Programming Assignment #2: Traditional Search Algorithms

Assigned: 20.11.2023 Due: 15.12.2023

1 Objective

With the help of this assignment, you will learn

- how to formulate a given problem as a search problem,
- how to solve it by applying uninformed and informed search algorithms,
- and how to evaluate the performance of search algorithms.

2 Requirements

- You are supposed to perform PA0 and PA1, before this assignment.
- You must implement your solution in Python and using the SimpleAI package. (Other solutions will not be accepted!)
- **Use your solution to PA1 as starter code.**
- From the SimpleAI package, you will specifically need *models.py* and *traditional.py* codes (located under simpleai/search directory). You are suggested to read their documentation.
- You are also suggested to inspect the sample search problems (hello_world, eight_puzzle, game_walk, missioners) provided in the SimpleAI package. (They are located under samples/search directory.)

3 Specification

In this assignment, you will formulate the NQueens problem as a search problem and solve it using traditional search algorithms: Breadth-First Search (BFS), Uniform Cost Search (UCS), Depth First Search (DFS), Depth Limited Search (DLS), Iterative Deepening Search (IDS), Greedy Search, and A* Search.

For the general definition and state representation of the NQueens problem, refer to PA1. In addition to the definition in PA1, assume that you are allowed to move a single queen in its column at each step and each move is of cost 1. In other words, each action will be in the form of "Move Queen i to row j", where $1 \leq i, j \leq N$.

3.1 Implementation

3.1.1 Additions to the NQueens class in PA1

Use your solution to PA1 and do the following modifications.

- You have to define NQueens class as a child of abstract SearchProblem (see models.py).
- There are 3 main functions (*actions, is_goal, result*) that you have to implement for uninformed search algorithms, and one function (heuristic) for informed search algorithms. (In addition to these functions, you may define internal helper functions if you need.)

- ***__init__(self, N):*** Class constructor that initializes the attributes. Calls parent class constructor with the initial state. The initial state can be randomly generated or initialized by the user as before. You can define the list of all possible actions here (as in missioners example).
- ***actions(self, state):*** Returns the list of possible actions from the state given as parameter.
- ***result(self, state, action):*** Computes and returns the resulting new state when the given action is performed from the given state.
- ***is_goal(self, state):*** Returns whether the state given as parameter is a goal state. Note that a goal state is a state on which the number of attacking pairs is 0. (Hint: You already implemented this in PA1.)
- ***heuristic (self, state):*** Returns the estimated solution cost from the given state to the goal state. You can use the number of attacking pairs in the given state as a heuristic function.

A template class definition for NQueens is given in Figure 1.

```
#use your code for PA1 as starter
class NQueens(SearchProblem):
    def __init__(self, N):
        pass

    def actions(self, state):
        pass

    def result(self, state, action):
        pass

    def is_goal(self, state):
        pass

    def heuristic(self, state):
        pass
```

Figure 1: Template definition of NQueens class.

***A note about the *is_goal()* method:**

If you couldn't implement the code that calculates the number of attacking pairs in a state in PA1 properly, try your best to implement it in this assignment. In case you cannot implement, you can use the predefined goal states, in which the number of attacking pairs is 0, in your goal test. You can find these unique solutions by running the code here (you should sign up for free to run the codes and change N values). In other words, in the is_goal() method, you can just check whether the given state is among this list. Note that, if you do it in this way, some points will be deducted.

3.1.2 Testing

Outside the NQueens class, define a function that calls each algorithm and prints the results and statistics neatly. Sample report output is shown in Figure 2. In the output, you should state the algorithm, resulting state returned as the solution, resulting path found (sequence of actions to reach the goal), cost of the returned solution, viewer statistics, and optionally time taken for the algorithm.

For testing your code, you can use N=4 to 6. You can try N=7 or 8 but for some cases and some algorithms, it may take several hours (depending on your implementation and processor power). You can try the following initial states to make general conclusions about the performance and behaviour of the algorithms: 2323, 4311, 3442, 12345, 13154, 536142, 532512.

```
*** Uninformed Search Algorithms ***


*****************
Graph search? True
BFS
Resulting path:
[(None, '1432'), (('Move queen 1 to row 2', 1, 2), '2432'), (('Move
Resulting state: 2413
Total cost: 3
Viewer stats:
{'max_fringe_size': 120, 'visited_nodes': 87, 'iterations': 87}
Time taken: 0.0208 seconds
Correct solution?: True
*****************
```

Figure 2: Sample output.

3.2 Report

You will also write a report including the below sections. Don't forget to write your name, surname, and student ID in the report.

A – Development Environment

Briefly describe your development environment (Python version, OS, IDE, CPU properties, etc.).

B – Problem Formulation

Shortly explain the problem formulation including your *state specification, initial state*, *possible actions*, *transition model*, *goal test*, and *path cost*. (Do not copy and paste your code, explain in plain English.)

C – Results

Run your implementation with different parameters and inputs (N, initial state, depth limit for DLS, etc.). Put your output results in this section.

D – Discussion

This part is the most important part of the report and homework. Discuss the results of different search algorithms in terms of *completeness*, *optimality*, and *time and space complexity* by referring to your results in Part C and our lectures. Use graph search mainly, but you can also compare graph search and tree search versions of the algorithms. Discuss the effect of depth limit in depth limited search. Don't give purely theoretical information in this section, write your own observations and comments.

4 Submission

- The assignment can be done individually or in teams of a maximum of 3. (Teams can be a mixture of normal and evening education students. You can change your team in different assignments, although it is not suggested.)
- You are free to use any Python development environment that supports Interactive Python Notebook (.ipynb). Some alternatives: Jupyter Notebook, JupyterLab, VS Code, …
- You will submit your report in pdf format and source files (only the code you implemented, not the simpleai library codes).
- Place all your files in a zip archive with the name **AI_PA2_Surname1_Surname2_Surname3.zip and** submit using the Teams assignment module. Single submission for each team is sufficient.
- If you have further questions, you can send me an e-mail or send a message via MS Teams.

4.1 Late Submission Policy

The deadline for homework submissions is **23:59** at the specified date. For each additional day, a **25% cut-off** will be applied.

5 Academic Honesty Policy

You cannot borrow others' ideas or portions of codes, without giving a proper citation. This can be an Internet source, AI chatbot, or your friend. Clearly indicate which part of your code/report/idea is borrowed and state its source. Of course, you cannot get all or most of your work from others. Otherwise, you will be penalized.

Dr. Zeynep ÇİPİLOĞLU YILDIZ