First assignment - Artificial Intelligence (v1.0)
2025-2026

Logic & Optimization Group (LOG)
Universitat de Lleida
eduard.torres@udl.cat

The goal of this assignment is to evaluate the student's ability to model well-defined problems, implement search algorithms and evaluate their performance.

In the virtual campus, you will find the `assignment1.zip` file, which contains the following files and folders:

```
assignment1
├── algorithms/
├── problems/
│   ├── layouts/
│   │   └── ...
│   ├── kiwis_and_dogs.py
│   ├── nqueens.py
│   └── pacman.py
├── hlogedu_search-0.2.0-cp311-cp311-linux_x86_64.whl
└── docs.pdf
```

This assignment is based on the `hlogedu-search` framework. You can check the official documentation here: `http://ulog.udl.cat/static/doc/hlogedu-search/index.html`.

It can be done in groups of **at most 2 students**.

Notice that the total score of this assignment is **25 points**.

For all the executions, you can set a timeout of 300s using this command: `timeout 300 <cmd>`.

1. Modelling well-defined problems (10/25 points)

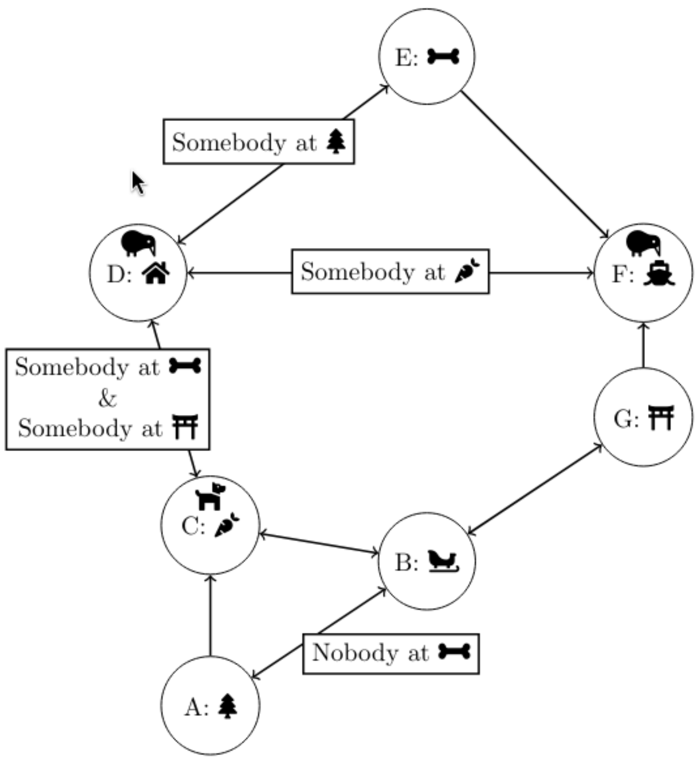
In this section, you are required to model the following problems using the `hlogedu-search` framework.

1.1. The *kiwis and dogs* problem (5/25 points)

Consider a directed graph $G = \{V, E\}$, where V is the set of vertices and E the set of edges. Among all the vertices in V we can find two special vertices: v_{bone}, which corresponds to a bone and v_{tree}, which corresponds to a tree.

We have n dogs and m kiwis at some of the vertices of the graph. At each turn, we can move one dog or one kiwi to another vertex following the edges of the graph. Additionally, to use certain edges, some conditions must be fulfilled (e.g. there must be someone (or nobody) at some specific vertex, see the example below). There are no restrictions on the number of animals that can be in a vertex at the same time. Each edge has a certain length c and each node is placed at a coordinate (x, y) which can be retrieved through the function $get\_coord(v)$.

The objective of this puzzle is to move all the kiwis to v_{tree} and all the dogs to v_{bone} minimizing the travelled distance.



Costs	
A $\leftrightarrow$ B	3
A $\rightarrow$ C	4
B $\leftrightarrow$ C	1
C $\rightarrow$ D	2
B $\leftrightarrow$ G	5
D $\leftrightarrow$ E	8
D $\leftrightarrow$ F	3
G $\rightarrow$ F	7

Figura 1: An example instance for the *kiwis and dogs* problem.

Your tasks:

1. Implement this problem in the `problems/kiwis_and_dogs.py` file. You have a minimal skeleton already implemented for this problem.
2. Check your implementation by running the different `hlog-*` algorithms. Summarize the obtained results in a table/plot and justify the differences.

NOTE: We do not have the `pygame` backend for this problem. Use the `graphviz` backend instead to visualize the different search trees.

1.2. The *N-Queens iterative repair* problem (5/25 points)

The objective of the N-Queens problem is to place N queens on an $N \times N$ chessboard such that no queen attacks another queen. According to the rules of chess, a queen can move (and therefore attack another piece) along its row, its column, or diagonally.

Figure 2 shows a solution to the *8-Queens* problem, where the lines indicate the squares attacked by the queen on square $d7$ (and to which it could move).

For this problem, we will start from an initial configuration of N queens already placed on the board. The queens **can only move according to the rules of chess**. The goal is to reach a valid configuration of the

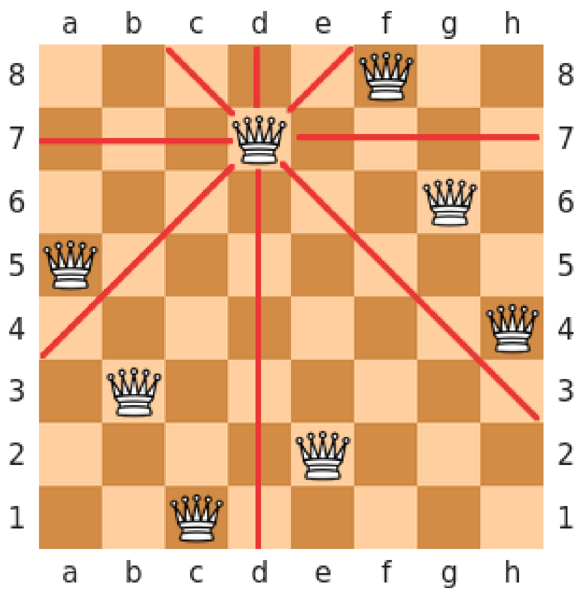


Figura 2: The N-queen Puzzle with $N = 8$ (Source: Wikipedia)

problem while minimizing the total distance travelled by all queens.

Your tasks:

1. Implement this problem in the `problems/nqueens.py` file. You have a minimal skeleton already implemented for this problem.
2. Implement a non-trivial admissible heuristic for this problem. Describe this heuristic in the documentation and justify the admissibility.
3. Check your implementation by running the different UCS and A* algorithms. Summarize the obtained results in a table/plot and justify the differences. Try at least 4 to 10 queens and 5 different seeds for each case (*TIP: Automate this using a script!*).

This problem can be parameterized from the CLI by setting `-pp n_queens=N -pp seed=S`, where `N` and `S` are integers.

Additionally, you can use the `pygame` visualizer to see a cool animation of your solutions!

2. Implementing search algorithms (10/25 points)

Implement the following algorithms:

1. Tree IDS (5/25 points)
2. Tree and Graph A* (5/25 points)

You can create the `ids.py` and `astar.py` files inside the `algorithms/` folder.

IMPORTANT:

- You cannot use any of the `hlog-*` algorithms for this part.

- You must generate the successors in lexicographical order (e.g. using the `sorted` function).
- You must keep track of the number of expanded nodes, and set the `expanded_order` and `location` attributes in the generated nodes, as explained in the `hlogedu-search` documentation.
- Make sure that the number of expanded nodes and solution cost is consistent with the algorithm that you are implementing.
- You can compare your results with the algorithms implemented in the tool. Use several benchmarks to validate your solution!

3. Evaluating the algorithms' performance in the Pacman problem (5/25 points)

Let's focus on the *Pacman* problem, which you already have implemented in the `problems/pacman.py` file.

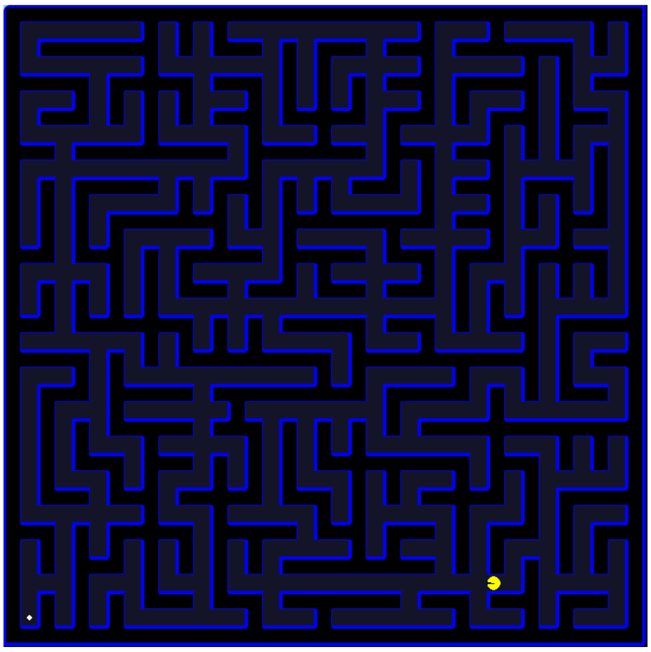


Figura 3: The *Pacman* `pygame` visualization in the *bigMaze* layout.

Our objective is to evaluate the performance of the different algorithms, both those in the framework and those that you have implemented.

To do so, you have a set of layouts inside the `problems/layouts` and `problems/layouts/wc3`. The first ones are small enough to be visualized using the `pygame` outputter. The other ones are maps extracted from Warcraft 3, and are pretty huge to be visualized properly. For these, it is better to use the `none` visualizer and just analyze the returned stats.

Your tasks:

1. Implement the `Manhattan` and `Euclidean` heuristics.
2. Execute the BFS, UCS and A* versions implemented in the framework, in their graph and tree versions, over **all** the provided layouts.
3. Execute your implemented algorithms. For the A*, try all the heuristics that you implemented.
4. Provide a table/graph comparing all these results. Explain the differences briefly.
5. *TIP: Automate this using a script!*

3.1. Delivery

The delivery must contain:

- All the **modified or added** source code files.
- Assignment report (max 6 pages).

All the required content must be delivered in a compressed file with the name `${Surname1}${Name1}_${Surname2}${Name2}_a1.zip` Remember to also include your names in the report.

```
${Surname1}${Name1}_${Surname2}${Name2}_a1.zip
├── algorithms/
│   ├── ids.py
│   └── astar.py
├── problems/
│   ├── kiwis_and_dogs.py
│   └── nqueens.py
└── report.pdf
```

If the automatic tool cannot run your code (be it delivery or implementation), that part will be graded 0.

4. *Changelog*

- v1.0: Base version.