

LINFO1361: Artificial Intelligence

Assignment 3: Solving Problems with Informed Search and Local Search

Amaury Fierens, Eric Piette
April 2024



Guidelines

- This assignment is due on **Thursday 18th April 2024, 18h00**.
- *No delay* will be tolerated.
- *Document* your source code (at least the difficult or more technical parts of your programs). Python docstrings for important classes, methods and functions are also welcome.
- Copying code or answers from other groups (or from the internet) is strictly forbidden. Each source of inspiration must be clearly indicated.
- Source code shall be submitted on the online *INGInious* system. Only programs submitted via this procedure will be graded. No program sent by email will be accepted.
- Respect carefully the *specifications* given for your program (arguments, input/output format, etc.) as the program testing system is *fully automated*.
- The answer to questions must be given by filling in the latex template provided. The final file must be submitted on *gradescope*. No report sent by email will be accepted.
- Nothing must be modified in the template except your answer that you insert in the *answer* environments as well as your names and your group number. The names are provided through the command *students* while the command *group* is used for the group number. The dimensions of *answer* fields *must not* be modified either. For the tables, put your answer between the "&" symbols. Any other changes to the file will *invalidate* your submission.
- To submit on gradescope, go to <https://gradescope.com> and click on the "log in" button. Then choose the "school credentials" option and search for *UCLouvain Username*. Log in with your global username and password. Find the course LINFO1361 and the Assignment 3, then submit your report. Only one member should submit the report and add the second as group member.
- Check this link if you have any trouble with group submission <https://help.gradescope.com/article/m5qz2xsnjy-student-add-group-members>



Deliverables

- The following files are to be submitted:
 - report_A3_group_XX.pdf: Answers to all the questions in a single report following the template given on moodle. Remember, the more concise the answers, the better. This deliverable should be submitted on gradescope as mentioned before.
 - The file `namazon.py` containing your Python 3 implementation of the N- Amazons problem solver. Your program should take the path to the instance

files as only argument. The search strategy that should be enabled by default in your programs is **A* with your best heuristic**. Your program should print the solution to the standard output in the format described further. The file must be encoded in **utf-8**.





Anti plagiat charter

As announced in the class, you'll have to electronically sign an anti plagiat charter. This should be done **individually** in the **INGInious** task entitled *Assignment 3: Anti plagiat charter*. Both students of a team must sign the charter.

1 Search Algorithms and their relations (4 pts)

1.1 A^* versus uniform-cost search

Consider the maze problem given on Figure 1. The goal is to find a path from  to  moving up, down, left or right. The black positions represent walls. This question must be answered by hand and doesn't require any programming.



Questions

1. Give a consistent heuristic for this problem. Prove that it is consistent. Also prove that it is admissible. (1 pt)
2. Show on the left maze the states (board positions) that are visited during an execution of a uniform-cost graph search. We assume that when different states in the fringe have the smallest value, the algorithm chooses the state with the smallest coordinate (i, j) ($(0, 0)$ being the bottom left position, i being the horizontal index and j the vertical one) using a lexicographical order. (1.5 pts)
3. Show on the right maze the board positions visited by A^* graph search with a manhattan distance heuristic (ignoring walls). A state is visited when it is selected in the fringe and expanded. When several states have the smallest path cost, this uniform-cost search visits them in the same lexicographical order as the one used for uniform-cost graph search. (1.5 pts)

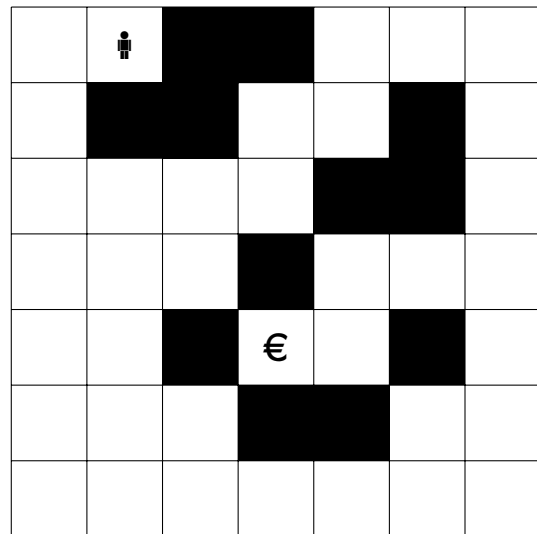
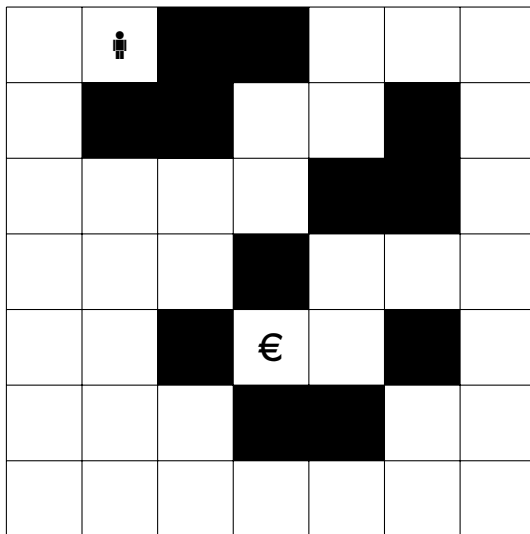


Figure 1

2 N- Amazons problem (8 pts)

The informed problem you will solve for this assignment is the N-Amazons problem. For this problem, the search procedures from aima-python3 will be helpful!

This problem is a variant of the N-Queens problem that is harder to solve than the original. In the N-Amazons problem, the Amazon is a piece that moves like a queen, but have two other moves possible that are extension of the classical knight move:

1. 3X2 move: the empress can go 3 tiles in one direction and 2 tiles in another direction
2. 4x1 move: the empress can go 4 tiles in one direction and 1 tile in another direction

Figure 2 shows a visual representation of all tiles attacked by the amazon piece on a 10x10 chessboard. Each tile attacked is in red. As you may notice, the amazon piece we describe in our problem makes it a lot harder to solve than the N-Queens problem.

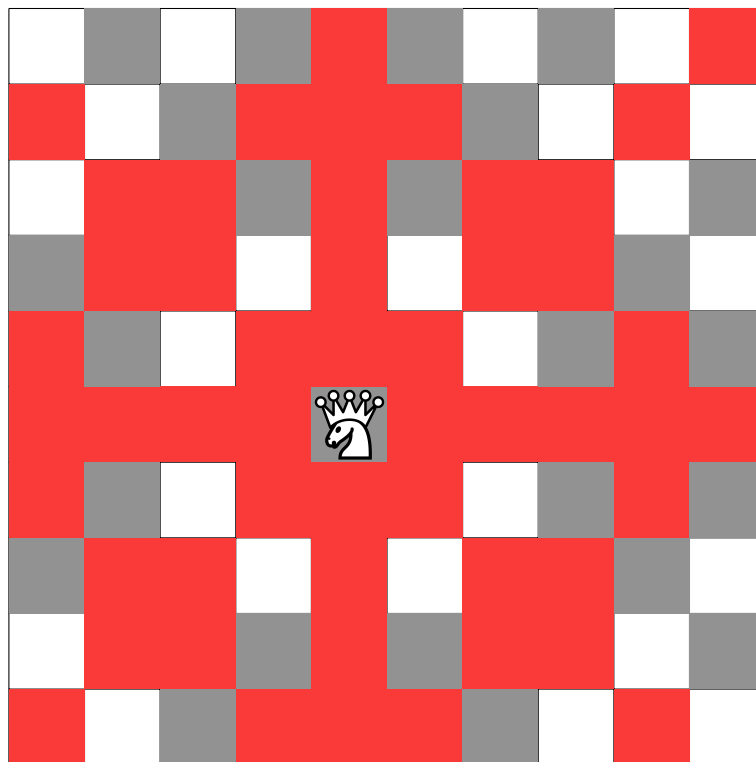


Figure 2

Input and output format

We use ASCII symbols in order to represent a state. Figure 3 shows an example of initial state for any instance. Each instance of test of your program is the integer corresponding to the size of the board and to the number of amazon to place. So an example of command line to call your function would be: `python namazon.py 35`.

```

#####
#####
#####
#####
#####
#####
#####
#####
#####
#####

```

Figure 3: The initial state represented in ASCII for a 10x10 board (N=10)

The different symbols in this problem as follows:

- A marks a tile that is occupied by an amazon
- # marks a tile that is not occupied by an amazon

A solution to this problem is composed of the successive states of the game. In this game, at each step, the goal is to place a new amazon in a new column, starting with the leftmost column and progressing column by column to the right.

The output of the program should be a minimal sequence of every intermediate chessboard, represented in the same way, starting with the initial state and finishing with the goal state, separated by an empty line. At each new state, a new amazon should appear in the leftmost column that does not contain amazon. The goal state is when the rightmost column contains an amazon.

Figure 4 gives an example of a valid solution for N=10. This solution, obtained by executing python3 namazon.py 10, is also optimal because it involves a minimal number of actions (or moves).

initial state	state 1	state 2	state 3
##### ##### ##### ##### ##### ##### ##### ##### ##### #####	##### ##### ##### ##### ##### ##### ##### ##### ##### #####	#A##### ##### ##### ##### ##### ##### ##### ##### ##### #####	#A##### ##### ##### ##### ##### ##### ##### ##### ##### #####
state 4	state 5	state 6	state 7
#A##### ###A##### ##### ##### ##### ##### ##### ##### ##### #####	#A##### ###A##### ##### ##### ##### ##### ##### ##### ##### #####	#A##### ###A##### ##### ##### ##### ##### ##### ##### ##### #####	#A##### ###A##### ##### ##### ##### ##### ##### ##### ##### #####
state 8	state 9	goal state	
#A##### ###A##### ##### ##### ##### ##### ##### ##### ##### #####	#A##### ###A##### ##### ##### ##### ##### ##### ##### ##### #####	#A##### ###A##### ##### ##### ##### ##### ##### ##### ##### #####	

Figure 4: A possible optimal solution output for N=10.

Be careful with the solution output format ! Your solver must respect the exact same format as in Figure 4, except that we put several states per line in this figure for display purpose.

For this, do not modify the output printing code in the example file `namazon.py` and do not print anything in addition.



Questions

1. Model the N-Amazon problem as a search problem; describe: (1 pts)
 - States
 - Initial state
 - Actions / Transition model
 - Goal test
 - Path cost function
2. Give an upper bound on the number of different states for an N-Amazon problem with $N=n$. Justify your answer precisely. (0.5 pt)
3. Give an admissible heuristic for a $N=n$. Prove that it is admissible. What is its complexity ? (1 pts)
4. **Implement** your solver. Extend the *Problem* class and implement the necessary methods and other class(es) if necessary. (0.5 pt)
5. **Experiment**, compare and analyze informed (*astar_graph_search*), uninformed (*breadth_first_graph_search* and *depth_first_graph_search*) graph search of aim-python3 on $N = [10, 11, 12, 13, 20, 25, 30]$. (3 pts for the whole question)

Report in a table the time and the number of explored nodes and the number of steps to reach the solution.

Are the number of explored nodes always smaller with *astar_graph_search*? What about the computation time? Why?

When no solution can be found by a strategy in a reasonable time (say 3 min), indicate the reason (time-out and/or exceeded the memory).
6. **Submit** your program on INGIgnious, using the A^* algorithm with your best heuristic(s). Your file must be named *namazon.py*. Your program should be able to, given an integer as argument, return the correct output. Your program must print to the standard output a solution to the N's given in argument for the N-Amazon problem, satisfying the described output format. (2 pts)

3 Local Search: Sudoku Problem (8 pts)

The local search problem you will solve is the Sudoku problem. While this problem is well-known and described, it is a non-trivial problem. The problem is, given an initial grid of 9×9 where some tiles are filled with a digit (1-9), to fill all the remaining tiles according to three rules:

1. Columns: Each column must contain all digits from 1 to 9 exactly once.
2. Rows: Each row must contain all digits from 1 to 9 exactly once.
3. Subgrids: There are 9 subgrids inside a sudoku grid. Each of these subgrids must contain all digits from 1 to 9 exactly once.

To illustrate, Figure 5 is an example of instance of sudoku unresolved: To formulate this problem in term of local search, the idea is to, by modifying tiles that aren't fixed (those that do not have fixed value at the beginning), fulfilling the whole grid while respecting the 3 rules of the sudoku.

	2		5		1		9	
8			2		3			6
	3			6			7	
		1				6		
5	4						1	9
		2				7		
	9			3			8	
2			8		4			7
	1		9		7		6	

Figure 5: Instance i02

3.1 Objective function

As the objective of the Sudoku is to place correctly each elements, the number of elements still to be placed correctly is equivalent to counting the number of conflicts and add it to the number of tiles not yet filled. Formally, the objective function of this local search problem can be written as follow. A conflict is defined as the fact that one rule is broken (i.e. two digits are the same in one column/row/subgrid).

A solution C to the Sudoku problem is evaluated by counting the number of conflicts and tiles not filled yet.

$$Value(C) = |E_C| \quad (1)$$

where

$$E_C = E_{NotFilled} \cup E_{Conflicts} \quad (2)$$

$$E_{NotFilled} = \{x \mid x \notin 1 - 9\} \quad (3)$$

$$E_{Conflicts} = \{(x, y) \mid x = y, (x, y) \in ForbiddenLocations, (x, y) \in \{1 - 9, 1 - 9\}\} \quad (4)$$

$$ForbiddenLocation = SameColumn \cup SameRow \cup SameSubGrid \quad (5)$$

3.2 Neighborhood

In local search, we have to build at each decision time a neighborhood in which a solution is picked depending on a specific criterion. Here, a simple criterion can be to generate, for all the tiles that are not fixed in the initial state, the list of all locations in the grid that could change and the values that could be put instead of the ones currently in place. Figure 6 is an example of such way to generate neighbors.

Other possibilities of generation are possible and you are welcome to try more optimized ones.

3	2	5	5	8	1	3	9	8
8	5	7	2	9	3	1	4	6
1	3	9	4	6	8	2	7	5
3	7	1	3	8	5	6	2	4
5	4	3	7	2	6	8	1	9
6	8	2	1	4	9	7	5	3
7	9	4	5	3	5	5	8	1
2	6	5	8	1	4	9	3	7
3	1	8	9	5	7	4	6	1

3	2	5	5	8	1	3	9	8
8	5	7	2	9	3	1	4	6
1	3	9	4	6	8	2	7	5
3	7	1	3	9	5	6	2	4
5	4	3	7	2	6	8	1	9
6	8	2	1	4	9	7	5	3
7	9	4	5	3	5	5	8	1
2	6	5	8	1	4	9	3	7
3	1	8	9	5	7	4	6	1

Figure 6: Neighbor generation: The sudoku on the left is shown during execution. All blue digits are the values that are not fixed and can be move by the algorithm, while black digits are those fixed at initial state. The sudoku on the right is the example of a neighbor to the left sudoku, where the 8 from the tiles on the left sudoku has been changed to the 9 in red.

3.3 Input and output formats

The input format describing the different instances on which you will have to test your program is shown on Figure 7. It is a simple text file containing 9 rows of 9 digits. If the digit is 0, then the value is not fixed and can be move by the algorithm. If the value is between 1 and 9, then the value is fixed and cannot be move by the algorithm.

```
020501090
800203006
030060070
001000600
540000019
002000700
090030080
200804007
010907060
```

Figure 7: Instance i02

The output format must be the same as the input format, followed by a blank line, then the inscription "Value(C): ..." filled with the number, then a blank line again. Figure 8 shows the output format.

```
426571398
857293146
139468275
971385624
543726819
682149753
794632581
265814937
318957462
```

Value(C): 0

Figure 8: Example with the instance i02 solved

3.4 Diversification vs Intensification

The two key principles of Local Search are intensification and diversification. Intensification is targeted at enhancing the current solution and diversification tries to avoid the search from falling into local optima. Good local search algorithms have a tradeoff between these two principles. For this part of the assignment, you will have to experiment this tradeoff.

3.5 Simulated Annealing with geometrical decrease

Here in particular, to explore the tradeoff between Diversification and Intensification, you will have to build a Simulated Annealing algorithm with a geometrical decrease. As a reminder, Simulated Annealing is a probabilistic optimization technique inspired by the annealing process in metallurgy. It involves iteratively exploring a solution space, accepting moves that improve the solution with a certain probability, and gradually decreasing the probability of accepting worse moves as the algorithm progresses, allowing for a more refined search for the optimal solution.



Questions

1. Formulate the Sudoku problem as a Local Search problem (problem, cost function, feasible solutions, optimal solutions). (2 pts)
2. You are given a template on Moodle: `sudoku.py`. Implement your own simulated annealing algorithm and your own `objective_score` function. Your program will be evaluated in on 15 instances (during 3 minutes each) of which 5 are hidden. We expect you to solve (get the optimal solution) at least 12 out of the 15. (6 pt)