

Solve Computational Problems

Easily, Reliably, Quickly, and at Scale



— Taha Rostami, 2025

1. Decision vs Optimization Problems


Tooooooooooooooooooooo many problems in computer science (and way beyond!) can be framed as either decision problems or optimization problems.


- **Decision Problems** → You just want a YES or NO answer.

Example: Is there a way to drive from my house, visit the grocery store, the gas station, and the post office, and return home in less than an hour?

- **Optimization Problems** → You're after the best version of something: shortest, longest, cheapest, most efficient...

Example: What's the shortest route to visit the grocery store, the gas station, and the post office, and return home, while minimizing the total time spent?

 You'll see these kinds of problems pop up everywhere — from games and puzzles to software design, machine learning, hardware, computational math, operations research, physics, etc.

 First smart move when facing a problem:
→ Can I frame it as a **decision** or **optimization** problem?

Once you do that, you unlock a powerful toolbox of algorithms and techniques to help solve it.

2. Picking the Right Algorithm: Complete vs Incomplete

Alright, you've got your problem framed. Now comes the fun part: choosing how to solve it. Not all algorithms are made equal — some go for guaranteed results; others go for speed and practicality.

Guarantee?	Methodology	Method
YES	Systematic & Exhaustive Search	Backtracking, A*, Branch & Bound...
NO	Local & Stochastic	Hill Climbing, Genetic Algorithms, Beam Search...

- Complete algorithms: Reliable, and often surprisingly fast. They always find a solution if one exists (and the best one, if optimizing).

- Incomplete algorithms: They may do great in practice, but there are no guarantees.

 Second smart move: → Do I need guarantees? Or is "good enough" actually good enough?

3. Coding It Up: Reality Check

Once you've framed your problem and chosen a suitable methodology, you'll need to select an algorithm to actually solve it. This could mean writing your own code from scratch or using someone else's solver. But here's the catch:

- Writing a **correct** and **efficient** search algorithm is hard.
- Maintaining and upgrading it later is even harder — swapping in a new algorithm might mean rewriting large parts of your code.

 **Now imagine this instead:**

Just like **SQL** allows you to ask complex questions about **data** without writing everything from scratch, what if there were **logical languages** where you could simply describe your problem — and let a solver handle the **reasoning** and find the solution for you?

- You describe your problem using a logical language, and solvers take care of finding the solution.
- Over time, solvers improve, and you get better performance without changing a thing.

 **Even cooler: Some tools give verifiable results.**

- If they **find a solution**, they issue a **certificate** confirming it.
- If they **don't**, they offer a **proof** explaining why no solution exists.

This is where logical languages (like propositional or first-order logic) shine. You describe the what, not the how, and let smart solvers do the heavy lifting.



So, they give you clarity, reusability, and trust — with far less pain down the road.

 Third smart move:

→ *Use logic-based tools.*

4. Then How to Start?

Start with the essentials:

- **Lecture Notes: Computational Mathematics, Lecture 7 – Satisfiability Solving**
 <https://cm.curtisbright.com/07-satisfiability-solving.pdf>
A quick and handy introduction to SAT solving.
- **Hands-On Tool: PySAT — A Python Toolkit for Prototyping with SAT Oracles**
 <https://pysathq.github.io/>
Try out SAT solving in Python.

Need Running Examples?

📌 I've solved a collection of problems here: <https://taharostami.github.io/SATLog/>

5. Moving Forward Further?




Ready to Dive Deeper?

- **Course: Declarative Programming (CS-E3220)**
 Tommi Junttila, Aalto University (Autumn 2020)
 Notes: <https://users.aalto.fi/~tjunttil/2020-DP-AUT/notes-sat/index.html>
- **Course: Automated Reasoning (CS433)**
 Ashutosh Gupta, IIT Bombay (2020)
 YouTube: <https://www.youtube.com/@automatedreasoning7411>
- **Reference Book:**
 *Handbook of Satisfiability - Second Edition*,
Armin Biere, Marijn Heule, Hans van Maaren, Toby Walsh (Eds.), 2021

Writing Your Own Solver?

 **Read: MiniSAT Paper — Niklas Eén and Niklas Sörensson, *An Extensible SAT-solver*, 2003.**

Want to Work with More Expressive Solvers?

-  Z3Py Tutorial with examples: <https://ericpony.github.io/z3py-tutorial/guide-examples.htm>
-  Programming Z3 by Nikolaj Bjørner (Talk), Simons Institute Boot Camp, 2021,
<https://www.youtube.com/watch?v=TgAVIqraCHo>
-  Programming Z3 by Nikolaj Bjørner:
<https://theory.stanford.edu/~nikolaj/programmingz3.html>

6. Gist

The following is for decision problems; almost the same applies to optimization.

