

COMP3702/COMP7702

Assignment 1: Search

Due 5pm, Friday, 9th of September 2011

This assignment counts 10% toward the final marks.

Purpose

This assignment is intended to practically acquaint you with informed search methodology and techniques. Additionally, it should provide some understanding of how an informed search algorithm for simple games, such as the 15-puzzle and variations, can be implemented.

Code

The supporting material can be used freely in your work: [zipped source code](#), [jar package](#), and [package description](#).

The supporting code is in Java and it is recommended that you use and modify this to complete the assignment. If you do not know Java currently, you will not need to learn a great deal of Java to do the assignment, but working in a pair might be a good idea.

Administration

The assignment is worth 10% of your final grade and you may work individually or in pairs. You are required to submit two separate components for this assignment, these are:

1. A single file containing all the source code files (e.g. *.java), including any you modify. Modified sections of source code should include comments and overall, the code should compile. There is no excuse for handling in a program which does not compile/interpret or run. If you are having trouble, some features may be omitted. Package and submit all the source code (*.java), including unmodified files, maintaining directory structure as necessary. You may package the source code into e.g. a .zip or .jar file.
2. A pdf with the written component of the assignment (see [template/example \[pdf\]](#) for general guidance). Please indicate in your pdf which source code files you modified and if there are any new files. Also indicate in your pdf exactly what parts you have attempted, in part or in full, so that we know what to look for when marking.

Submission of both parts is via the ITEE online submission website found at <http://submit.itee.uq.edu.au>. Your group name should be the student number of the person submitting the assignment. You should use your student number(s) within the pdf and code, and as part of the pdf and packaged code filenames. **Note: There will be marks**

deducted if your group name does not correspond to a student number or if your PDF and code do not contain your names and Student IDs.

If you resubmit your assignment, please resubmit all files relating to your solution. Your most recent submission will be the only submission marked.

In line with University policy collusion and plagiarism will not be tolerated; see the course profile for more details on these topics.

Each group must submit an assignment cover sheet signed by all members of the group. The online submission system will offer you a printable cover sheet as you submit your assignment. Otherwise, a suitable ITEE assignment cover sheet can be obtained from <http://studenthelp.itee.uq.edu.au/assignments/>. You should submit your completed form as soon as possible after submitting your assignment (details on where to submit these forms are coming soon).

You can discuss aspects of the assignment with other members of the class on the comp3702 newsgroup (uq.itee.comp3702 or see my.uq.edu.au). If you want to ask the lecturer any questions, sending an email to ruth@itee.uq.edu.au is the best option.

The Assignment

The 15-puzzle is an apparently simple puzzle for a single player, containing 15 numbered tiles and a blank space on a 4x4 board. The puzzle is set up in a random configuration and the goal is to move the tiles around until an ordered goal state is reached, having the tile numbers in order across and down the board, with the blank in the bottom right-hand corner (see below). The only moves available are to move a tile into a vertically or horizontally adjacent empty square. The state space for this problem is quite large, containing approximately 10^{13} states, so exhaustive search is a poor problem-solving strategy. You can play with an [applet](#) for inspiration. The 15-puzzle is a version of the n-puzzle.

| | | | |
|----|----|----|----|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 0 |

A variation of the 15-puzzle is the quad-4-puzzle, in which the 15 numbered tiles are numbered between 1 and 4. The goal state is with the 4 tiles numbered 1 in the top left corner, the 4 tiles numbered 2 in the top right corner, the 4 tiles numbered 3 in the bottom left corner, and the 3 tiles numbered 4 in the bottom right corner with the blank in the bottom right corner:

| | | | |
|---|---|---|---|
| 1 | 1 | 2 | 2 |
| 1 | 1 | 2 | 2 |
| 3 | 3 | 4 | 4 |
| 3 | 3 | 4 | 0 |

Source code has been provided which implements a breadth-first search strategy to solve the 15-puzzle and the quad-4-puzzle (see Code above).

In this assignment, you are asked to implement heuristic search algorithms to solve the 15-puzzle and the quad-4-puzzle, including the use of your own heuristic functions, and to document and describe the algorithms and their behavior.

The following is the marking scheme. You should carefully document all the parts of the assignment in a single PDF. Your programs should be well designed and should use meaningful variable names, correct indentation of code, etc.

| Part | Description | Marks |
|------|--|-------|
| 1 | <p>Problem Representation</p> <p>Formally represent the quad-4-puzzle problem.</p> <p>Your assignment must contain the following:</p> <ol style="list-style-type: none"> 1. List all relevant objects in the problem description and state the possible states of the game, the initial state of the game, goal test conditions, and step cost / path cost (0.5 marks); 2. Represent the states in a well-known data structure, such as array, queue, stack, linked list, or your choice of mathematical model and describe the representation (0.25 marks); 3. Represent the state space with mathematical formulas, operations (including the successor function), and constraints and describe the representation (0.25 marks); | 1 |
| 2 | <p>Heuristic Functions</p> <p>Implement the following heuristic functions for the 15-puzzle.</p> <p>h_1 = the number of misplaced tiles</p> <p>h_2 = (Manhattan distance) the sum of the distances of the tiles from their goal positions</p> <p>Your assignment must contain the following:</p> <ol style="list-style-type: none"> 1. Formal description of the heuristics h_1 and h_2 (0.25 marks); 2. Pseudocode for the heuristics h_1 and h_2, including comments and a description (0.25 marks); 3. Correct implementations of the heuristics h_1 and h_2 in Java (0.5 marks). | 1 |
| 3 | <p>Invented Heuristic Functions</p> <p>Implement two heuristic functions of your own invention, h_3 and h_4 (1 mark each), for the quad-4-puzzle.</p> <p>Your assignment must contain the following for each heuristic:</p> <ol style="list-style-type: none"> 1. Formal description of your heuristic (0.25 marks each); | 2 |

| | | |
|---|--|---|
| | <p>2. Pseudocode for your heuristics, including comments and a description (0.25 marks each);</p> <p>3. Correct implementations of the heuristics in Java (0.5 marks each).</p> <p>You cannot use h1, h2, or the following heuristic function described in the text book:</p> $h5 = \max \{h1, h2\}.$ | |
| 4 | <p>Informed Search Algorithms</p> <p>Implement the Greedy and A* search algorithms (1 mark each):</p> <p>Your assignment must contain the following for each search algorithm:</p> <ol style="list-style-type: none"> 1. Formal description of the search algorithm (0.25 marks each); 2. Pseudocode of the search algorithm, including comments and a description (0.25 marks each); 3. Correct implementation of the pseudocode in Java (0.5 marks each). | 2 |
| 5 | <p>Repeated States</p> <p>Implement detection of repeated states:</p> <p>Your assignment must contain the following:</p> <ol style="list-style-type: none"> 1. Formal description of how to detect and avoid repeated states (0.25 marks); 2. Pseudocode showing how you detect and avoid repeated states, including comments and a description (0.25 marks); 3. Correct implementation of repeated state detection and avoidance in Java (0.5 marks). | 1 |
| 6 | <p>Evaluate</p> <p>Evaluate the heuristic functions for the fifteen-puzzle and the quad-4-puzzle with Greedy and A* search.</p> <p>Your assignment must contain the following:</p> <ol style="list-style-type: none"> 1. Effective branching factors (EBFs) for depths (solution lengths) 5, 10, 15, 20, and 25 for the heuristic functions for the fifteen puzzle and the quad-4-puzzle and both search algorithms. EBFs should be averaged over 5 runs for each depth. (1 mark) 2. Evaluation of strengths and weaknesses of the heuristic functions based on evaluation of EBFs, space and time complexity for evaluating your heuristic functions, accuracy of the heuristic functions, completeness, and optimality. (2 marks) <p>Note: You can use the supplied code for generating random puzzle states to collect 5 test results for each depth. Then, run the supplied code (in Node.java) or write and run your own code to calculate EBFs for each</p> | 3 |

| | | |
|--|--|--|
| | depth. For each test run, you need to obtain the total number of nodes generated and the depth (solution length) to obtain its effective branching factor. | |
|--|--|--|

Implementation marks are given based on the following criteria: Your program is well designed and can solve the puzzles with Greedy and A* search and your heuristic functions (within a reasonable time).

Notes:

-To assist with testing, we recommend the following restrictions on your code:

The jar/zip file must contain the classes named "**search.fifteen.FifteenSearchApp**" and "**search.quadfour.QuadFourSearchApp**" with the methods named "**solveH1G**", "**solveH1A**", "**solveH2G**", "**solveH2A**" etc. (naming: 1 = heuristic 1, 2 = heuristic 2, G = Greedy, A = A*) as in the supplied code. These should:

1. Take a PuzzleState object (a 4x4 array representation of the puzzle),
2. Solve a puzzle with your own Greedy and A* implementations (with detection and avoidance of repeated states if implemented) with the provided heuristic functions and your own heuristic functions, and
3. Return an array of actions defined in class PuzzleState. (Note: the actions in the array must be ordered so that performing each action (starting from the last element of the array and working towards the first element) on the given original state will solve the problem.)

Examples of these methods are provided. You must modify these methods to solve the puzzles with your own Greedy and A* implementations with your heuristic functions. You can extend the classes to implement your own representations of states, nodes, and actions. You will need the action definitions given in class PuzzleState in order to return valid action sequences in FifteenSearchApp.solveH_().

If you extend PuzzleState to use your own representation of states, make sure that you map the array representation of puzzle in PuzzleState into your own representation in FifteenSearchApp.solveH_(). If you use your own action definitions, you also need to map your action sequences of solutions into the actions defined in class PuzzleState when you return an array of actions in FifteenSearchApp.solveH_().

-In all cases, describe how to compile/interpret and run your program (include name and version of compiler/interpreter).

-Where possible, include your student number (if working individually) or both student numbers with a separator (if working in a pair) at the beginning of submitted filenames. Also make sure to include your name and student number(s) in your document and in any modified code files.

-If using the supplied Java code to implement your own FifteenSearchApp.solveH1G(), FifteenSearchApp.solveH1A() etc we should be able to test and evaluate your code.

-Otherwise, you need to provide basic test and evaluation code for your program and document this.

Java notes:

-Read the [package description](#) to understand the classes that you will need to change for this part.

-You may find an [Eclipse project creation tutorial](#) useful.