Trabalho prático individual n^o 1

Inteligência Artificial / Introdução à Inteligência Artificial
Ano Lectivo de 2019/2020

25-26 de Outubro de 2019

I Observações importantes

1. This assignment should be submitted via *Moodle* within 27 hours after the publication of this description. The assignment can be submitted after 27 hours, but will be penalized at 5% for each additional hour.
2. Complete the requested functions in module `"tpi1.py"`, provided together with this description. Keep in mind that the language adopted in this course is Python3.
3. Include your name and number and comment or delete non-relevant code (e.g. test cases, print statements); submit only the mentioned module `"tpi1.py"`.
4. You can discuss this assignment with colleagues, but you cannot copy their programs neither in whole nor in part. Limit these discussions to the general understanding of the problem and avoid detailed discussions about implementation.
5. Include a comment with the names and numbers of the colleagues with whom you discussed this assigment. If you turn to other sources, identify those sources as well.
6. All submitted code must be original; although trusting that most students will do this, a plagiarism detection tool will be used. Students involved in plagiarism will have their submissions canceled.
7. The submitted programs will be evaluated taking into account: performance; style; and originality / evidence of independent work. Performance is mainly evaluated concerning correctness and completeness, although efficiency may also be taken into acount. Performance is evaluated through automatic testing. If necessary, the submitted modules will be analyzed by the teacher in order to appropriately credit the student's work.

II Exercices

Together with this description, you can find the module `tree_search`, similar to the one used in practical classes, with small changes and additions.

The module `tpi1_tests` contains the `Cidades` class and the `cidades_portugal` search domain, which you already know. This domain is used for testing. The module already contains several tests. If needed, you can add other test code in this module.

Don't change the `tree_search` module. Module `tpi1` contains `MyTree`, a class derived from `SearchTree`. In the following exercices, you are asked to complete certain methods in this class. All code that you are asked to develop should be integrated in the same module.

1. Create a search method `search2()` similar to the original method `search()` of class `SearchTree`, and add code to assign values to the following attributes.

   In each node of the search tree:

   - `depth` - Depth of the node (the root is at depth 0);
   - `cost` - Acumulated cost along the path from the root to the node;
   - `evalfunc` - The value of the A* evaluation function;
   - `children` - List with all children of the node, or `None`, in case the node wasn't expanded yet.

   In the search tree itself:

   - `solution_cost` - Total cost of the found solution.
   - `solution_length` - Number of state transitions.
   - `total_nodes` - Total number of nodes of the generated tree.

   Example:

```
>>> p1 = SearchProblem(cidades_portugal,'Braga','Agueda')
>>> t1 = MyTree(p1,'breadth')
>>> t1.search2()
['Braga', 'Porto', 'Agueda']
>>> t1.show()
Braga
  Porto
    Agueda
    Aveiro
    Guimaraes
  Guimaraes
    Porto
    Lamego
>>> t1.solution_length, t1.total_nodes, t1.solution_cost
(2, 8, 136)
>>> p2 = SearchProblem(cidades_portugal,'Aveiro','Beja')
>>> t2 = MyTree(p2,'breadth')
>>> t2.search2()
['Aveiro', 'Coimbra', 'Leiria', 'Santarem', 'Lisboa', 'Beja']
>>> t2.solution_length, t2.total_nodes, t2.solution_cost
(5, 615, 476)
```

2. Implement the method `astar_add_to_open()` to support the A* search strategy. This method is already called in the method `add_to_open()` of `SearchTree`.

   Example:

```
>>> p3 = SearchProblem(cidades_portugal,'Braga','Evora')
>>> t3 = MyTree(p3,'astar')
>>> t3.search2()
['Braga', 'Porto', 'Agueda', 'Coimbra', 'Leiria', 'Santarem', 'Evora']
>>> t3.solution_length, t3.total_nodes, t3.solution_cost
(6,149,454)
```

3. Implement the method `effective_branching_factor`, which computes the effective branching factor of a previously generated search tree.

   Following the previous examples:

```
>>> t1.effective_branching_factor()
2.192575225830078
>>> t2.effective_branching_factor()
3.366852559204913
>>> t3.effective_branching_factor()
2.0638977847649507
```

4. Implement the method `update_ancestors` which will propagate the evaluation function updwards in the search tree, as is done in algorithms like RBFS or SMA*. This method must be called in `search2` each time new nodes are added to the open nodes queue. With this update, the evaluation function in each non terminal node should be the minimum value of the evaluation function in any of the descendants of that node.

   Example:

```
>>> t1.show(True)
Braga [134.30696746902277]
  Porto [136.0]
    Agueda [136.0]
    Aveiro [153.38477631085024]
    Guimaraes [193.91393867445294]
  Guimaraes [134.30696746902277]
    Porto [134.30696746902277]
    Lamego [192.07591289387688]
```

5. Implement the method `discard_worse`, which discards some leaf nodes considered less promissing. When the parameter `max_nodes` is given in the constructor of `MyTree`, the `search2` method verifies if the number of nodes in memory is higher than `max_nodes`. If that is the case, `search2` will call `discard_worse` one or more times until the number of nodes in memory becomes smaller or equal to `max_nodes`. The method `discard_worse` finds the non terminal node with highest evaluation function among those terminal nodes in which all children are leaf nodes. Once this node is identified, the children (leaf nodes) are removed from the open nodes queue and the parent goes back to the queue. Obviously, like in SMA*, this modification of the search method attempts to keep memory usage under a certain bound (given by `max_nodes`).

   When the search is concluded, the following attributes should be left in the search tree:

   - terminal_nodes - Number of terminal nodes in memory.
   - non_terminal_nodes - Number of non terminal nodes in memory.

   Example:

```
>>> p45 = SearchProblem(cidades_portugal,'Braga','Faro')
>>> t4 = MyTree(p45,'astar')
>>> t4.search2()
['Braga', 'Porto', 'Agueda', 'Coimbra', 'Leiria', 'Santarem', 'Evora', 'Beja', 'Faro']
>>> t4.total_nodes,t4.non_terminal_nodes,t4.terminal_nodes
(244, 85, 159)

>>> t5 = MyTree(p45,'astar',100)
>>> t5.search2()
['Braga', 'Porto', 'Agueda', 'Coimbra', 'Leiria', 'Santarem', 'Evora', 'Beja', 'Faro']
>>> t5.total_nodes,t5.non_terminal_nodes,t5.terminal_nodes
(257, 37, 62)
```

III Clarification of doubts

This work will be followed through `http://detiuaveiro.slack.com`. The clarification of the main doubts will be placed here.