CMPT 310 - Artificial Intelligence Survey

Assignment 1

Due date: February 8, 2021 goal state is : 1 2 3 4 5 6 7 8 0 J. Classen
10 marks January 25, 2021

Important Note: Students must work individually on this, and other CMPT 310, assignments. You may not discuss the specific questions in this assignment, nor their solutions with any other student. You may not provide or use any solution, in whole or in part, to or by another student.

You are encouraged to discuss the general concepts involved in the questions in the context of completely different problems. If you are in doubt as to what constitutes acceptable discussion, please ask!

Software

Most of the assignments will be using Python 3[1], and you should install the aima-python[2] code on your computer (including the sample data sets). Follow the installation instructions on the aima-python page.

Since aima-python uses Git, it makes sense to download the code through Git.

The 8-Puzzle

In this assignment you get a chance to experiment with some (informed) search algorithms.

In the aima-python repository, take a look at the class called `EightPuzzle`. Take some time to read and understand it, including the `Problem` class that it inherits from.

Put the coding part of your answers to the following questions in a Python 3 file named `a1.py` that starts like this:

```
# a1.py

from search import *

# ...
```

Do not use any modules or code except from the standard Python 3 library, or from the textbook code from Github.

[1] https://www.python.org/
[2] https://github.com/aimacode/aima-python

Question 1: Helper Functions (2 marks)

Write a function called `make_rand_8puzzle()` that returns a new instance of an `EightPuzzle` problem with a random initial state that is solvable. Note that `EightPuzzle` has a method called `check_solvability` that you should use to help ensure your initial state is solvable.

Write a function called `display(state)` that takes an 8-puzzle state (i.e. a tuple that is a permutation of `(0, 1, 2, ..., 8)`) as input and prints a neat and readable representation of it. `0` is the blank, and should be printed as a `*` character.

For example, if `state` is `(0, 3, 2, 1, 8, 7, 4, 6, 5)`, then `display(state)` should print:

```
* 3 2
1 8 7
4 6 5
```

Question 2: Run A* with Statistics (2 marks)

By default, the A* implementation in `aima-python` uses the misplaced-tiles heuristic. In your program, include top-level code such that when the program is executed, it creates a random 8-puzzle problem instance (using your code above), prints the instance, and solves it by means of A*. Afterwards, it should report:

- the total running time for the search algorithm in seconds;
- the length (i.e. number of tiles moved) of the solution;
- the total number of nodes that were *expanded* during search.

You will probably need to make some modifications to the A* code to get all this data. Also, be aware that the time it takes to solve random 8-puzzle instances can vary from less than a second to hundreds of seconds (see hints below)!

Modify your program such that the user can optionally provide the initial state for a problem instance on the command line. For example, to solve `(0, 3, 2, 1, 8, 7, 4, 6, 5)`, the program should be called using

```
$ python3 ai.py 0 3 2 1 8 7 4 6 5
```

If no instance is given, the program instead creates the random instance as described above.

Question 3: Manhattan Distance Heuristic (2 marks)

Write a function `manhattan(n)` that implements the *Manhattan distance* heuristic, taking a `Node n` as its argument. Extend your top-level code such that it additionally runs A* using this heuristic function.

Be careful: there is an incorrect Manhattan distance function in `tests/test_search.py`. Don't use that, but implement your own (correctly working!) version.

Question 4: Gaschnig's Heuristic (2 marks)

One relaxation of the 8-puzzle is the assumption that a tile can move from one square to the blank position directly, even if it is not a neighbouring square. The exact solution of this problem defines *Gaschnig's heuristic*. Implement it in a function called `gaschnig(n)`, and again extend your top-level code such that it additionally runs A* with this heuristic.

Question 5: Compare Algorithms (2 marks)

Create 10 (more would be better!) random 8-puzzle instances (using your code from above), and solve each of them using the 3 different heuristics for the A* algorithm. Each version should be run on the exact same set of problems to make the comparison fair.

Write a PDF document in which you discuss your results. It should at least contain a (single) table that summarizes your data. Be sure to include helpful descriptive statistics like the min, max, average, and median values. You are also encouraged to include helpful or informative graphs of your data.

Furthermore, in your PDF, answer the following questions:

- Based on your data, can you give a recommendation for what you think is the best algorithm? Explain how you came to your conclusion.
- Is it possible to rule out one of the heuristic functions without looking at experimental data, but purely based on theoretical consideration? Hint: We say that an admissible heuristic h_1 is *at least as accurate* as another admissible heuristic h_2 iff for every n, $h_1(n) \geq h_2(n)$.
- Can you suggest a heuristic that is always at least as accurate as all 3 heuristic functions discussed here?

What to Submit

Please put all your code into a single Python 3 file named `a1.py`, all your data, tables, charts, discussion, etc. in a PDF file named `a1.pdf`, and submit them on Canvas before the due date listed there.

If you want to make changes to code in `search.py`, or in files other than `a1.py`, please copy the code you want to change into `a1.py`, and change it there. Don't make any changes to other files in the textbook code.

Hints

- When you are testing your code, you might want to use a few hard-coded problems that don't take too long to solve. Otherwise, you may spend a lot of time waiting for tests to finish!

- One easy way to time code is to use Python's standard `time.time()` function, e.g.:

  ```
  import time

  def f():
     start_time = time.time()

     # ... do something ...

     elapsed_time = time.time() - start_time

     print(f'elapsed time (in seconds): {elapsed_time}s')
  ```

  Python has other ways to do timing, e.g. check out the `timeit` module.

- To process command line arguments, use the `sys` module:

  ```
  import sys

  print('Number of arguments: ', len(sys.argv))
  print('Argument List: ', sys.argv)
  ```

  Running the above as file `test.py` using

  ```
  $ python test.py arg1 arg2 arg3
  ```

  yields

  ```
  Number of arguments: 4
  Argument List: ['test.py', 'arg1', 'arg2', 'arg3']
  ```

  Note that the filename of the program is considered the first argument.

- If you download the textbook code as a Git repository, then you can create a branch called `a1` for this assignment, e.g. something like:

  ```
  $ git branch a1 git checkout a1
  ```

  This way you can make any changes you like while maintaining a copy of the original code. You are not required to use Git, but since the code is stored as a Git repository, it's probably the best way to get it.