Artificial Intelligence

AI Homework

Prof. Marco Favorito
Prof. Francesco Fuggitti
{favorito,fuggitti}@diag.uniroma1.it

SAPIENZA
UNIVERSITÀ DI ROMA

Autumn Term (AY 2025–26)

Introduction

In this homework, you will be asked to **write code** that implements an entire pipeline of AI experiments:

1. modeling/implementing the problem
2. using AI techniques that we studied to solve the problem,
3. running experiments.

Finally, you will be asked to **write a final report** describing your work.

The homework can give you **up to 3 bonus points** that will be added to the mark of your exam.

The homework is optional and individual (but you can discuss it together).

The deadline is the **12th of January, 2026, Anywhere on Earth (AoE)**.

Task 1: Implementation/Modeling of an AI Problem

We ask you to pick a problem you are interested in, model it and implement it:

- n-**Queens** (scaling parameter: n)
- **"generalized 15-puzzle"** (scaling parameter: the size of the square)
- **Sudoku**: Norvig's "classic" benchmark sets: https://norvig.com/CQ/
- **Grid-pathfinding** ($n \times n$ gridworld, with walls): hand-made, or download benchmarks (see https://benchmarks.pathfinding.ai/2d-benchmarks/grid-maps/),
- **Sokoban**: you will find many benchmarks online (e.g. the original Sokoban levels).
- **Imperial Earth Calendar Puzzle**, a.k.a. "A-Puzzle-A-Day" (link to Medium blogpost)
- ...you name it! You can make a proposal, and we will review and (possibly) approve it.

If your problem has no obvious numeric size parameter (e.g., Sudoku, Sokoban), you can use a sequence of instances of increasing difficulty or size instead.

Task 2: AI Techniques

We ask you to apply/implement two different AI techniques to solve **the same problem**. One of them is fixed; the other requires the use of a General Problem Solver.

1. A*;
2. One of the following techniques, at your choice:
   - Reduction to CSP, then use a CSP solver;
   - Reduction to SAT, then use a SAT solver;
   - PDDL modeling, then use a planner.

Task 2.1: Implementation of A*

- The implementation of A* is mandatory; you cannot opt out if you want to complete the homework.
- The version you should implement is the one presented in the pseudocode of **Slide 32 of "Chapter 4 - Classical Search - Part II"**.
  - In particular: A* with duplicate elimination and no reopening.
- It is allowed (and encouraged) to use standard priority queue / heap data structures or other utilities from your language's standard library, but you must implement the A* logic yourself.
- The implementation should be modular enough to support different heuristics.
- The implementation should be modular enough to support different environments/problems as input.

Task 2.2: Implementation of AI Technique ($\neq$ A*)

For the other AI technique, you have the following options:

- **Reduction to CSP**. Use any available CSP solver. For example: OR-Tools, Z3 (also via PyZ3) ... or implement a CSP solver yourself ;-)
- **Reduction to SAT**. Use any available SAT solver. For example: CaDiCaL, Glucose, MiniSAT, etc., or use auxiliary libraries/frameworks, such as PySAT (for Python)... or implement a SAT solver yourself ;-)
- **Planning**. Model the problem as PDDL domain/problem files, and then use a planner to solve it: Fast Downward, Fast Forward, LAMA, etc. Or use the wrapper `planutils` (link to paper/video).

Task 2.2 (cont'd)

Achtung!

For CSP/SAT/PDDL, we require the solver to be integrated into your code, not just run manually.
In particular, your code should automatically:

- generate the CSP/CNF/PDDL from your problem instance and write it to a file or feed it directly to the solver;
- call the solver/planner;
- parse the solver output, and reconstruct the solution in your own representation.

You can use external libraries that handle dispatching for you, such as the previously mentioned PySAT, PyZ3, and planutils.

Task 3: Experiments

We require you to run the two algorithms (A* with one or more suitable heuristics, + your choice) on the problem you implemented/modelled multiple times, by increasing the scaling parameter, and measure (some of) the following metrics

- Running time of each run.
- number of expanded/generated nodes in the search graph;
- average/maximum/minimum branching factor;
- maximum number of nodes kept in memory (either in the explored set or in the frontier) at any given time;
- other interesting metrics returned by the CSP/SAT solvers, or by the planners;

Branching factor is only meaningful for search-based algorithms (A*, planning); you can skip it for SAT/CSP.

Final Report

Finally, you must write all your results into the final report (in PDF format, with a page limit of approximately 6-10 pages). It should contain the following information:

- **Name, Surname, Student ID, etc.**
- **Section "Introduction"**: here you summarize what you have done: your algorithms/problem choices, how you structured your solution, some interesting results, etc.;
- **Section "Task 1: Problem"**: which problem you chose, high-level description of your implementation;
- **Section "Task 2.1: Implementation of A*"**: describe your implementation, i.e., if there are some implementation details worth mentioning, how you implemented your data structures, etc.;
- **Section "Task 2.2: Implementation of X"**: same as before, but for your choice "X".
- **Section "Task 3: Experimental Results"**: plot/show the metrics we discussed earlier.
- **Section "How to run"**: how to set up the execution environment (dependencies like MiniSAT, OR-Tools, Fast Downward, etc.), which command to run, how to choose the algorithm/problem instance, and how to reproduce the experimental results.

Examples

- A* and Reduction to CSP on n-Queens;
- A* and Planning on "generalized" 15-puzzle;
- A* and Reduction to SAT on Sudoku
- ...

Some combinations make more sense than others: e.g., SAT solvers for path finding are less straightforward...

Examples

- A* and Reduction to CSP on n-Queens;
- A* and Planning on "generalized" 15-puzzle;
- A* and Reduction to SAT on Sudoku
- ...

Some combinations make more sense than others: e.g., SAT solvers for path finding are less straightforward... but nevertheless very doable! See this document: "Planning and SAT".

Other Rules

- We do not impose restrictions on the **platform** you use, nor on the **programming language** you use (Python, C++, Java, etc.).
  - We suggest using Python because we don't evaluate "code efficiency," and it's good for fast prototyping.
- The code must be provided as a **GitHub repository**.
  - If you don't have a GitHub account yet, create one.
  - To share it privately with us, follow the GitHub documentation at this link (use our `@diag.uniroma1.it` email address).
  - If you are unable to use GitHub due to specific constraints, please contact us, and we will arrange an alternative solution.
- We require you include a `README` file with:
  - a short description of the project,
  - "how to run" instructions;
  - dependencies (and how to install them)

Recap

What you have to do:

- **Task 1**: Choose one problem (e.g., n-Queens, generalized 15-puzzle, Sudoku, Grid pathplanning, Sokoban, ...) and implement it as an environment (for Planning, model it in PDDL).
- **Task 2**: Apply two AI techniques on **the same problem**:
  - A* (mandatory, implemented by you, with duplicate elimination and no reopening);
  - One among: Reduction to CSP + solver, Reduction to SAT + solver, PDDL + planner.
- **Task 3**: Run experiments by scaling the problem / using harder instances, collect metrics (time, nodes, etc.), and compare the two approaches.
- **Final Report**: 6–10 pages (PDF) + GitHub repo with code and README (how to run, dependencies, etc.).

Other rules:

- Optional, individual homework (discussion with colleagues is allowed).
- Up to **3 bonus points** on the exam.
- We will carry over the homework bonus points to every exam session of this 2025/2026 edition of the course.
- Deadline: **12th of January 2026 AoE**. You can send your HW anytime before the deadline.

Conclusion

- The homework is intentionally **open-ended**; feel free to fill in the missing gaps as you see fit.
- We would like you to be self-motivated to produce high-quality work that you enjoy doing. See this as an opportunity to finally apply the theory you saw in the course.
- Don't stress yourself; write to us if you have any doubts or concerns.
- We care much more about the "bigger picture" (your work conceptually "makes sense") rather than tiny bugs, imperfections, sub-optimal implementation, etc.
- It is acceptable to seek help online, ask ChatGPT, or discuss with colleagues; however, each student must write their own code and submit their own report.
- If you use substantial external code or tutorials, please cite them in the report.