Assignment: The 8-puzzle and A*

COM1005 Machines and Intelligence
Semester 2: Experiments with AI Techniques

HEIDI CHRISTENSEN
©UNIVERSITY OF SHEFFIELD
VERSION 1.1

This assignment carries 30% of the marks for COM1005

DUE DATE: Wednesday 4th May 2022 at 15:00 (UK Time)

In this assignment you will experiment with search strategies for solving the 8-puzzle problem.

Rules

Unfair means: This is an individual assignment so you are expected to work on your own. You may discuss the general principles with other students but you must not work together to develop your solution. You must write your own code. All code submitted may be examined using specialized software to identify evidence of code written by another student or downloaded from online resources.

The 8-puzzle sliding game

Figure 1 shows an example of a start configuration of the 8-puzzle sliding game and the *target* or goal state you want to end up with, where all the tiles are arranged in number order. The rules of the game are simple: you can slide any tiles neighbouring the empty gap into that gap.

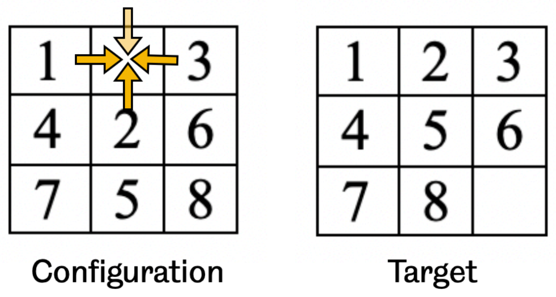


Figure 1: *Rules of the 8-puzzle game: Rearrange the tiles to the desired pattern by sliding adjacent tiles into the space. Note that when the space is at the edge, only three moves are possible.*

The problem

In this assignment you will implement a solution to the 8-puzzle by state-space search, using the search engine described in the lectures. You should experiment with search strategies as described below and write the results up in a short experimental report, using LaTeX.

LaTeX is a document preparation program used for most technical writing - see the lab class in week 8 for a brief tutorial on how to use it including the online editor Overleaf.

Provided code

The provided `searchEngine` folder contains two subfolders (`BFS` and `Astar`) that each contain the Java classes you have become familiar with in the lab classes - they are the same as the ones used for e.g., the jugs search and the map traversal lab class. They implement a simple search engine with depth-first, breadth-first, branch-and-bound and A* search strategies to choose from.

The Astar subfolder also contains an additional class (`EpuzzGen`) that will allow you to generate a random 8-puzzle. You use it like this:

```
    // generator given seed
    int seed = 23456;
    EpuzzGen gen = new EpuzzGen(seed);

     // generate puzzle providing difficulty
    int d = 6;
    int[][] puzz = gen.puzzGen(d);
```

The parameter provided (seed='12345') is a random seed which you choose. You can miss it out, but if you include it you will be able to try the same set of puzzles with different strategies. This is useful when you want to compare different algorithms.

The puzzle is returned as a 3 by 3 matrix of `ints`. The empty tile is represented by a 0. The parameter ('6' here) allows you some control over the *degree of difficulty* of the problem – i.e., how much search will be required to find the solution, on average. The higher it is, the more difficult. Don't go below 6; 12 is hard.

You can call `puzzGen()` as many times as you like. Each time it will give you a new puzzle.

Only 50% of 8-puzzles have a solution, but there is a way of deciding whether a given puzzle is solvable. That check is coded into `puzzGen` so all the puzzles it generates are solvable.

What you must do

Using the Java code provided for the assignment do the following **five tasks**:

Task 0: Set up a GitHub (or similar) repository for keeping versions of your code as you develop your solution. The repository should be private[1]

[1] Uploading copyrighted material to a public website can be considered as grounds for an unfair means case by the University.

and you do not need to share it with me unless I request to see it. Make sure to push updates regularly. The purpose of using a git repository is twofold: i) to develop good habits and practice around version control, and ii) in case of a suspicion of unfair means, you have clear evidence that code was developed along the way by yourself. You must add the url of your github repository to your LaTeX report (I've added a nifty little footnote to the title where this info should go (marks are deducted if this is missing)).

Task 1: Implement and test the 8-puzzle search for breath-first search The `searchEngine/BFS` folder contains the code for the simple search engine used for e.g., the jugs problem.

The search engine in the `searchEngine/BFS` folder implements the following search strategies:

- Breadth-first
- Depth-first

The search engine in the `searchEngine/Astar` folder implements the following search strategies:

- Branch-and-Bound
- Astar

In this case we won't consider depth-first, and branch-and-bound is the same as breadth-first because we have uniform costs. We will compare breadth-first with 2 variants of A* (see later tasks).

You should not need to change the code for the search engine except perhaps to control how much it prints as a search proceeds, and to stop the search after a given number of iterations (in an 8-puzzle the search tree can become large).

To implement the 8-puzzle breath-first search (BFS) you will need to define a class `EpuzzleState` and a class `EpuzzleSearch` (note: we can't have files beginning with an "8" so use "E" instead). You will also need a class for running the tests. Call this `RunEpuzzleBFS`.

To test your implementation of the **breadth-first** search you should test the code with the initial configurations (P1, P2, P3) and the same goal state (target) as shown in Figure 2.

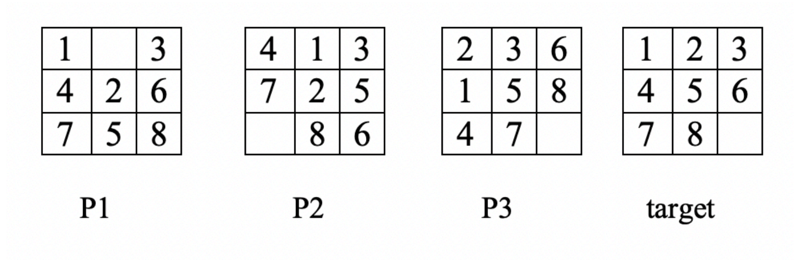


Figure 2: *Three test patterns for your BFS and the target configuration.*

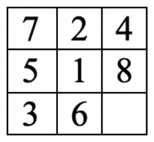
Add the code for this test to `RunEpuzzleBFS`.

Task 2: Implement the 8-puzzle for Astar In the `searchEngine/Astar` folder, you now need to adapt your 8-puzzle classes for A*. This means that your 8-puzzle state must now include a local cost **g** (always 1 for the 8-puzzle), and an estimated remaining cost `estRemCost`, which is used by A* and must be an underestimate of the true cost.

Code two alternative methods in your 8-puzzle state class for computing `estRemCost`, assuming that the target pattern is always the same, as above.

- **Hamming** distance, which is defined as *the number of tiles out of place.*
- **Manhattan** distance, which is defined as *the sum of the moves each tile needs to make before it is in its correct position.*

Figure 3 shows an example of how these two distances should be calculated for a particular tile configuration.



Hamming: 7 (all out of place except 2)
Manhattan: 16 (2+0+4+3+1+2+2+2)

Figure 3: *Illustration of the* `estRemCost` *for a particular puzzle configuration using Hamming and Manhattan distances, respectively.*

Task 3: Compare the efficiency of BFS and A* The experiment is to compare the efficiency of breadth-first, A*(Hamming) and A* (Manhattan) over a number of puzzles of varying difficulties. You are testing the hypothesis that:

> *A* is more efficient than breath-first, and the efficiency gain is greater the more difficult the problem and the closer the estimates are to the true cost.*

Perform experiments to test this hypothesis.

CAUTION: 8-puzzle search trees can be surprisingly large. You may have to wait a long time for a search to conclude.

Task 4: Produce a report Your experimental report should describe your implementations and your results. Your report should include at the very least:

- Description of your domain specific implementations for the 8-puzzle domain.
- Presentation of results obtained when testing the efficiency of the two search strategies (breadth-first and A*) for different ways of estimating the remaining cost (using Hamming or Manhattan distances) and different difficulties of puzzles.
- A comparison of the results - can you verify your hypothesis?

A LaTeX report template is provided, with compulsory sections specified. You may add more sections if you wish: include in your report what you think is interesting. Technical writing is to the point though, so do not feel the need to add long narratives.

Note 1: *you **must** use the LaTeX template provided for your report. However, you will not be marked on your ability to use LaTeX to format your report (i.e., there are no extra marks for making it look fancy!). That being said, LaTeX is an essential piece of software for communicating research and results in engineering and science, so we want you to get experience using it early on. Make sure to submit the pdf of your report, not the .tex file*

Note 2: *you should not have to make any changes to the Java code provided except to control the amount of printout during a search and perhaps to modify what a successful search returns. It's a good idea to revisit the week 1-3 lab classes for a reminder of how you worked with the different search strategies.*

Note 3: *please add this declaration to the top of your report: "I declare that the answers provided in this submission are entirely my own work. I have not discussed the contents of this assessment with anyone else (except for Heidi Christensen or COM1005 GTAs for the purpose of clarification), either in person or via electronic communication. I will not share the assignment sheet with anyone. I will not discuss the tasks with anyone until after the final module results are released."*

Summary of assignment

1. Implement a breadth-first and an A* search for the 8-puzzle problem using the code provided.
2. Run your code with different different estimates for the remaining costs and different difficulties of puzzles to assess the efficiency of the two types of search algorithms as described above.
3. Complete your report with the results and conclusions.

What to submit

Submit your solution, `as a zipped file` called `SURNAME_FIRSTNAME.zip`, where `SURNAME` is your surname (family name), and `FIRSTNAME` is your first name (given name). This zip file should contain:

- a `.pdf` copy of your report named `SURNAME_FIRSTNAME.pdf`
- your code (include the full `searchEngine` folder including the files provided with the assignment.

Marking Scheme		
Percentage points		Categories and example score weights; Marks will be awarded according to these criteria:
50%		**Implementation of breadth-first and A* search**
	25%	Sensible implementations of `EpuzzleState` and `EpuzzleSearch` for BFS and A* and the two respective test classes. Note: marks will be deducted if your code doesn't compile and run on the command line. You can use your favourite IDE but make sure that I can compile & run your code from the command line in the `searchEngine` subfolders like this for BFS "javac RunEpuzzleBFS.java" and "java RunEpuzzleBFS". You will lose marks, if I can't do this without modifying code.
	15%	Sensible and correct implementations of Hamming and Manhattan distances.
	10%	Well documented and well presented code. This is about using comments to ease understanding of more complex parts of the code, and the consistent use of indentations, brackets and appropriate choices of variable names.
30%		**Experimental results**
	30%	Good assessment of efficiency of BFS and A*. More points are given for exploring a good number of different difficulties.
20%		**Report: high quality of presentation and communication of your experimental results in your report**
	10%	Quality of communication of results in e.g. tables and figures
	10%	Quality of discussion of results and conclusions
Total: 100%		

Marks and feedback will be returned through Blackboard within 3 weeks.