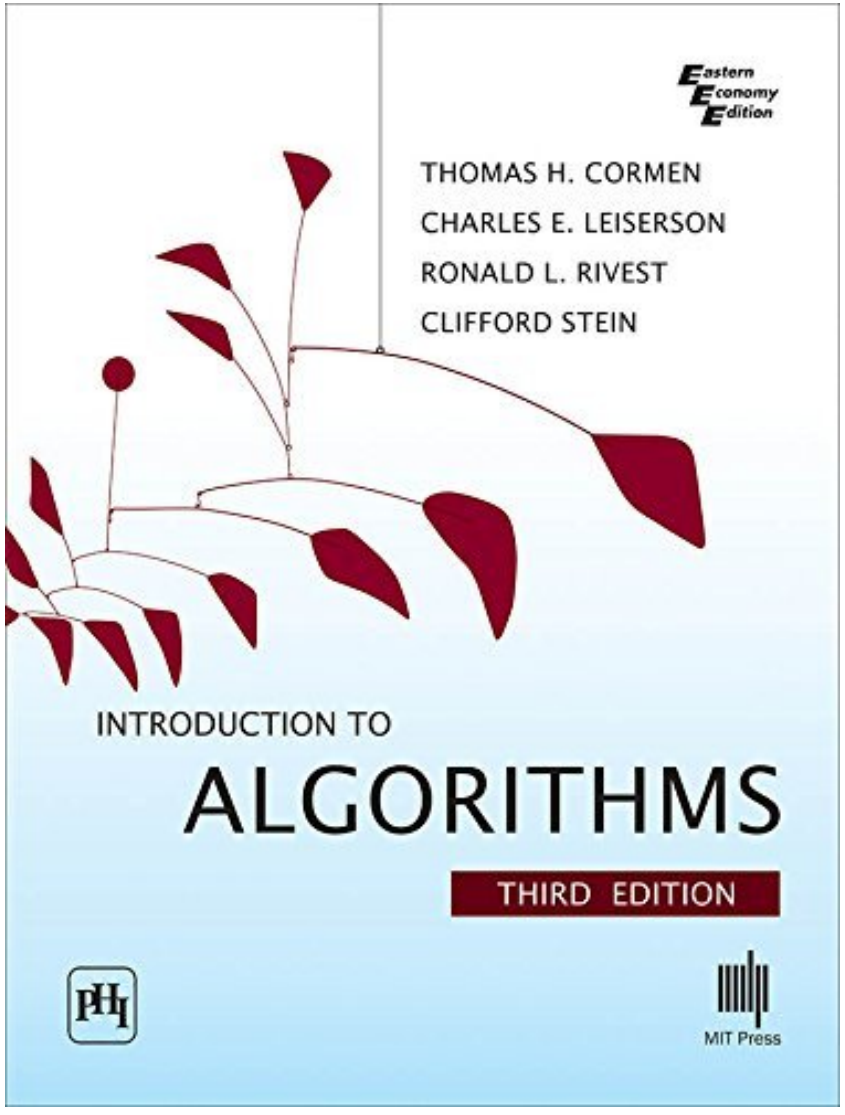
# Why?

- Why shouldn't we just study a catalogue of algorithms, for each of them analyzing scope and performances?
- There are various reasons why we proceed this way.
  1. There are **too many interesting algorithms**.
     - There algorithms dealing with, e.g., strings, graphs, trees, and solving problems of so many different kinds.
     - The time at our disposal is rather limited, and there is nothing like a basic toolset every student should know.
  2. In AI and ML, **there is just one algorithm**.
     - There is just *"one master algorithm"*.
     - Jokes aside, the diversity of algorithmic techniques seems to be more crucial in traditional CS than in AI.

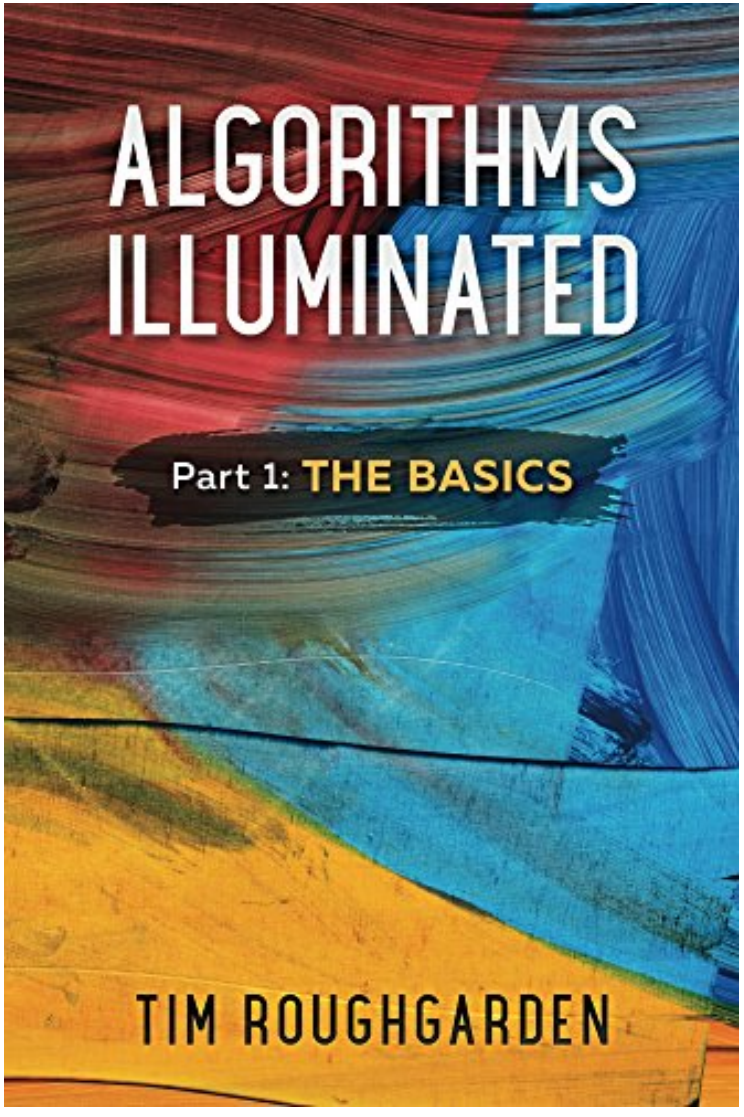
# Why?

- Why shouldn't we just study a catalogue of algorithms, for each of them analyzing scope and performances?
- There are various reasons why we proceed this way.
  1. There are **too many interesting algorithms**.
     - There algorithms dealing with, e.g., strings, graphs, trees, and solving problems of so many different kinds.
     - The time at our disposal is rather limited, and there is nothing like a basic toolset every student should know.
  2. In AI and ML, **there is just one algorithm**.
     - There is just *"one master algorithm"*.
     - Jokes aside, the diversity of algorithmic techniques seems to be more crucial in traditional CS than in AI.
  3. Learning new algorithms is **is easier** than learning why algorithms have their own limits.
     - Tons of interesting resources are available, and with a basic knowledge of programming and combinatorics, one can learn about specific algorithms.

# Some References on "Positive" Algorithmics



# Some References on "Positive" Algorithmics



# The Value of Negative Results

## A Quote from Thomas A. Edison

*"Negative results are just what I want. They're just as valuable to me as positive results. I can never find the thing that does the job best until I find the ones that don't."*

# The Value of Negative Results

## A Quote from Thomas A. Edison

*"Negative results are just what I want. They're just as valuable to me as positive results. I can never find the thing that does the job best until I find the ones that don't."*

- If we know that the task we want to accomplish is intrinsically *hard*, and we know *in what* its hardness resides, we can:
  1. Direct our work towards methodologies *specifically designed* for hard tasks.
  2. *Avoid* spending our time trying to design solutions that cannot exist.
  3. Inform the stakeholders *in advance* that the solution we are going to design will likely be inefficient.
  4. Decide that the task is *too difficult*, and that it cannot be accomplished the way we want.
  5. Decide that it is better to switch to a *relaxed* version of the task, which instead admits reasonable solutions.

# Example Problems - I

## Natural Number Multiplication

Suppose you want to write a `Python` program which multiplies two positive integer numbers, which can both be expressed as $n$-digits numbers, but which are represented as lists rather than scalars. Suppose that the operations at your disposal are just the basic arithmetic operations *on digits*, so that you cannot just multiply the two numbers. How should you write your program?

# Example Problems - II

## Maximizing the Number of Invitees

Suppose you are going to marry your boyfriend (or girlfriend) soon, and you want to decide the list of invitees. Since your soon-to-be father-in-law will pay all bill, you have all the interest in keeping the list of invitees as large as possible. But actually, it would be impossible to invite all the people in the set $L = \{a_1, \ldots, a_m\}$ of all candidate invitees, since some of them are kind of incompatible, and cannot be invited together, i.e. there is another set $I = \{(b_1, c_1), \ldots, (b_n, c_n)\} \subseteq L \times L$ whose elements are those candidate invitees pairs which are incompatible. Would it be possible to maximize the length of the list of invitees?

# Example Problems - III

## Learning Programs with Finite Inputs and Outputs

Suppose your ML teacher asks you to write a learning algorithm capable of reconstructing a program by querying it as a black box, without looking at the code. You can assume that the program takes in input a list of booleans of a fixed (rather big) length $n$, and produces in output one boolean value. How would you proceed? On which input would you query the program? Would it be possible to query the program on significantly less inputs than the $2^n$ possible ones?

# The Module's Structure

1. **Introduction to Computational Complexity**
   - Historical, Conceptual, and Mathematical Preliminaries
   - The Computational Model, and Uncomputability
   - The Class P
   - The Class NP and NP-Completeness
   - Optimization Problems and Their Hardness

# The Module's Structure

1. **Introduction to Computational Complexity**
   - Historical, Conceptual, and Mathematical Preliminaries
   - The Computational Model, and Uncomputability
   - The Class P
   - The Class NP and NP-Completeness
   - Optimization Problems and Their Hardness
2. **Some Rudiments of Computational Learning Theory**
   - From Computation to Learning
   - Positive and Negative Results about Learning Problems
   - The VC Dimension

# Thank You!

Questions?