Dept. of Computer Science & Engineering
York University

Assignment 3

Total marks: 120.

Out: November 12
Due: November 26 at noon

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

In this assignment you are going to implement a solver to the N-puzzle 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.

First, a bit of background. The N-puzzle is the simple (one-person) game we discussed briefly in class where tiles numbered 1 through N are moved on a square grid of $N + 1$ cells (i.e. the grid is $\sqrt{N+1} \times \sqrt{N+1}$). Any tile adjacent to the blank position can be moved into the blank position. By moving tiles in sequence we attempt to reach the goal configuration. For example, in the figure below, we see three game configurations: 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. The objective of the game is to reach the goal configuration from some starting configuration in as few moves as possible. The goal is, independent of the size of the grid, always defined as the ordering of all tiles enumerating from left to right and top to bottom, with the blank at the lower right corner.

Note that not all starting configurations can reach the goal.

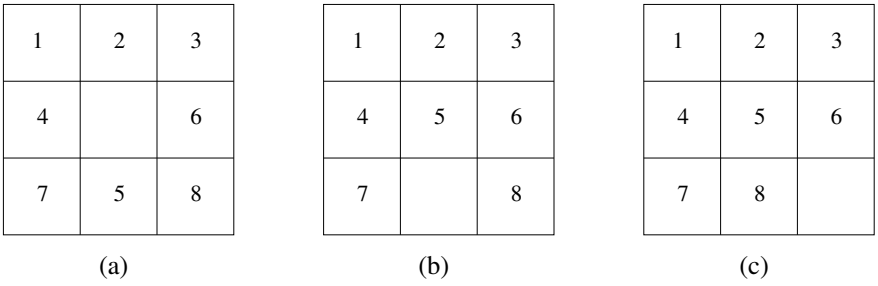

Figure 1: Three configurations of the 8-puzzle.

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

1. A^* search with path checking (`astar.pl`).
2. A^* search with cycle checking (`astarCC.pl`).
3. IDA^* search with path checking (`idastar.pl`).

All of above files use some common code from `astarcommon.pl`. Also, some simple examples of search spaces that show how these search routines are applied (`simpleSpace.pl`, and `waterjugs.pl`) are available.

The first 5 questions require you to write SWI-Prolog code. Start from the file `a3handin.pl`*, which already contains some of the code you need and fill in the missing parts. In your answers, do not use features of Prolog that have side effects such as* `assert` *and* `retract`.

Question 1. (8 marks) State Representation

Decide how you want to represent the configuration of a N-puzzle. Then write down the representation for the puzzle configurations shown in Figure 2. Fill the corresponding predicates in `a3handin.pl`, `init(+Name, -State)` where `Name` is the letter in the figure (i.e. `a`,`b`,`c`, or `d`) and `State` is your representation of the configuration.

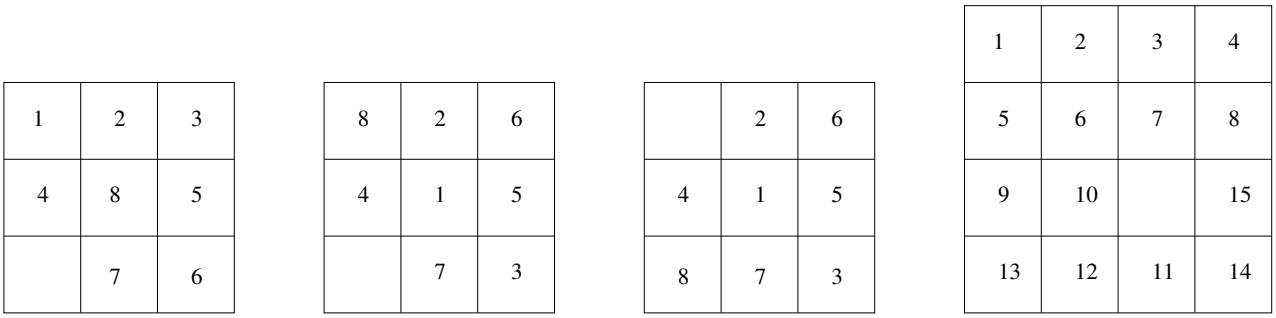

Figure 2: Four problems you will solve. Please use these names, i.e. `a`, `b`, `c`, and `d`.

Question 2. (5 marks) Goal Predicate

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

Note that we require you to implement this predicate, as well as the ones for questions 3, 4, and 5, so that it works for any size of puzzle. This may at first seem more difficult than it is. Some form of counting will do!

Question 3. (15 marks) Successors Predicate

Implement the predicate `successors(+State, -Neighbors)` that holds if and only if `Neighbors` is a list of elements `(Cost, NewState)` 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 that the cost of every move is constant and equal to 1 in this problem.

Question 4. (2 marks) Equality Predicate

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

Question 5. (30 marks) Heuristic Predicates

In `a3handin.pl` the null heuristic `hfn_null/2` is already given. In addition, implement the following two 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, 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.

The next 6 questions don't require any programming. Edit the file `a3answers.txt` *to answer the questions.*

Consider a forth 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 2 (a) 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 2 (a) hfn_inversions = 4.

Question 6. (5 marks) Heuristics I

Which of the four heuristics are admissible?

Question 7. (15 marks) Heuristics II

Suppose 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 marks) 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 marks) 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 marks) 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?

Submit your report for this assignment electronically by the deadline. To submit electronically, use the following Prism lab command:

```
submit 3401 a3 a3handin.pl a3answers.txt
```

Your Prolog code should work correctly on Prism.