Trabalho prático individual nº 2

Inteligência Artificial / Introdução à Inteligência Artificial
Ano Lectivo de 2017/2018

24 de Novembro de 2017

I Observações importantes

1. This assignment should be submitted via *Moodle* within 36 hours after the publication of this description. The assignment can be submitted after 36 hours, but will be penalized at 5% for each additional hour.
2. Complete the requested functions in module `"tpi2.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 `"tpi2.py"`.
4. You can discuss this assignment with colleagues, but you cannot copy their programs neither in whole nor in part.
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: correctness and completeness; style; and originality / evidence of independent work. Correction and completeness will be usually evaluated through automatic testing. If necessary, the submitted modules will be analyzed by the teachers in order to appropriately credit their work.

II Exercícios

Together with this description, you can find modules bayes_net and tree_search . They are similar to the ones used in practical classes, but small changes and additions were introduced. In particular, some extra parameters were added to the constructor of class SearchNode in module tree_search , which are stored without any processing. If needed, these parameters can be used to store additional data in the search tree.

The module `tpi2_tests` contains a bayesian network and a search domain for testing. If needed, you can add other test code in this module. Don't change the `bayes_net` and `tree_search` module. Module `tpi2` contains the classes `MyBN` and `MyTree`. In the following exercices, you are asked to complete certain methods of these classes. All code that you may need to develop should me integrated in the same module.

1. Develop a method individual_probabilities () in class `MyBN` to compute the individual probabilities of all variables of the network. The result is given in the form of a dictionary. (Note: Efficiency will be scored in this exercice.)

   Exemplo:

   ```
   >>> bn.individual_probabilities()
   {'s_m': 0.049999999999999815, 'c_s': 0.08119999999999986,
   'car_s': 0.030812271244308048, 'b_v': 0.626838256,
   'm_f': 0.00999999999999999, 's_p': 0.29999999999999993,
   'h': 0.6079227848000001, 'c_c': 0.1381296412247284,
   's_q': 0.227892143602013, 'v_p': 0.11382846093000477,
   's_s': 0.2465302809951367, 'a': 0.0030000000000000118,
   'f_s': 0.0999999999999998, 'd': 0.010000000000000016,
   'b_a': 0.001999999999999999, 's_t': 0.002780000000000004}
   ```

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

   - self . solution_cost - Total cost of the found solution.
   - self . total_nodes - Total number of nodes of the generated tree.

3. In SearchTree, implement methods `uniform_add_to_open()` and `greedy_add_to_open()` to support the uniform cost and greedy search strategies. These methods are already called in method `add_to_open()` of `SearchTree`. Add code in `search2()` to compute all the required information.

   Exemplos:

   ```
   >>> t1 = MyTree(problems[0],'uniform')
   >>> t1.search2()
   ['Beja', 'Evora', 'Santarem', 'Leiria', 'Coimbra', 'Viseu']
   >>> t1.total_nodes,t1.solution_cost
   (76, 469)

   >>> t2 = MyTree(problems[0],'greedy')
   >>> t2.search2()
   ['Beja', 'Evora', 'Santarem', 'Castelo Branco', 'Viseu']
   >>> t2.total_nodes,t2.solution_cost
   (12, 561)
   ```

4. Develop a function `profile_strategy()` that evaluates the performance of a given search strategy on a set of problems. For each problem, the function uses uniform cost search to determine the optimal solution and then computes ratios $g/g_{optimal}$ and $n/n_{optimal}$, where g and n are the cost of the found solution and the number of nodes of the generated tree. Similarly, $g_{optimal}$ and $n_{optimal}$ are the cost and number of nodes obtained with uniform cost search. As result, the function returns a tuple containing the averages of these ratios for the given set of problems.

   Exemplos:

```
>>> profile_strategy('depth',problems)
(1.7539801870611735, 0.555213833382433)

>>> profile_strategy('breadth',problems)
(1.0269893618377826, 0.8812035502911303)

>>> profile_strategy('greedy',problems)
(1.0399111656258069, 0.2566715261287271)
```

5. Develop a method `bidirectional_search()` in `MyTree` that searches both forward from the initial state and backward from the goal state. When the node selected for expansion in one of these searches contains a state already found by the other search, the process ends. (Note: It is enough to check the open nodes of the other search. If this state appears several times in the open nodes, then the relevant node is the one with lowest cost.)

   Exemplo:

```
>>> t3 = MyTree(problems[0],'uniform')
>>> t3.bidirectional_search()
['Beja', 'Evora', 'Santarem', 'Leiria', 'Coimbra', 'Viseu']

>>> t3.total_nodes,t3.solution_cost
(34, 469)
```

III Clarification of doubts

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

1. Mais exemplos para o exercício 5:

```
>>> t4 = MyTree(problems[1],'greedy')
>>> t4.bidirectional_search()
[ 'Braga', 'Porto', 'Agueda', 'Coimbra', 'Leiria', 'Santarem',
  'Lisboa', 'Beja', 'Faro' ]
>>> t4.total_nodes,t4.solution_cost
(22, 739)

>>> t5 = MyTree(problems[6],'depth')
>>> t5.bidirectional_search()
[ 'Guimaraes', 'Braga', 'Porto', 'Agueda', 'Coimbra',
  'Leiria', 'Santarem', 'Lisboa' ]
>>> t5.total_nodes,t5.solution_cost
(19, 439)
```