

# Trabalho prático individual nº 1

## Inteligência Artificial / Introdução à Inteligência Artificial Ano Lectivo de 2021/2022

19-20 de Novembro de 2021

### I Important remarks

1. This assignment should be submitted via *GitHub* within 28 hours after the publication of this description. The assignment can be submitted after 28 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 assignment. 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 account. 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 Exercises

Together with this description, you can find the module `tree_search`, similar to the one initially provided for the practical classes, but with small changes and additions, namely:

- All generated nodes are stored in the list `all_nodes`.
- The position of each node in that list is used as a unique identifier of the node. Therefore, the parent of a node is also identified by the respective integer identifier.
- Terminal and non-terminal nodes are being counted.
- There is a new search strategy "`rand_depth`" which uses a pseudo-random approach to produce different results each time it is invoked with a different seed.

You can also find in this assignment the module `ciudades` containing the `Ciudades(SearchDomain)` class that you already know.

Don't change the `tree_search` and `ciudades` modules.

The module `tpi1_tests` contains several test cases. If needed, you can add other test code in this module.

Module `tpi1` contains the classes `MyNode(SearchTree)`, `MyTree(SearchTree)` and `MyCities(Ciudades)`. In the following exercises, you are asked (or may need) to complete certain methods in these classes. All code that you need to develop should be integrated in the module `tpi1`.

1. Develop a method `maximum_tree_size(depth)` in the `MyCities` class which, given a certain depth, estimates the maximum size of a tree with that depth. In order to produce this estimate, you should use the average number of neighbors per state as an estimate for average branching factor. Assume also that there is no loop prevention (therefore, states can be repeated along a path in the search tree).
2. Implement the `astar_add_to_open(lnewnodes)` method to support the A\* search strategy. Create a method `search2()` in `MyTree` similar to the one in `SearchTree`, but supporting A\* search.
3. Implement the `repeated_random_depth(num_attempts)` method which carries out a certain number of attempts (`num_attempts`) of '`rand_depth`' search and returns the best obtained result, in terms of cost. The best tree should be stored in `self.solution_tree`. The seed for each attempt should be the number of the attempt.
4. Add code in the `search2()` method to maintain in each node a new attribute `eval`. When the node is created, this attribute receives the value of the A\* evaluation function. After expansion of a node, `eval` should always contain the lowest value of the evaluation function in a leaf (terminal node) under that node. In order to achieve this, develop the method `propagate_eval_upwards(node)` which, upon expansion of `node`, updates its `eval` attribute and propagates this update to its ancestors.
5. Add code in the `search2()` method to perform graph search when the `atmostonce` input flag is `True`. As you know, graph search is a kind of tree search that keeps only the best occurrence of each state, i.e., the occurrence where the state has the lowest accumulated cost. Therefore, each state occurs at most once in the generated tree.
6. One way to optimize solutions produced by tree search is to perform successive passes through the solution, replacing parts of the solution with "shortcuts". Develop a method `make_shortcuts()` in the `MyTree` class which, given a solution previously stored in `self.solution`, tries to produce a better (but not necessarily optimal) path. In each iteration, the function walks the path from left to right trying to detect states  $S_i$  and  $S_j$ , where  $j - i > 1$ , for which there is a direct transition from  $S_i$  to  $S_j$ .

For example, in the path

```
[ 'Porto ', 'Aveiro ', 'Figueira ', 'Coimbra ', 'Leiria ' ]
```

we see there is a shortcut (direct connection) from the second state (Aveiro) to the fourth state (Coimbra). Therefore, we can replace the sub-path

```
[ 'Aveiro ', 'Figueira ', 'Coimbra ' ]
```

by the direct connection, obtaining a new path:

```
[ 'Porto ', 'Aveiro ', 'Coimbra ', 'Leiria ' ]
```

The procedure that detects replaceable subpaths is repeated until no subpaths are detected under these conditions. Finally, the function returns the optimized path.

The function should also store the performed optimizations, in the form of a list of tuples  $(S_i, S_j)$ , in `self.used_shortcuts`.

### III Clarification of doubts

This work will be followed through <http://detiuaveiro.slack.com>. The clarification for the main doubts will be added here.

1. To submit the assignment, all we have to do is push our final version to GitHub?

Resposta: Yes, just push the final version.

2. (18:46, 18:59) There was an error in the variable name `self.terminal(s)`: at one point, it was `self.terminal` and at another point was `self.terminals` ; the same happened with `self.non_terminal(s)`. This has been fixed (however, the `terminals` and `non_terminals` that really matter are the ones you will put on `MyTree` and namely in `search2()`). I took the opportunity to correct other details, especially in the assignment description above.

3. Shouldn't `self.terminals` be declared in the constructor?

Resposta: Not needed, because it will not be incremented; only at the end it is calculated.

4. In ex. 2 `MyNode` should be used, but the first node is not ours because of the `super()` call in the `MyTree` constructor. Can we replace this call with our own custom constructor?

Resposta: You can; all you have to ensure is that your code runs with my version of the `tree_search` and `cidates` modules.

5. In ex. 1, is it to return the estimate in integer or decimal?

Resposta: You can round to integer.

6. I am not really understanding what the objective in ex. 3 is.

Resposta: `'rand_depth'` is a strategy defined in `tree_search`; in ex. 3 you just need to run a certain number of attempts of `'rand_depth'` and pick the best result in terms of cost.

7. One student was trying to solve ex. 3 using `search()`; you should all use `search2()`, otherwise you may have compatibility problems between the `tpi1` classes and the `tree_search` classes.

8. Why is `eval` higher than the A\* evaluation function in some nodes?

Resposta: The description says: "After expansion of a node, `eval` should always contain the lowest value of the evaluation function in a leaf (terminal node) under that node." The A\* evaluation function in leaf nodes is usually higher than in non terminal nodes. Check the slides on RBFS; it can help.

9. Regarding `atmostonce`, in the tests it is never passed in the `search2()` argument; I think it is being wrongly passed in the constructor.

Resposta: Yes, now corrected.

10. Is it to use attribute `eval` for tree search?

Resposta: The `eval` is not used in the rest of the assignment; could be used to implement RBFS or SMA\* on top of that, but it would complicate the assignment.

11. As for the `eval` propagated upwards, we just replace in the ascending nodes, or is it to choose from all childs?

Resposta: In propagation, you really have to see the children in each ascending node.

12. I'm not talking about the children of the newly expanded node, but the children of the nodes that are further up.

Resposta: Yes, those too. The `eval` starts by being the A\* evaluation function, and then it gets updated based on the expansion of the descendants.

13. In ex. 4, we cannot compare the `eval` of the child nodes because there is only `node.parent` and not `node.children`

Resposta: The `node.children` does not exist until you create it; nothing prevents you.

14. What is the objective of exercise 5. I'm asking because I only solved until exercise 4 and ran exercise 5 and the results were the same as expected.

Resposta: There was in fact an error in the tests of ex. 5; the flag was being given in the wrong place. This was corrected.

15. There are updates; in `tree_search`, the `show()` method was added; it may be useful for debug; you can modify it as needed. The results file was also updated (the `eval` values were not correct).

16. I believe I've done Ex. 5 correctly, but that a part of the results depends on Ex. 4. If that were the case, would I still get some score for these exercises or not?

Resposta: everything that is correct counts; it's not 0% or 100%.

17. If I may ask, is every line of the tests worth the same, or are they weighted differently?

Resposta: They are weighted differently; even the test cases may be weighted differently; and there will be other testcases. The ones provided are mainly for clarification purposes. After understanding the main goal, you should consider special cases that are part of the problem. The main special cases are also covered in the tests that I will carry out.

18. Can we add arguments in `propagate_eval_upwards()` or other functions.

Resposta: Only optional arguments.

19. One student made the point that line `lnewnodes.append(len(self.all_nodes))` in `SearchTree.search()` is wrong; in fact it should be `lnewnodes.append(len(self.all_nodes)-1)`. Note also that in my `search2()` code, used to generate the results file, this error doesn't exist.
20. In ex. 4, to prevent different people obtaining different results, please use rounded (integer) `eval` values.
21. In `tpi1_tests`, ex. 5, the print of the `eval` value was commented out; you will not be penalized for not keeping `eval` correctly updated in ex. 5.