Project - Intelligence Artificielle 2022/2023

Deadline: January 2nd 2023

n-sliding puzzle

You are going to program a solution for the sliding block puzzle with n tiles. We recall that in the *n-sliding puzzle* there are n tiles (labeled from 1 to n) and an unoccupied position arranged in a square grid. It is possible to slide the tiles horizontally or vertically into the empty space. The goal is to retrieve the configuration where the tiles are ordered with the empty cell in on the corners (without loss of generality, assume that the empty space has to be placed on the bottom right corner).

Common versions are the *15-sliding puzzle* (also known as the *15 puzzle*, and known in French as « Taquin ») with 15 tiles in a 4x4 grid, and the *8-sliding puzzle* with 8 tiles in a 3x3 grid.

12	1	2	15
11	6	5	8
7	10	9	4
	13	14	3

An instance of the 15-sliding-block puzzle

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

A solved 15-sliding-block puzzle

You will solve this problem using the classic search methods seen in class. In particular you will implement informed search using A* and compare it to uninformed search.

The project is done in teams of 2 or 3 students. You will need to declare the composition of your team using a provided form on Moodle (you will be given further instruction later).

Requirements

First, should *formalize* the puzzle as a search problem, defining the STATES, ACTIONS, GOAL test and PATH cost (use discrete structures and/or pseudocode).

Then, you will solve the problem using A* considering the two heuristics seen in class:

- h_1 : number of displaced tiles, and

- h_2: sum of Manhattan distances.

You will compare the execution of informed search (A*), with h_1 and h_2 , and uninformed search (Uniform Cost search/breadth-first search).

***Caution**: if your team involves 3 students, you will also have to implement a third algorithm, IDA*.*

Use an objected-oriented programming language of your own choice. You will provide a program that can be executed from the command line (notebooks, such as Jupyter, are not allowed).

You should provide a detailed report about your work. The report should clearly describe your program design and your implementation choices. Moreover, you should describe the details of the execution of the algorithm on sample instances. You will also provide experimental results comparing the algorithms with respect to different instances.

The development of a GUI is not required. However, testing your program should be easy and intuitive for others. The program should also include a method to generate random problems and a procedure to read a puzzle configuration from a text file.

Caution*: reference any resource you use (papers, technical reports, code,…) in your report.*

The project will be submitted using Moodle. You will submit an archive file (ZIP file or Gzipped tarball) including the source files, the report, and the instructions on how to compile (if applicable) and run the program.

***Approximate* grading scheme (can be changed without notice):**

- Analysis and formalization of the problem and of the heuristics, 4 points
  - The formalization should be as precise as possible and pave the way for the program design
  - You can use pseudocode for the function *Results*, etc
- Design and implementation of the required algorithms and auxiliary functions: 12 points
  - The implementation should cover any valid value of n $(n \in \{3,8,15,24,35, \ldots\} = \{k^2-1 \mid \forall k \in \mathbb{N} \setminus \{0, 1\}\})$
  - The code must be documented (containing a README file) and well-commented
  - The code should be object-oriented and the abstractions should be consistent with the formalization (states, actions, transition model, heuristics…)
  - The implementation choices should be described and motivated in the report
- User manual and detailed presentation of example uses: 2 points
  - The examples should be meaningful so that we can understand how the algorithms work
- Experimental study and comparison of results: 4 points
  - You will test the algorithms (UCS, A* with h_1, A* with h_2; and IDA* as well if the team is made of three students) considering different problems and for different values of *n.* Compare computation times. Set a time limit and show the maximum *n* that each algorithm can solve within it.
  - To evaluate space complexity, you will compare the algorithm with respect to the size of the frontier during the execution. Again, compare with different problems and different values of *n*.
- Extra points: graphical interface (1 point), additional search algorithm such as bidirectional search (2 to 3 points).