# PRISM

## Lecture 3 - Scientific Problem Solving

Roger Grosse and Ishtiaque Ahmed

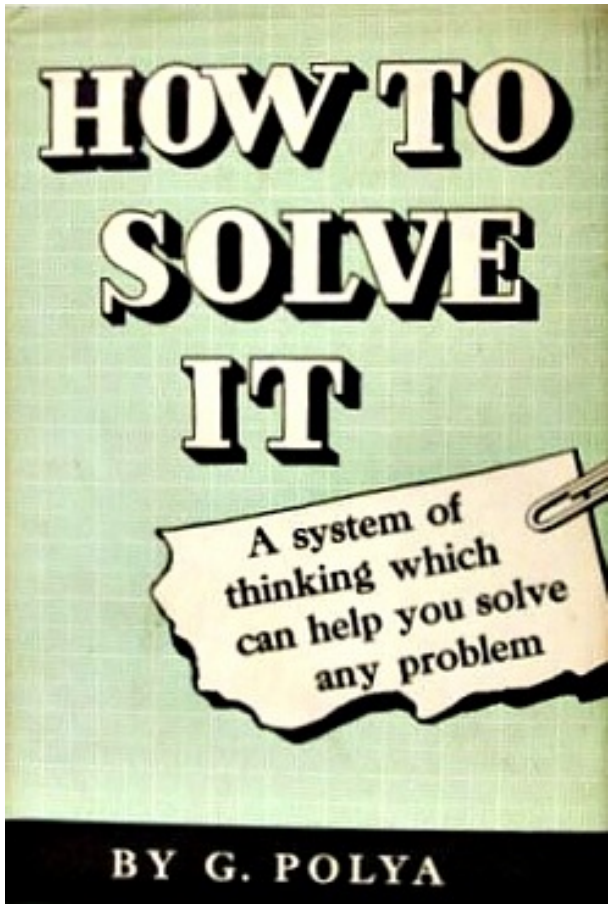
University of Toronto, Winter 2021

# Today

- Some research problems require genuine creativity, but this is unusual
- Most problems you'll work on can be solved methodically ("99% perspiration")
- **Today:** my attempt to list some problem solving strategies that are relevant to all areas of CS
- If I get hit by a bus tomorrow, my (senior) grad students should be able to finish their projects and keep publishing based on the advice here

# Generic Research Strategies

- George Polya's classic 1945 book, "How to Solve It"
- Generic problem solving heuristics (reason by analogy, decompose into subproblems, work backwards from the goal, etc.)
- Illustrated with high school mathematics, but the advice is very broadly applicable (even beyond math)
- An inspiration for a lot of early work in AI (such as the term "heuristic")



# Generic Research Strategies

What if you're stuck?

- Have you spent at least an hour brainstorming things to try?
  - This is usually enough time to come up with more ideas than you can possibly follow through on!
  - Your supervisor or grad student mentor can help you prioritize
  - Your supervisor will be very impressed if you come prepared with a list of ideas
- Have you read the literature on topics that are obviously related?
  - What tricks and other advice have they come up with?
- Have you found an experimental setup where you can iterate quickly?
  - **Machine learning:** small datasets where you can train a model in 15 minutes
  - **Math:** special cases

# Generic Research Strategies

Consider related problems:

- Is there an easier variant of the problem that still suits your needs?
  - **Math:** add simplifying assumptions
  - **Software:** limit the functionality
- Can you solve a more general version of the problem?
  - Usually more general problems are harder, but not always!
  - Can help focus your efforts if the details of the original problem are irrelevant
  - If the more general case turns out to be impossible, that can highlight relevant features of the original problem
- Examine special cases
  - Easier to analyze one specific function, data instance, algorithm, etc. than the "general case"
  - Look for patterns

# Generic Research Strategies

Think adversarially:

- Search for reasons your approach can't possibly work
  - Especially important early on — want to find out as fast as possible if you started down a wrong path
  - **Math:** try to find counterexamples to your conjecture
  - **AI:** give your algorithm information for free which you're hoping it will be able to infer — if it still doesn't work, then you know to give up
    - E.g., if you think inferring 3-D geometry will help with object recognition, try training a classifier on images labeled with 3-D information
- Try to break your method
  - **Software:** search for hard edge cases
  - **Experimental design:** look for confounds, alternative explanations for your results
- You often see a lot of new insights when you put on the adversary hat
  - Your failure to come up with counterexamples may give hints to why a statement is true

# Generic Research Strategies

Be careful:

- There are many different things that can go wrong and ruin an experiment, mathematical analysis, etc.
- **Software:**
  - Use version control
  - Write unit tests and possibly regression tests
  - Be able to reproduce your past results
  - Keep your core, well-tested experimental code separate from your throwaway scripts
  - Keep the configuration separate from the algorithmic logic
- **Math:**
  - Check the results of complicated derivations numerically
  - Sanity check your formulas (correct units, plausible direct/inverse relationships, etc.)
  - Plug in special cases
- **General:**
  - Determine error bounds for all your measurements
  - Reproduce past results
  - Run some experiments where you *know* how they should turn out (since these can reveal bugs)

# Generic Research Strategies

Figuring out why something doesn't work:

- Many different reasons an algorithm might not work well, e.g.
  - there's a bug in your code
  - the information it's exploiting isn't very useful for the problem
  - the information it's expoiting is already captured by other methods
  - the approximations you're making degrade the accuracy too much
  - your hyperparameter search picks a configuration that effectively disables your algorithm
  - your method actually works, but you're using the wrong metrics
  - plus many more that are specific to your problem
- Spend some time brainstorming all the possible explanations you can think of
- Then play "20 Questions", designing experiments to distinguish the hypotheses

# Generic Research Strategies

Getting help from your mentor:

- ~~"What should I do now?"~~
- Be as specific as possible
  - If your algorithm isn't working, what specific experimental results lead you to conclude it's not working?
  - Explain what you've already tried
  - Come prepared with a list of ideas — it's easier for your mentor to help prioritize than to answer your question from scratch
- Make an effort to communicate things clearly
  - Beware the illusion of transparency: you've been working on something for weeks/months, so things that are obvious to you won't be obvious to other people
  - Make clear, clean visualizations (axes clearly labeled, etc.)