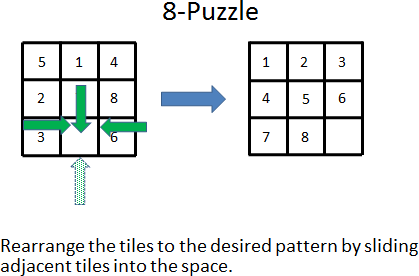
Machines and Intelligence Experiments with AI Techniques The 8-puzzle and A\*

# The 8-puzzleproblem



Target

In this assignment you will implement a solution to the 8-puzzle by state-space search, using the search engine described in the lectures, and experiment with search strategies.

# What you mustdo

* 1. *Implementing 8-puzzlesearch*

In the **search2** folder in the Java download on the COM1005 MOLE site is the code for the simple search engine and for Jugsproblems.

*Note that you may have to recompile the code for your version of Java.*

* + - Following the instructions in section 2.9 of the lecture notes, write classes to imple- ment a state-space search for 8-puzzleproblems.
    - You should not need to change the code for the search engine except perhaps to con- trol 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 becomelarge).
    - Test your implementation with **breadth-first** searches for the following initial con- figurations and the same goal state asabove:

|  |  |  |
| --- | --- | --- |
| 1 |  | 3 |
| 4 | 2 | 6 |
| 7 | 5 | 8 |

|  |  |  |
| --- | --- | --- |
| 4 | 1 | 3 |
| 7 | 2 | 5 |
|  | 8 | 6 |

|  |  |  |
| --- | --- | --- |
| 2 | 3 | 6 |
| 1 | 5 | 8 |
| 4 | 7 |  |

P1 P2 P3

* 1. *Experimenting with SearchStrategies*

The search engine in the **search4** folder implements the following search strategies:

* + - Breadth-first
    - Depth-first
    - Branch-and-Bound
    - A\*

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’ll compare Breadth-first with 2 variants of A\*.

* + - Adapt your 8-puzzle classes for search4. 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 forcomputing

## estRemCost, assuming that the target pattern is always the same, as above.

* + - * **Hamming** distance, which is the number of tiles out ofplace.
      * **Manhattan** distance, which is the sum of the moves each tile needs to make before it is in its correctposition.

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

|  |  |  |
| --- | --- | --- |
| 7 | 2 | 4 |
| 5 | 1 | 8 |
| 3 | 6 |  |

* + - In search4, the class **EpuzzGen** will generate random 8-puzzles for you.Usage:

EpuzzGen gen = new EpuzzGen(12345); //create 8 puzzle generator

*The parameter (‘12345’ there) 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*

int[][] puzz=gen.puzzGen(6); //generate a puzzle

The puzzle is returned at a 3 by 3 matrix. The empty tile is 0.

*The parameter (‘6’ there) 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 .

* + - You won’t need the usual printouts in a search, so there is a variant ofrunSearch, **runSearchE**, which does no printing and returns the efficiency of the search as a float. Failure returns0.

**The experiment is to compare the efficiency of breadth-first, A\*(Hamming) and A\* (Manhattan) over a number of puzzles.** 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.*

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

1. **What to handin**

* **Commented code**, ready to run. Give your code for 2.1 and 2.2separately
* **Testing**: each method you write should be separately tested. **You should not run any searches until you have done this.** The tests themselves should cover every logical case and test results should be commented, e.g. **sameState**: returns true for 2 different instances with identical states, returns false for 2 instances with differentstates.
* **Experimental Results** which address the hypothesis above. Consider how best to present results: tables, graphs, bar charts? Discuss your results – what do they show? Could you improve on the A\*estimates?

# How to handin

By MOLE

Deadline Thursday 1st August 2019, 3pm.

# MarkScheme

## Basic Implementation

8-puzzlestate class 35% of the credit for this assignment 8-puzzlesearchclass 5%

Solutions to P1,P2, P3 10%

## Experiments

|  |  |
| --- | --- |
| Modifications to classes | 20% |
| Design of Experiments | 10% |
| Presentation of results | 15% |
| Discussion | 5% |

For programming, around 60% of the credit is for the quality of the code, 20% for documentation and 20% for testing.