Artificial Intelligence for Robotics II

Assignment 01

Report

Group H

Muhammad Ali Haider Dar
5046263@studenti.unige.it
MSc Robotics Engineering
University of Genoa, Italy

Table of Contents

Introduction

The assignment focused on the operational usage of an AI planner and tested the understanding of planning models. It was inspired by a few robotic coffee shops functional in Japan, which used robots as waiters and baristas. The tasks reimagined a similar scenario involving a robot waiter, a robot barista, customers, and four tables. It attempted to efficiently model the scenario in such a way that the robots plan and perform their tasks effectively as the number of customers increases.

The details pertaining to the planning domain and planning problems, as mentioned in the assignment document, are as follows:

Planning Domain

Coffee Shop Layout

The shop layout is illustrated in Figure 1 and elaborated as follows:

- The coffee shop has the bar counter on the very top, and 4 tables for customers.
- Each table is 1-meter apart from any other (assuming Euclidean geometry does not apply here, so tables 1 and 4 are also 1 meter from each other).
- The bar is 2 meters away from tables 1 and 2.
- Table 3 is the only table of 2 square metres, all the others are of 1 square metre.

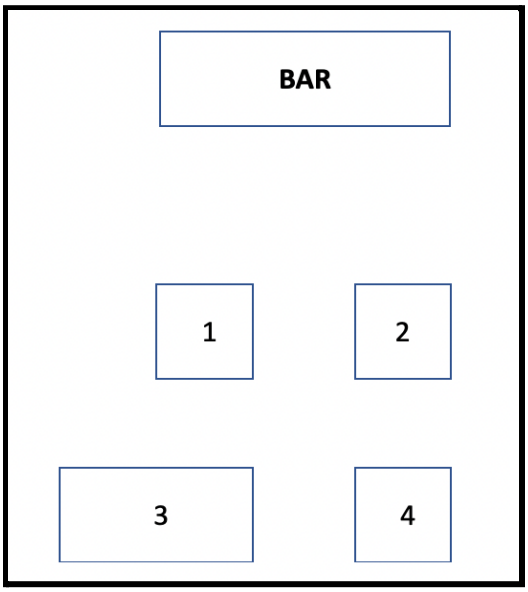

Figure 1. Coffee shop layout

Orders

In the coffee shop, there are two robots: one barista and one waiter. The barista is in charge of preparing drinks. Orders are placed by customers by using a dedicated interface at the entrance

of the shop. This does not need to be modelled and it can be assumed that all the orders, along with the respective customers and the place they sit at, are known at the starting of the planning problem.

Drink Preparation

Cold drinks are faster to prepare than warm drinks. It takes the barista robot 3 time units to prepare a cold drink, and 5 time units to prepare a warm drink.

Serving Customers

Once ready, the drinks are put on the bar, where the waiter robot can pick them up. The waiter can grasp a single drink using one of its grippers, and bring it to the table where the corresponding customer is seated. The waiter is not able to grasp one drink per each gripper, but can only bring a drink at a time if it is not using a tray. If it decides to use a tray, then the waiter can carry up to 3 drinks at the same time, but it's moving speed is reduced – for ensuring everything is balanced. When using a tray, the waiter cannot carry any additional drink, besides the 3 on the tray. The tray can be taken from the bar, and must be returned there after use. The waiter is not allowed to leave the tray on a table.

The waiter moves at 2 meters per time unit; 1 meter per time unit if it is using the tray.

Cleaning Tables

Finally, the robot has to clean tables: it takes 2 time units per square meter to clean a table. The robot cannot clean a table while carrying the tray.

Planning Problems

The following problems will be addressed:

Problem 1

There are 2 customers at table 2: they ordered 2 cold drinks. Tables 3 and 4 need to be cleaned.

Problem 2

There are 4 customers at table 3: they ordered 2 cold drinks and 2 warm drinks. Table 1 needs to be cleaned.

Problem 3

There are 2 customers at table 4: they ordered 2 warm drinks. There are also 2 customers at table 1: they ordered 2 warm drinks. Table 3 needs to be cleaned.

Problem 4

There are 2 customers at table 4 and 2 customers at table 1: they all ordered cold drinks. There are also 4 customers at table 3: they all ordered warm drinks. Table 4 needs to be cleaned.

The report will discuss the planner and planning model used along with the output generated by each of the problem files. It will present a performance analysis of the planner with the increase of problem complexity. The steps and terminal commands required to replicate the results will also be discussed.

Methodology and Discussion

Planner

The planning engine used for this project was ENHSP-20. ENHSP, acronym for Expressive Numeric Heuristic Search Planner, is a PDDL automated planning system which reads domain and problem files as input and provides a time-stamped sequence of actions to achieve the goal state. This planning engine builds an incremental graph and does a heuristic search to explore only the nodes which gets the planner closer to the goal. The details for ENHSP can be accessed *here*.

Domain File

In the domain file '`airo2-domain.pddl`', the object types were declared for 'place', 'order', 'waiter', 'Bar', 'Table', and 'Drink'. The predicates in use were also declared with their relevant objects as predicate arguments, followed by functions. After that, the following major events of the given scenario were implemented with actions, processes, and events:

Preparation of Order

The action 'action-start-order' takes Drink as a parameter with the preconditions that the barista is free, the order is not prepared, and the order hasn't started preparing. The effect is that the barista is now not free, the order has started preparing, and the length of order is assigned the initial value 0.
The process 'process-prepare-order' takes Drink as a parameter with the precondition that the order has started preparing. The effect is that the length of order fluent is increased per unit time.
The event 'event-end-order' takes Drink as a parameter with the preconditions that the order has started preparing and that the length of the order fluent has the value either 3 or 5 units of time, depending on whether the order was of a cold or hot drink. The effect is that the start preparing order is set to false again, order is prepared, order is ready, and the barista is free again.

Serving the order

The action ‘serve-table’ takes waiter and Table as parameters with ‘served’ set to false as the precondition. The effect is that ‘served’ and ‘serving’ are set to true.

Order delivery

The action ‘pick-up-order’ takes Bar, waiter and order as parameters with the preconditions that the waiter is at the Bar, is not carrying a tray or any order, and the order is ready. The effect is that the order is ready, the hand is not free anymore and the order is carried.
The action ‘put-down-order’ takes Table, waiter and order as parameters with the preconditions that drink has been ordered from a table, no tray is being carried, the waiter is at that table, order is being carried, the order is being served. The effect is that the order is not being carried anymore, the order has been delivered and the hand is free again.

Waiter move

The action ‘action-start-move’ takes waiter, initial place and goal as parameters with the preconditions that waiter is free and waiter is at the initial place. The effect is that the initial place is now free, the waiter is neither free nor at the initial place, the waiter has started moving towards the goal, and the fluent containing distance from goal is assigned the distance from initial place to the goal.
The process ‘process-moving’ takes the waiter and goal as parameters with the precondition that the waiter starts moving towards the goal. The effect is that the distance from the goal is decreased per unit time depending upon the number of trays carried by the waiter.
The event ‘event-end-move’ takes waiter and goal as parameters with the precondition that the goal is free, the waiter has already started moving towards the goal, and the distance from goal has reached 0. The effect is that the goal is not free anymore, the waiter is not moving, the waiter is at the goal and is now free.

Clean table

The action ‘action-clean-table-start’ takes the goal and waiter as parameters with the precondition that the waiter is free, the table is not clean, the waiter is at the table, and the hand is free. The effect is that the waiter starts cleaning the table, the time to clean is assigned the value according to the length of the table, and the waiter is not free anymore.
The process ‘process-cleaning-table’ takes the goal and waiter as parameters with the precondition that the waiter has started cleaning the table. The effect is that the time to clean the table is decreased by unit time.
The event ‘event-clean-table-end’ takes the goal and waiter as parameters with the precondition that the waiter has started cleaning the table and that the time to clean the table has reached 0. The effect is that the waiter stops cleaning the table, the table is cleaned, and the waiter is free.

Carrying tray

The action ‘pick-2-tray’ takes the Bar, waiter, first order and second order as parameters with the precondition that the waiter is at the Bar, the first and second orders are not the same, the hand

is free, the tray is not taken and is empty, and both orders are ready. The effect is that the both orders are not ready anymore, the hand is not free, the tray is taken and isn't empty, the tray length is assigned value 1, both orders are carried by the waiter, and the waiter is carrying two drinks.
The action 'pick-3-tray' takes the Bar, waiter, and third order as parameters with the precondition that the waiter is carrying two drinks, is at the Bar, and the third order is ready. The effect is that the third order is not ready anymore, the third order is carried by the waiter, the waiter is not carrying two drinks but is carrying three drinks.

Putting down drinks

The action 'put-down-3-drink' takes Table, waiter and third order as parameters with the precondition that the waiter is carrying the three drinks, the waiter is at the table and serving, and the third drink ordered is carried by the waiter to the table. The effect is that the waiter is not carrying the third drink anymore, order is delivered, and the waiter is now carrying two drinks instead of three.
The action 'put-down-2-drink' takes Table, waiter and second order as parameters with the precondition that the waiter is carrying the two drinks, the waiter is at the table and serving, and the second drink ordered is carried by the waiter to the table. The effect is that the waiter is not carrying the second drink anymore, order is delivered, and the waiter is now carrying one drink instead of two.
The action 'put-down-1-drink' takes Table, waiter and first order as parameters with the precondition that the waiter is carrying one drink, the waiter is at the table and serving, and the drink ordered is carried by the waiter to the table. The effect is that the waiter is not carrying the drink anymore, order is delivered, the waiter is not carrying any drink and the tray is empty.

Putting down tray

The action 'drop-tray' takes the Bar and waiter as parameters with the precondition that the tray is being carried, is empty, and the waiter is at the table. The effect is that the tray is not being carried anymore and the hand is free.

Problem Files

The discussion of problems 1 to 4 mentioned above are discussed as follows:

Problem 1

In the problem file '`airo2-problem-1.pddl`', the objects used are declared in the 'objects' scope with their respective types i.e. object bar with the type Bar; objects table1, table2, table3 and table4 of type Table; objects drink-1 and drink-2 of type Drink; and object w of type Waiter.
The predicates are initialized in the 'init' scope according to problem requirements i.e. fluent-hot for drink-1 and drink-2 is initialized with the value 0 because the customers ordered cold drinks. The distances between the tables and the bar as well as the table sizes are also assigned their respective values according to the coffee shop layout discussed earlier. The number and type of

drinks (hot or cold) are initialized to the tables which ordered them. All tables are initialized as free.
In the goal scope, the following declarations are made: drink-1 and drink-2 are delivered and table3 and table4 are cleaned.

Problem 2

In the problem file '`airo2-problem-2.pddl`', the objects used are declared in the 'objects' scope with their respective types i.e. object bar with the type Bar; objects table1, table2, table3 and table4 of type Table; objects drink-1 to drink-4 of type Drink; and object w of type Waiter.
The predicates are initialized in the 'init' scope according to problem requirements i.e. fluent-hot for drink-1 and drink-2 is initialized with the value 0 because the customers ordered cold drinks whereas fluent-hot for drink-3 and drink-4 is initialized with the value 1 because the customers ordered hot drinks. The distances between the tables and the bar as well as the table sizes are also assigned their respective values according to the coffee shop layout discussed earlier. The number and type of drinks (hot or cold) are initialized to the tables which ordered them. All tables are initialized as free.
In the goal scope, the following declarations are made: drink-1 to drink-4 are delivered and table1 is cleaned.

Problem 3

In the problem file '`airo2-problem-3.pddl`', the objects used are declared in the 'objects' scope with their respective types i.e. object bar with the type Bar; objects table1, table2, table3 and table4 of type Table; objects drink-1 to drink-4 of type Drink; and object w of type Waiter.
The predicates are initialized in the 'init' scope according to problem requirements i.e. fluent-hot for drink-1, drink-2, drink-3 and drink-4 is initialized with the value 1 because the customers ordered hot drinks. The distances between the tables and the bar as well as the table sizes are also assigned their respective values according to the coffee shop layout discussed earlier. The number and type of drinks (hot or cold) are initialized to the tables which ordered them. All tables are initialized as free.
In the goal scope, the following declarations are made: drink-1 to drink-4 are delivered and table3 is cleaned.

Problem 4

In the problem file '`airo2-problem-4.pddl`', the objects used are declared in the 'objects' scope with their respective types i.e. object bar with the type Bar; objects table1, table2, table3 and table4 of type Table; objects drink-1 to drink-8 of type Drink; and object w of type Waiter.
The predicates are initialized in the 'init' scope according to problem requirements i.e. fluent-hot for drink-1, drink-2, drink-3 and drink-4 is initialized with the value 0 because the customers ordered cold drinks whereas fluent-hot for drink-5, drink-6, drink-7 and drink-8 is initialized with the value 1 because the customers ordered hot drinks. The distances between the tables and the bar as well as the table sizes are also assigned their respective values according to the coffee shop layout discussed earlier. The number and type of drinks (hot or cold) are initialized to the tables which ordered them. All tables are initialized as free.

In the goal scope, the following declarations are made: drink-1 to drink-4 are delivered and table4 is cleaned.

Running the Program

1. The ENHSP planner has a Java dependency which can be met by Java version 16.0.1 in Ubuntu 20.04. Run the following lines successively on command terminal to download and install the required jdk:
   ```
   # sudo dpkg -i jdk-16.0.1_linux-x64_bin.deb
   # sudo update-alternatives --install /usr/bin/java java /usr/lib/jvm/jdk-16.0.1/bin/java 1
   # sudo update-alternatives --install /usr/bin/javac javac /usr/lib/jvm/jdk-16.0.1/bin/javac 1
   ```

2. In your workspace, clone the ‘enhsp-20’ branch of ENHSP planner:
   ```
   # git clone -b enhsp-20 https://gitlab.com/enricos83/ENHSP-Public.git
   ```

3. Make the files executable:
   ```
   # chmod +x ./enhsp
   ```

4. Navigate to the assignment folder:
   ```
   # cd <assignment_folder>
   ```

5. Run the planner specifying the domain file `airo2-domain.pddl` and each of the problem files, for example `airo2-problem-1.pddl` file:
   ```
   # /root/Desktop/enhsp/ENHSP-Public/enhsp -o airo2-domain.pddl -f airo2-problem-2.pddl
   ```

6. The terminal output generated using the above command can also be exported in text file format by adding `> <file_name>.txt` after the above command, for example:
   ```
   # /root/Desktop/enhsp/ENHSP-Public/enhsp -o airo2-domain.pddl -f airo2-problem-2.pddl > airo2-problem-2-output.txt
   ```

Results and Conclusion

The output generated against the assigned problems was exported into the respective .txt files and compiled in tabular form, as shown in the table below. The values were plotted to analyze the relation between different parameters and recognize the trend against them. The graphs are illustrated on Figures 2, 3, and 4.

Perimeters	Problem 1	Problem 2	Problem 3	Problem 4
Plan Length	47	67	88	124
Elapsed Time	18.0	26.0	34.0	49.0
Metric (Search)	36.0	52.0	67.0	96.0
Planning Time	1330	3804	117254	70291
Heuristic Time	904	3300	114110	68815
Search Time	1004	3457	116911	69914
Expanded Nodes	530	2568	69091	13794
States Evaluated	1513	7614	245640	53694
Fixed Constraint Violations during Search (zero-crossing)	0	0	0	0
Number of Dead-Ends Detected	0	0	0	0
Number of Duplicates detected	1256	2790	208687	15526

Table 1. Table of Comparison

Figure 2 shows a uniform trend of plan length, elapsed time and metric (search) parameters as the problem complexity increases. This analysis supports the intuition behind the relation between these parameters.

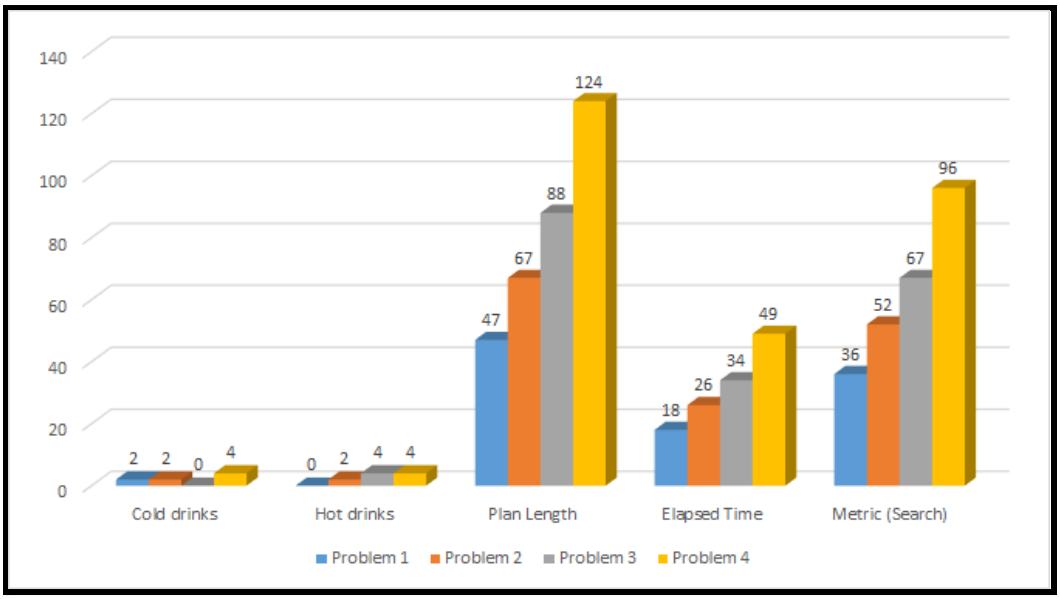

Figure 2. Relation between cold and hot drinks with plan length, elapsed time, and metric (search)

Figure 3 shows a high correlation and a uniform trend between the parameters planning time, heuristic time, and search time. However, it also shows the highest values of these parameters for problem 3, followed by problem 4, problem 2 and problem 1.

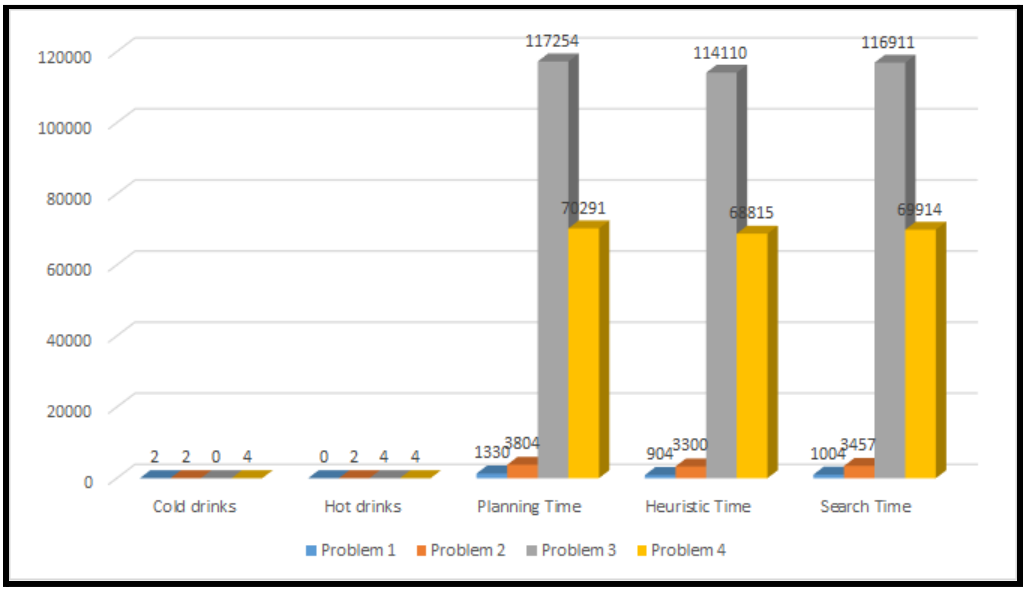

Figure 3. Relation between cold and hot drinks with planning time, heuristic time, and search time

Figure 4 also shows a uniform trend between the expanded nodes, states evaluated and number of duplicates detected. However, in this case, the values of problem 3 for all three parameters far exceed those of problem 4, problem 2 and problem 1, with a very large margin.

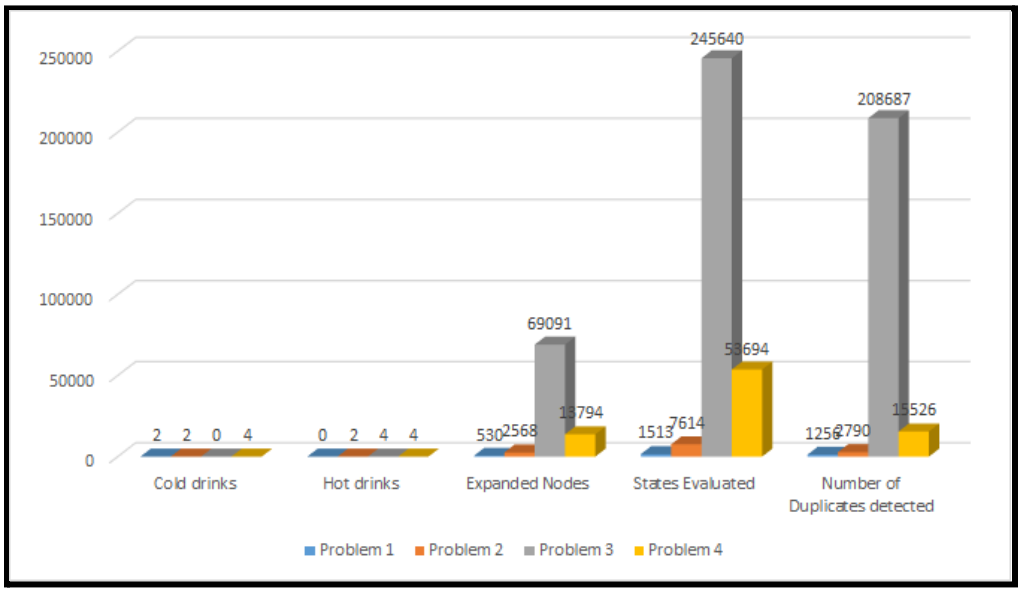

Figure 4. Relation between cold and hot drinks with expanded nodes, states, and duplicates detected

The performance can also be assessed against the peak complexity within the existing problem layout i.e. when the coffee shop reaches the maximum possible number of customers, all tables are fully occupied with a total of 18 customers, and all customers order warm drinks. In future work, the performance of such a problem can be evaluated using a machine of better computation power than the one employed for the assigned problems.