Trabalho prático individual nº 1

Inteligência Artificial
Ano Lectivo de 2024/2025

25-26 de Outubro de 2024

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.
3. Include your name and number and comment out 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 them 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 `tree_search` module. You can also find attached the modules `cidades`, `strips`, and `blocksworld`, containing the `Cidades(SearchDomain)`, STRIPS(SearchDomain) and other related classes. These modules are similar to the ones initially provided for the practical classes, but with some changes and additions, namely:

- Loop prevention (repeated states along a path) is implemented
- The Cidades(SearchDomain) is given fully implemented.
- The STRIPS(SearchDomain) is given fully implemented, except for the heuristic.

Don't change the tree_search, cidades, strips and blocksworld 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 MyTree(SearchTree), MyNode(SearchNode) and MyBlocksWorld(STRIPS). In the following exercices, you are asked to complete certain methods in these classes. All code that you need to develop should be integrated in the module tpi1.

1. Create a new method search2() in class MyTree, similar to the original search method, and make sure that nodes (class MyNode) have the attributes depth, cost, and heuristic, with the usual meaning, and also action, the action that led from the parent state to the current state. In addition, make sure the search tree (class MyTree) contains the following counters:
    - num_open - number of open nodes, i.e. nodes in the queue;
    - num_solution - number of solution nodes found (one of the techniques below finds several solutions);
    - num_skipped - number of skipped nodes, i.e. nodes removed from the queue but not expanded (one technique skips some nodes).
    - num_closed - number of closed nodes, i.e. nodes removed from the queue and expanded.
2. In MyTree, implement the method astar_add_to_open(lnewnodes), which manages the queue of open nodes according to the values of the A* evaluation function. In the case of a tie in the evaluation function, use the node depth as a second creterion and the alphabetical order of states as a third criterion.
3. The *Informed Depth-First Search* technique consists of normal depth-first search, but the children produced when expanding a node are sorted according to the A* evaluation function. Therefore, the first child to be expanded will be the one with lowest evaluation function. In MyTree, implement the method informeddepth_add_to_open(lnewnodes), to manage the queue when this technique is being used. In the case of a tie in the evaluation function, use the alphabetical order of states as a second criterion.
4. The MyTree constructor has an optional argument improve, which by default is false. In the default case, the search stops when it finds the first solution. When improve=True, the search will continue until the queue is empty, and at that point it returns the best solution found. To prevent exhaustive search, any visited node, with A* evaluation function equal or higher than the cost of the best solution so far, will be skipped (not expanded).
5. Develop the method check_admissible(node) in MyTree, that, given a solution node, checks if all nodes in the path from the root have heuristic values lower or equal to the actual cost of reaching the solution node.

6. Develop the method `get_plan(node)` in `MyTree`, which, given some node, returns the sequence of actions from the root to that node.

7. In class `MyBlocksWorld`, develop the method `heuristic(state,goal)`, which, given a state and the goal, estimates the cost of reaching the goal from that state. This exercice will be scored taking into acount, not only if the solution found with A* is correct and optimal, but also how large/small is the search tree.

III Clarification of doubts

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