Assignment 2

EECS 3401 A

November 6, 2020

This assignment is **due on November 22** at 23:59. Late submissions will not be accepted.

> Your report for this assignment should be the result of your own **individual work**. Take care to avoid plagiarism. You may discuss the problems with other students, but do not take written notes during these discussions, and do not share your written solutions.

Introduction

In this assignment, you are going to implement a solver to the N-Puzzle (a generalized version of the 8- and 15-Puzzle discussed in class) using three different search algorithms: A* , A* with cycle-checking, and IDA*. We are providing you with the generic implementations of these algorithms in Prolog. Your task will be to formulate the N-Puzzle as a search problem and to run experiments with these algorithms.

The N-Puzzle The N-Puzzle consists of a set of square tiles numbered 1, 2, ..., N placed on a square grid of $N+1$ cells, leaving one cell empty. The grid dimensions are $\sqrt{N+1}$ by $\sqrt{N+1}$ (which makes sense only if $\sqrt{N+1}$ is an integer). Any tile can occupy exactly one cell (can't be in-between two cells). If a tile is adjacent to the blank cell, it can be moved into that blank cell.

The game starts with the tiles scrambled, and the goal is to arrange them in the correct order as shown in Figure 1 *in as few moves as possible*, with the blank space at the end. Note that some starting configurations are unsolvable.

1	2	3
4		6
7	5	8

(a)

1	2	3
4	5	6
7		8

(b)

1	2	3
4	5	6
7	8	

(c)

Figure 1: The configuration (b) can be reached from configuration (a) by sliding tile "5" up. Configuration (c) can be reached from configuration (b) by sliding tile "8" to the left. Configuration (c) is the goal configuration.

What you are given We provide you with the following three search algorithms implemented in SWI-Prolog:

1. A* search with path checking (`astar.prolog`)

2. A* search with cycle checking (`astarCC.prolog`)

3. IDA* search with path checking (`idastar.prolog`)

All of these files use some common code from `astarcommon.prolog`. Also, some examples of usage on simple search spaces are provided (`simpleSpace.prolog`, `waterjugs.prolog`).

Part I: Programming

Question 1 [8 points] — State Representation

Decide how you wish to represent the configuration of an N-puzzle. Then write down the representation for the puzzle configurations shown in Figure 2. In the file `a2handin.prolog`, fill the predicate `init(+Name, -State)` where `Name` is the label in the figure (i.e., `a`, `b`, `c`, or `d`) and `State` is your representation of the corresponding configuration.

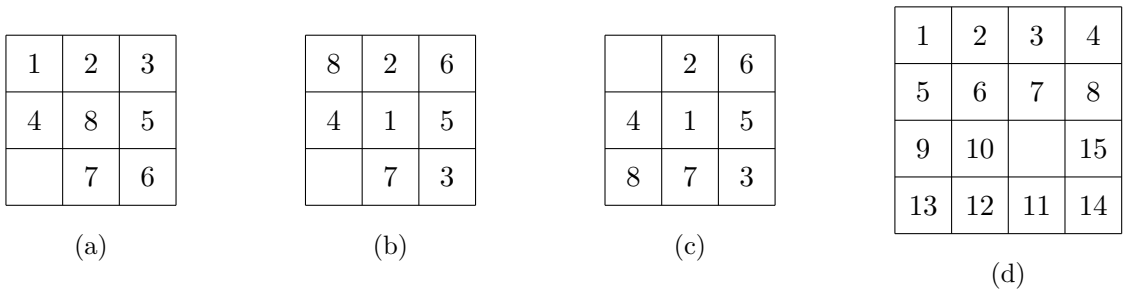

Figure 2: Four problems you will solve. Please use these names, i.e., `a`, `b`, `c`, `d`.

Question 2 [5 points] — Goal Predicate

Implement the predicate `Goal(+State)` that holds if and only if `State` is a *goal state*.

Recall: a goal state is a state where the tiles are arranged in the ascending order of their numbers, from 1 to N, left-to-right and top-to-bottom, as shown in Figure 1c for the case of $N = 8$. *Your implementation should work for any size of puzzle N.* This task may at first seem more difficult than it is. Some form of counting will do.

Question 3 [15 points] — Successors Predicate

Implement the predicate `successors(+State, -Neighbours)` that holds if and only if `Neighbours` is a **list** of elements `(Cost, NewState)`[1] where `NewState` is a state reachable from `State` by moving a tile down, left, right, or up (into the blank) and `Cost` is the cost of doing so. Assume the cost of every move is constant and equal to 1.

Question 4 [2 points] — Equality Predicate

Implement the predicate `equality(+State1, +State2)` which holds if and only if `State1` and `State2` represent the same state.

[1]Tuples like `(a,b,c)` are perfectly legal Prolog terms. You can treat them just like any other term, e.g., unify with a variable: `X = (a,b,c).`

Question 5 [30 points] — Heuristic Predicates

In the provided file `a2handin.prolog` you will find the *null heuristic* `hfn_null/2`. By design, it always returns 0 regardless of the state.

Your job is to **implement the following two additional heuristics**:

- `hfn_misplaced(+State, -V)` where `V` is the number of misplaced tiles (excluding the blank) in `State`, that is, the number of tiles which are not at the position where they should be according to the goal configuration.
- `hfn_manhattan(+State, -V)` where `V` is the sum of all the Manhattan distances between the current and the goal position of every tile (except the blank). That is, instead of just counting the number of misplacements, we also take the 'distance' of each misplaced tile to its destination into account. This is more informative and should guide our search better. The *Manhattan distance* between two positions is simply the sum of the absolute differences in the x and in the y direction.

Finally, **run the test cases** you just implemented using the various heuristics. You can do so by calling the predicates `go/2`, `goCC/2`, and `goIDA/2`. For instance, `go(a, hfn_misplaced)` will try to solve the first problem using A* search together with the heuristic made up from the number of misplaced tiles.

Part II: Analysis

The following questions do not require any programming. Edit the file `a2answers.txt` to enter the answers to the questions.

Consider a fourth heuristic defined in terms of *inversions*: For a puzzle configuration, we say that a pair of tiles a and b are *inverted* if $a < b$ but the position of b is before a in the left-to-right, top-to-bottom ordering described through the goal state. For instance, in the configuration in Figure 2a, the pairs $(5, 8)$, $(7, 8)$, $(6, 8)$, and $(6, 7)$ are inverted. We define a new heuristic $hfn\_inversions$ as the number of inversions in a configuration. So for the said configuration in Figure 2a, $hfn\_inversions = 4$.

Question 6 [5 points] — Heuristics I

Which of the four heuristics are admissible?

Question 7 [15 points] — Heuristics II

Suppose that for sliding a tile to the left we would change the cost from 1 to 0.5 and leave all the other moves the same cost.

Does this affect the admissibility of the heuristics? Which of them are admissible now? For any which is not, why not?

Question 8 [15 points] — Heuristics III

Now suppose we would change the cost for sliding a tile to the left to 2 and leave all the other moves the same cost.

Does this now affect the admissibility of the four heuristics? Again, which of them are admissible? For any which is not, why not?

Question 9 [5 points] — Performance

In the former modification (sliding to the left costs 0.5), can you say for sure which heuristic will be the fastest (expand the least number of states) in finding a (not necessary optimal) solution? Explain.

Question 10 [20 points] — Heuristics IV

One can obtain another heuristic for the N-Puzzle by relaxing the problem as follows: let's say that a tile can move from square A to square B if B is blank. The exact solution to this problem defines Gaschnig's heuristic. Explain why Gaschnig's heuristic is at least as accurate as `hfn_misplaced`. Show some cases where it is more accurate than both the `hfn_misplaced` and `hfn_manhattan` heuristics. Can you suggest a way to calculate Gaschnig's heuristic efficiently?

To hand in your report for this assignment, put the files `a2handin.prolog` and `a2answers.txt` in a ZIP archive called `a2answers.zip` and submit it **through eClass** by the deadline.