COMP2611
Artificial Intelligence

Assignment 1: Search Algorithms

Astu Dent — `scs2000ad@leeds.ac.uk` (**Submitter**)[1]
Anne Otheren — `scs2001ano@leeds.ac.uk`

You must choose ONE of the following declarations, which describes your situation and include it on the front page of your submission. It is expected that nearly all of you would use declaration A. You may opt for B if there is some reason that makes working in a pair difficult or unsuitable for you. Hopefully very few, if any, of you will be putting declaration C. But please do inform module staff if some issue arises with your pairing. We may be able to fix things or repair you.[2]

Declaration A.
We confirm that we have worked as pair on this project and both of us have made significant contributions to both parts of the assignment.
We are aware that both members of the pair will receive the same grade.
We confirm that **only one** submission submission has been made via Gradescope.
The submitter confirms that they have agreed the final submitted version of this report with the other member of the pair.
The submitter confirms that, after submitting the report to Gradescope, they have added the other member of the pair to the group associated with the submission.

Note: Only one member of the pair should submit on Gradescope and should then add the other member. After submission, you will see a page with a button that enables you to select another group member.

Declaration B.
I confirm that I have chosen to undertake this project on my own rather than as part of a pair. I understand that the grading criteria are the same for individual submissions as for subissions in a pair.

Declaration C.
I have not been able to do this project as part of a pair, because of some reason beyond my control. I confirm that I have made the module leader aware of this issue as soon as the difficulty became apparent and have received appropriate help and guidance regarding my situation.

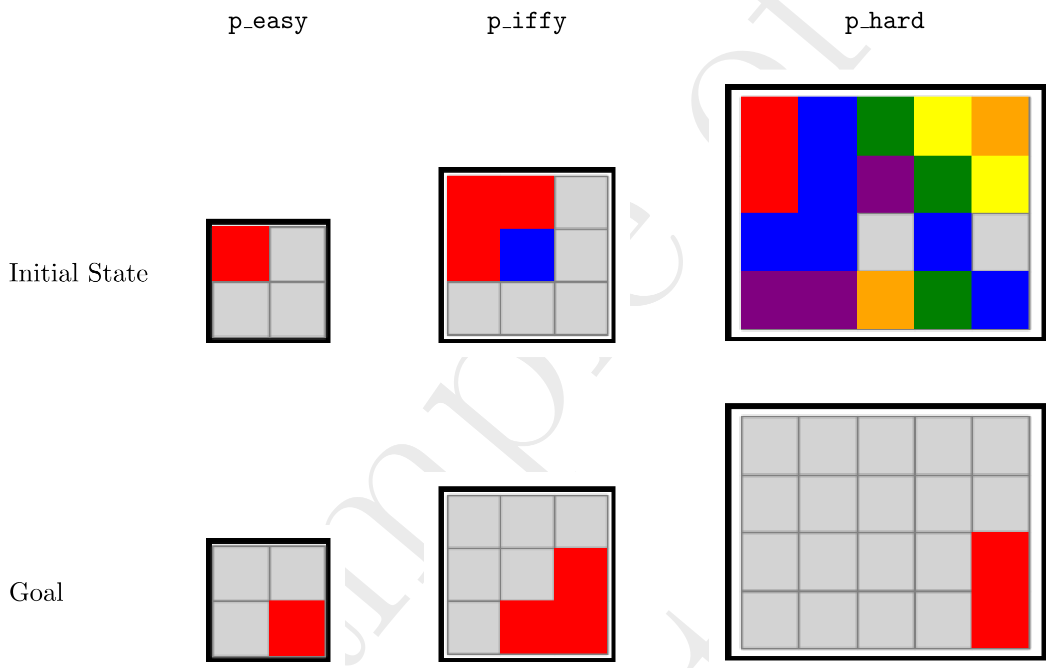
Note: If you do have some problem with contacting and/or collaborating with your partner, it is important that you contact the module leader as soon as possible. It is expected that any such issue be raised at least a week before the submission deadline.

[1] *If you were working as a pair you need to indicate the submitter.*
[2] Bear in mind that you don't need to find a partner yourself. You will be automatically assigned a random partner if you have not formed a pair by the end of the week in which the assignment was released.

A. Sliding Blocks Puzzle Search Investigation

A1(a) Puzzle Test Cases

After some experimentation we decided to investigate the following cases of Sliding Block Puzzle:



Note: those examples are not good ones for your actual experiments. The `p_easy` *is far too easy.* `p_iffy` *is silly and impossible.* `p_hard` *is also almost certainly impossible. You need to find cases that differentiate capabilities of the algorithms. The search algorithm should be able to work for quite large and complex problems, but of course its performance will vary a lot depending on algorithm settings and heuristics used.*

You can easily save the `BlockState` *images that are generated by the code in* `SearchExercise7.ipynb` *(right-click on the image in the notebook), and include these in your report.*

A1(b) Heuristics

*The following gives some **silly** examples. Do not use use those! They just show a typical layout of how you should give the heuristics in your report. You paste in the code and give a very brief description of how it works.*

You can actually just use heuristics very similar to those examples given in the `SearchExercise7.pynb` *file. That would be sufficent for a good overall mark if you do a good set of tests and a nice table of results. But for a top mark you would probably need a more accurate and/or elaborate heuristic.*

We designed the following two heuristics to use for our investigations and test results:

Drop 'n' Grab Huristic

Our `drop_n_grab` heuristic is defined as follows:

```
def drop_n_grab( state ):
     drop.all( [tiles in state] )
     for tile on floor as it:
         grab it
     return num(grab.s)
```

Our algorithm simply drops all the tiles on the floor and counts how many grabs are required to pick them up.

Manhattan Spider Heuristic

Our `manhattan_spider` heuristic is defined as follows:

```
def manhattan_spider( state ):
     swing:
        from:
           skyscraper
        to:
           skyscraper
     return num(swing.s)
```

The algorithm uses a `swing` block to iterate over Manhattan. At the end of the swing transit the number of swings is returned.

A1(c) Search Algorithm Test Sequence

After experimenting with various search options we found that the following sequence of tests gives an informative set of statistics regarding the performance of a wide range of search algorithms and options, when applied to the 8-Puzzle sliding tiles problem:

```
## Our testing code:
puzzle_1 = ....

search( puzzle_1, ..., ... ... )
 :
 :
 :

puzzle_2 = ....

search( puzzle_2, ..., ... ... )
 :
 :
 :
```

For this answer you can just paste in the code that your ran to get your test results. But you may like to add a few sentences of explanation as well.

You need to decide on exactly which tests you do. But remember that the table of results needs to fit on one page. Try to get a set that covers the main options but not too many that it is difficult to understand the results.

A2. Results

The results obtained for the first ‘dum case’ problem instance were as follows:

`dum_case_101` **Results**								
	splayed				pegged			
	tips	flips	d-flips	que?	tips	flips	d-flips	que?
Nose forward	100	1	2	Y	10	11	808	$*$
Nose forward	10,000	111	1,000,123	Y	200	22	2777	$*/?$
:	:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:	:
:	:	:	:	:	:	:	:	:
Chin up	100		∞	N	0	-77	∞	N

Table 1: This is a stupid table. It just illustrates the kind of table structure you could use.

For a different test which was chosen to be floppier than the other, we got the following results:

`dum_case_102` **Results**								
	splayed				pegged			
	tips	flips	d-flips	que?	tips	flips	d-flips	que?
Nose forward	0	0	0	0	0	0	0	0
Nose forward	?	?	?	?	?	?	?	?
:	:	:	:	:	:	:	:	:
Chin up	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Table 2: This is other table shows another ‘dum case’ example.

You could have 3 or 4 tables, an you could have tables of different kinds. You need to try to make the tables show the results as clearly and informatively as you can. Don’t ask the lecturers and assistants about what should be the exact contents and layout of the tables. Deciding on that is part of the assignment.

You need to make sure ***all the tables together fit on one page.*** *And if the meaning is not obvious from the table itself you will need to add some concise explanatory sentences on the same page. (For example: ‘The symbol ‘$\emptyset$’ means my laptop ran out of battery.’)*

A3. Observations

Here are some silly observations. In your report you should list significant and/or interesting (not silly) observations that you have made regarding behaviour of the different search algorithms and options.

After examining our results, we gained deep understanding of search algorithms. Of the many interesting observations we made, the most flabbergasting were as follows:

- The observation that **Chin up** performed an infinite number of *d-flips* when pegged was truly flabbergasting, but is probably due to the flips being negative for this configuration. Negative *flips* are well-known to be problematic when applied to chin-based algorithms. This is because there are no nostrils to unblock.
- It can be seen that the **pegged** option reduced tips by at least a factor of 10 for all nose and chin positions. For this kind of problem it is clear that pegging is always beneficial since results without pegging can be seen to be worse in all cases.
- ...
- ...

You should list at least 4 observations, possibly up to 8 or more. But you need to make sure they are clear and distinct, and have some interest. Just adding many detailed small points will not gain extra marks and could actually lose marks as the overall list will become less informative.

It will be good to consider general patterns as well as individual results and also consider how the patterns and results are related to general characteristics of search strategies in relation to the problem.

B. Robot Worker Scenario

This part of the assignment is similar to Part A. However, it is more open ended and you are expected to use your own initiative of how exactly to do the task and write it up. But you still have a maximum of one page for each of the four. In fact, half a page for each will be sufficient for a reasonable mark, provided your answers are clear and satisfy the requirement.

B.1. My/Our Robot Scenario

Starting with the code given in the Search Exercise 5 notebook create your own robot worker scenario by adding extra items and/or rooms and/or doors. If you wish you can change completely change the types of items and rooms to your own theme. You can also make more technical changes such as adding actions for opening and/or locking doors; or more complex conditions on how objects can be moved. You could potentially even enable different types of goal to be specified.

It is strongly advised that you start by making only very minor changes and then experiment with the search algorithms in order to obtain an initial set of results. Then if you have time you could make the scenario more complex (and of course save a copy of your initial simple solution in case you want to go back to it).

In presenting your answer for this question ***do not include your code****. Instead you should give a clear concise description of the scenario. You should specify the items, the rooms, connecting doors and any other relevant details. You should also briefly describe the possible actions of the robot and the possible goals that could be specified. It is recommended that you include a simple diagram of the room layout, including any significant and/or novel features, so the reader can easily visualise the situation.*

B.2. Heuristic(s)

You should define and describe at least one heuristic that can guide the robot to solving planning problems in your scenario situation.

It is sufficient to implement a very simple heuristic. Considering how something like the `misplaced_tiles` *heuristic for 8-Puzzle might be applied to a Robot Worker situation should give you an idea for a basic heuristic.*

It is possible to devise much more sophisticated heuristics that can enable solution of more complex scenario. Remember that a heuristic should be an estimate of the remaining number of actions from a given state to the nearest goal state. In most cases it is best to have a heuristic that gives a number that is close to but does not exceed the actual number of remaining actions.

B.3. Results

You should present your results in a clear way that brings out the most import and things you have discovered. Probably this would be in the form of one or more tables with a few sentences of explanation.

Note that you do not need to include the code of the search test sequence use to carry out the tests that generated the data that you put in the table. However, it would be a very good idea to devise and maintain a test sequence similar to what you did for **A.1**. *This will enable you to easily rerun and/or modify the test sequence in order to check your results perhaps produce more informative results.*

	a	b	c	d	e	...
?	?	?	?	?	?	...
:	:	:	:	:	:	

B.4. Key Findings

This should be done in a similar way to **A.4***, but of course you need to identify key findings for how the different search algorithms and/or your heuristics behave when used to solve problems in your scenario.*

The most significant observations we made regarding the use of search algorithms applied to our robot worker scenario were as follows:

- A general observation is that ...
- We found that in the case of ...
- ...
- ...