UNIVERSITÀ DEGLI STUDI DI GENOVA

MASTER DEGREE COURSE IN ROBOTICS ENGINEERING

Artificial Intelligence for Robotics II

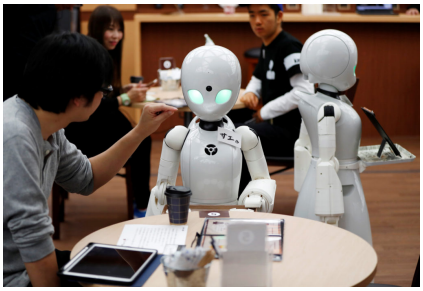
Project Report

Manuel Delucchi, Gianluca Galvagni, Enrico Piacenti, Claudio Demaria, Emanuele Giordano

May, 2023

1 Introduction

As part of the project, our team used the ***Planning Domain Definition Language*** (***PDDL***) to model a *Coffee Shop Scenario* with the aim of using AI to automate the environment. One of the key features of our model was the incorporation of ***durative-actions***, which allowed us to capture actions that have a continuous effect over a period of time, such as preparing drinks or cleaning tables. By creating a detailed model of the coffee shop scenario using *PDDL* and durative-actions, we were able to generate *AI plans* that could automate various tasks within the environment, improving efficiency and reducing the need for manual intervention. The project course gave us a hands-on opportunity to apply our knowledge of *AI planning techniques* and explore the potential of *PDDL* and *durative-actions* in real-world scenarios.



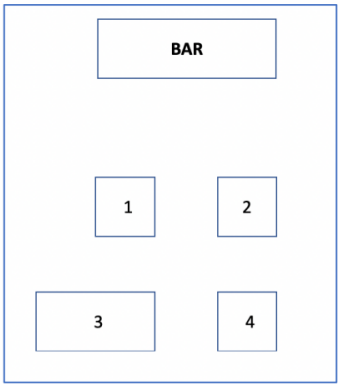
"*OriHime-D, Dawn Avatar Robot Cafè*"

1.1 Coffee Shop Scenario

The ***Coffee Shop Scenario*** consists of:

- ***A Barista Robot***: it must prepare the drinks ordered by the customers;
- ***Two Waiter Robots***: are in charge of serving the customers and cleaning the tables when the customers have left. The waiters can choose to grab a drink with their gripper or use a tray to carry more drinks at once (maximum of 3). When using the tray, the robots move more slowly to improve their balance.

Coffee Shop Layout: There is a bar at the top and 4 tables for customers. Each table is 1 metre away from every other table. The bar is 2 metres from tables 1 and 2. Table 3 is the only 2 square metre table, all the others are 1 square metre. Note that the barista remains stationary at the bar to prepare drinks, while the waiters must move between tables and the bar to perform their duties.



"*Coffee Shop Layout*"

2 Planner

2.1 LPG

LPG, or ***Local Search for Planning Graphs***, is a heuristic search algorithm designed to efficiently solve planning problems. Planning problems involve finding a sequence of actions that can transform an initial state into a desired target state. The planning graph is a representation of the planning problem that captures the dependencies between states and actions.

LPG uses a local search strategy to explore the space of possible solutions. The algorithm starts by generating an initial solution and then iteratively modifies the solution to improve its quality. At each iteration, *LPG* selects a subset of actions to modify and applies a local search operator to improve the solution. This process continues until a satisfactory solution is found or a predefined stopping criterion is met.

LPG handles ***PDDL 2.1*** domains. The domain syntax in *PDDL 2.1* includes two important new features, *durative-actions* and *functions* which are referred to as *numeric fluents*.

2.2 Installing

To download and use the ***LPG Planning Engine*** on your own machine, simply click on the following link: lpg++. After clicking on the hyperlink the download will start automatically.
If you are using *Ubuntu* as your main operating system, you can use the terminal to navigate to the folder where you installed *lpg++* and then make it as executable by typing the command: *chmod +x lpg++*.

Note: Make sure to have *lpg++* file inside the folder together with the *.pddl* files.

2.3 Running

In order to run the program it is necessary to specify the value of three parameters indicating:

1. A file containing a set of *PDDL 2.1* operations: ***Domain***;
2. A file containing the initial state and goals of the problem formalized in *PDDL 2.1*: ***Problem***;
3. A ***running mode***: we used ***speed mode***. In speed mode, *LPG* finds a solution (of any quality) as quickly as possible.

The first two must be specified using:

- *-o <operator file name>*
- *-f <problem file name>*

Finally, you can use type on the terminal → *./lpg++ -o <operator file name> -f <problem file name> -speed*

Where *<operator file name>* is the domain file called *"caffe-bar-domain.pddl"* and the *<problem file name>* are the problem files called *"problemN.pddl"* (with $N = 1, 2, 3, 4$).

3 Model Description

3.1 Drink Preparation

As mentioned earlier, the ***barista robot*** is responsible for preparing the drinks, and customers place their orders using a dedicated interface at the entrance of the shop. At the start of the planning problem, it is assumed that all orders are known, including the customer who placed each order. The barista robot needs 3 time units to prepare a cold drink and 5 time units to prepare a hot one.
Firstly, the barista begin by preparing the drinks at the bar. To model this we used *two durative-actions*:

- "***PrepareIcyDrinks(?d - drink, ?r - barista)***" which has a duration of *t = 3 time units*;
- "***PrepareHotDrinks(?d - drink, ?b - barista)***" which has a duration of *t = 5 time units*.

3.2 Pick-Move-Drop Drink(s)

Since the beginning of the project, we have planned for ***two waiter robots*** to serve customers and clean tables. However, due to their physical limitations, the two robots cannot occupy the same space at the same time, and each table can only be served by one of the waiters. Therefore, each waiter is responsible for handling the orders for a specific table.

Once the drinks have been prepared, they are placed on the bar where they can be picked up by the robot waiter. The waiter robot can pick up a single drink with one of its grippers and transport it to the table where the corresponding customer is seated. The waiter robot cannot pick up more than one drink at a time with its gripper, unless it decides to use a tray. By using a tray, the waiter robot can carry up to three drinks at the same time, but its movement speed is reduced to ensure that everything remains balanced. The waiter robot cannot carry any more drinks than the three on the tray, and the tray must be returned to the bar after use. It is also not allowed to leave the tray on a table. The waiter robot moves at a speed of 2 metres per time unit, but only 1 metre per time unit when using the tray. The *(simple) instantaneous actions* used to model this section are:

- ***PickOne-HOT-Drink(?d - drink, ?w - waiter, ?b - bar)***: the waiter picks up a single hot drink at the bar;
- ***PickOne-ICY-Drink(?d - drink, ?w - waiter, ?b - bar)***: the waiter picks up only a single icy drink at the bar;
- ***PutDownOneDrink(?d - drink, ?w - waiter, ?t - table, ?c - customer)***: the waiter puts down a single drink (regardless of whether it is icy or hot) at the customer's table;
- ***PickTwoDrinks(?d1 - drink, ?d2 - drink, ?w - waiter, ?b - bar)***: the waiter picks up two drinks at the bar with a tray;
- ***PickThreeDrinks(?d1 - drink, ?d2 - drink, ?d3 - drink, ?w - waiter, ?b - bar)***: the waiter picks up three drinks at the bar with a tray;
- ***PutDownOneDrinksWithTray(?d - drink, ?w - waiter, ?t - table, ?c - customer)***: the waiter puts down a single drink at the customer's table while holding the tray;
- ***TrayEmptyDrink(?w - waiter)***: action used to indicate if there are no more drinks on the tray;
- ***PickUpTray(?w - waiter, ?b - bar)***: the waiter picks up the tray at the bar when there is more than one drink on the bar (the velocity of the robot is decreased by 1 unit per metre);
- ***DropTray(?w - waiter, ?b - bar)***: the waiter puts down the tray (the velocity of the robot is increased by 1 unit per metre).

Furthermore, a *durative-action* called ***MoveFromTo(?from - location, ?to - location, ?w - waiter)*** was used for movement. The duration of this action is calculated by dividing the distance between the *?from* and *?to* locations (expressed in metres) by the current velocity of the robot (expressed in metres per time unit).

Note: To satisfy the customer, the drink must be served before it gets too cold. It takes *t = 4 time units* for an hot drink to become cold, and a customer will only accept an hot drink that has been delivered within 4 time units of being put on the bar by the barista. In order to achieve the goal, the waiter must pick up the hot drink immediately after it is prepared by the barista and placed onto the bar. Failing to do so will result in the drink turning cold, leading the customer to reject it.

3.3 We also Serve Food

The Coffee Shop serves delicious biscuits that require no preparation and can be picked up from the bar by the waiter. All customers with a cold drink will also receive a biscuit, but only after they have received their drink. The waiter cannot bring the drink and the biscuit at the same time and must deliver the drink first and then go back to the bar to get a biscuit for the customer. The *(simple) instantaneous actions* used to model this section are:

- ***PickOneBiscuit(?u - biscuit, ?w - waiter, ?b - bar, ?c - customer)***: the waiter picks up only a single biscuit at the bar;
- ***PutDownOneBiscuit(?u - biscuit, ?w - waiter, ?t - table, ?c - customer)***: the waiter puts down a single biscuit at the customer's table;
- ***PickOneBiscuitWithTray(?u - biscuit, ?w - waiter, ?b - bar, ?c - customer)***: the waiter picks up one biscuit at the bar by the means of a tray;
- ***PutDownOneBiscuitWithTray(?u - biscuit, ?w - waiter, ?t - table, ?c - customer)***: the waiter puts down one biscuit at the bar by the means of a tray;
- ***TrayEmptyBiscuit(?w - waiter)***: action used to indicate if there are no more biscuits on the tray.

3.4 Finish your Drink

After receiving the drink, a customer will finish it in 4 time units and will leave. When all the customers at a table have left, the waiter robot can clean it. The robot is required to clean all the tables to complete the problem. It takes 2 time units per square metre to clean a table. The robot cannot clean a table while carrying the tray. For this section, we used *durative-actions*:

- ***ConsumeHotDrink(?d - drink, ?c - customer, ?t - table)***: simulates the durative-action of a consumer consuming a hot drink delivered by the waiter at their table, the action lasts for *t = 4 time units*;
- ***ConsumeIcyDrink(?d - drink, ?c - customer, ?t - table)***: simulates the durative-action of a consumer consuming an icy drink delivered by the waiter at their table, the action lasts for *t = 4 time units*;
- ***ConsumeBiscuit(?u - biscuit, ?c - customer, ?t - table)***: simulates the durative-action of a consumer consuming a biscuit delivered by the waiter at their table, the action lasts for *t = 1 time unit*;
- ***CleanTable(?w - waiter, ?t - table)***: the waiter cleans the table when all customers have left. The duration of this action is calculated by multiplying 2 time units per square metre by the width of the tables (expressed in metres).

Then, we have some *(simple) instantaneous actions*:

- ***LeaveTable(?t - table, ?c - customer)***: simulates the departure of the customer from the table after finishing drinking/eating;
- ***EmptyTable(?t - table)***: action used to indicate if there are no more consumers at the table.

4 Performance Analysis of the Planning Engine

When using *LPG* as a planning engine in *PDDL* programming, the parameters returned by the planner can vary depending on the specific implementation or configuration. However, typically, you can expect to obtain the following information:

- ***Total time***: This parameter denotes the total execution time taken by the planner to generate the plan. It includes both the search time and any additional processing time.

- ***Search time***: This parameter represents the time taken by the planner specifically for the search phase, which involves exploring the planning space, generating candidate plans, and evaluating their quality.
- ***Actions***: The planner returns the sequence of actions comprising the generated plan.
- ***Duration***: This parameter indicates the total execution time required for actions to be completed.
- ***Total number of flips***: This parameter depends on several factors, including the complexity of the planning problem, the number of actions, the size of the state space, and the number of state-action pairs that are visited during the learning process.

  Performing a performance analysis of your plan can be highly beneficial. By examining the parameters returned by the planner, you can gain insights into various aspects of the plan's *quality*, *efficiency*, and *execution*. For example, you can assess the plan's *optimality* based on the plan quality. The total time and search time can help you gauge the efficiency of the planner and evaluate its *scalability*. Analyzing the plan duration and the total number of flips can provide insights into the *plan's execution characteristics* and *potential areas for improvement*.

  Note: When using *fluents*, *actions* and *durative-actions* within the same *PDDL* domain, we need to specify some requirements, specifically *adl*, *fluent* and *durative-action*. Unfortunately, these requirements are only supported by one planner, i.e. *LPG*. We tried to run our domain and problems from other engines, but without success. Therefore, the following tests we performed to evaluate the performance analysis are all based on the use of *LPG* as an algorithm to solve planning problems.

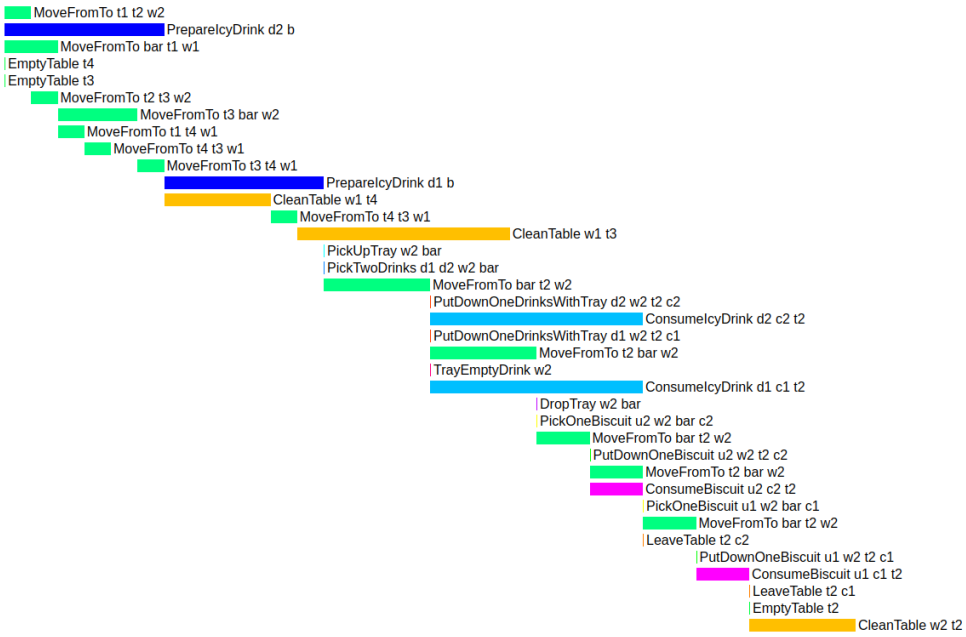
  Note: *100 tests* were performed for each problem.

4.0.1 Problem 1

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

	Actions	***Duration***	***Num Flips***	***Search Time***
Mean	38.05	20.33	542.96	0.05
Standard Devation	4.65	4.641	399.594	0.041

A visual time schedule produced by a test of the *PDDL* first problem is presented below:

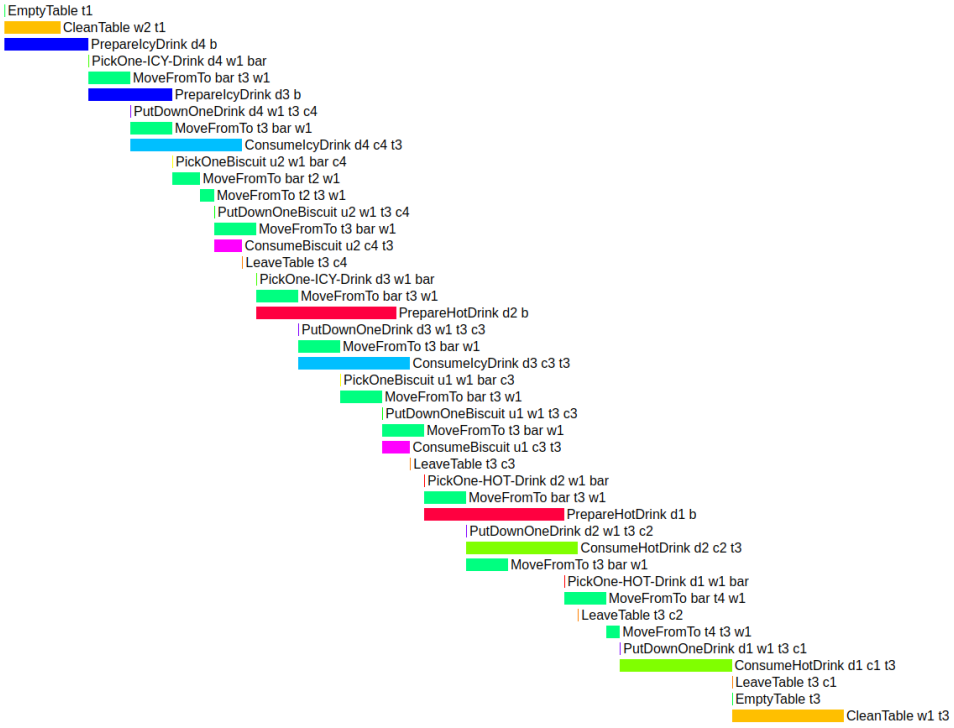


4.0.2 Problem 2

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

	Actions	***Duration***	***Num Flips***	***Search Time***
Mean	50.19	36.995	1123.94	0.149
Standard Devation	6.794	7.551	785.862	0.121

A visual time schedule produced by a test of the *PDDL* second problem is presented below:

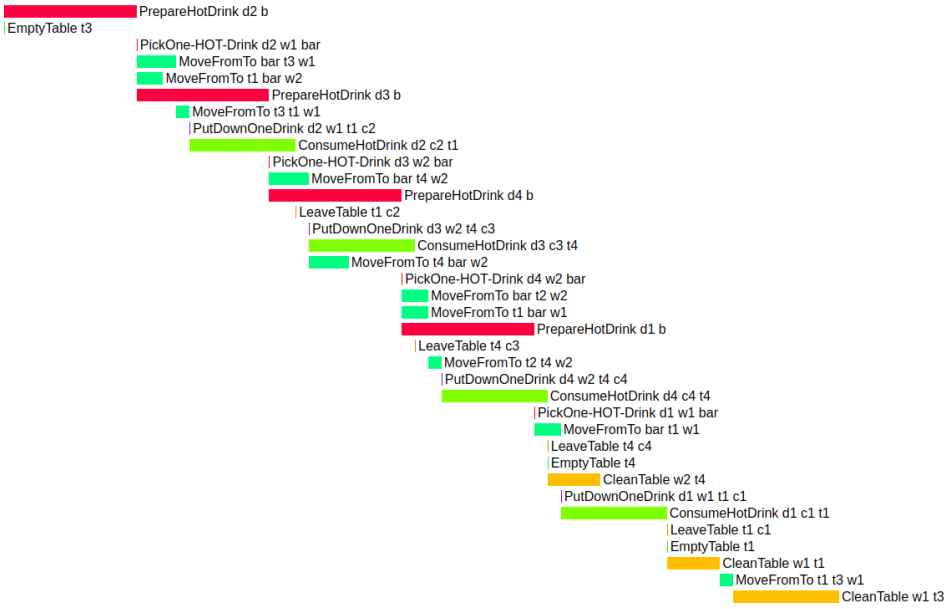


4.0.3 Problem 3

Specifications: 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.

	Actions	***Duration***	***Num Flips***	***Search Time***
Mean	38.36	30.12	473.79	0.04
Standard Devation	2.583	2.692	343.991	0.03

A visual time schedule produced by a test of the *PDDL* third problem is presented below:

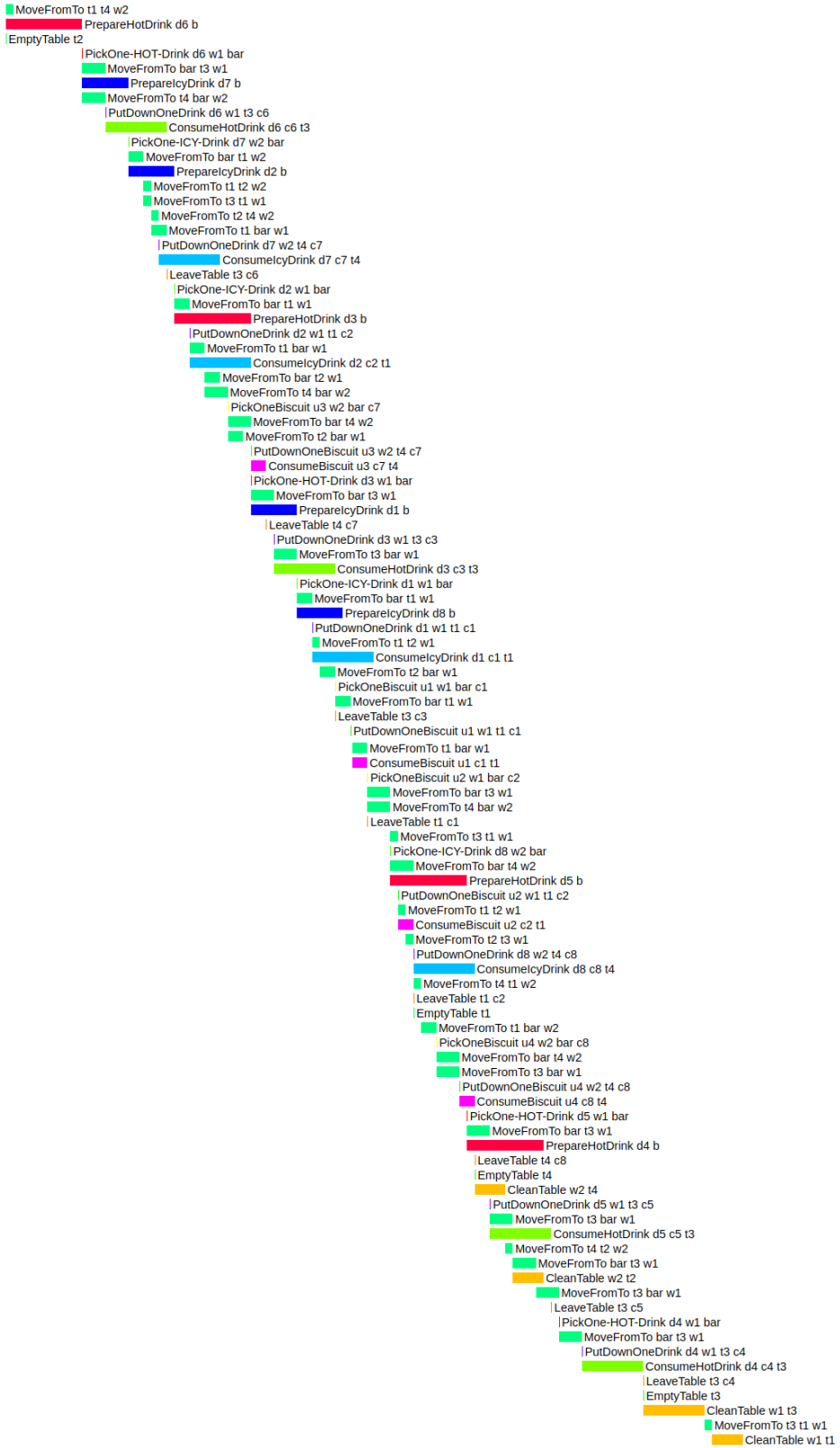


4.0.4 Problem 4

Specifications: 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 2 needs to be cleaned.

	Actions	***Duration***	***Num Flips***	***Search Time***
Mean	97.51	54.645	97711.2	21.995
Standard Devation	6.04	6.467	90739.919	20.404

A visual time schedule produced by a test of the *PDDL* fourth problem is presented below:


MoveFromTo t1 t4 w2
PrepareHotDrink d6 b
EmptyTable t2
PickOne-HOT-Drink d6 w1 bar
MoveFromTo bar t3 w1
PrepareIcyDrink d7 b
MoveFromTo t4 bar w2
PutDownOneDrink d6 w1 t3 c6
ConsumeHotDrink d6 c6 t3
PickOne-ICY-Drink d7 w2 bar
MoveFromTo bar t1 w2
PrepareIcyDrink d2 b
MoveFromTo t1 t2 w2
MoveFromTo t3 t1 w1
MoveFromTo t2 t4 w2
MoveFromTo t1 bar w1
PutDownOneDrink d7 w2 t4 c7
ConsumeIcyDrink d7 c7 t4
LeaveTable t3 c6
PickOne-ICY-Drink d2 w1 bar
MoveFromTo bar t1 w1
PrepareHotDrink d3 b
PutDownOneDrink d2 w1 t1 c2
MoveFromTo t1 bar w1
ConsumeIcyDrink d2 c2 t1
MoveFromTo bar t2 w1
MoveFromTo t4 bar w2
PickOneBiscuit u3 w2 bar c7
MoveFromTo bar t4 w2
MoveFromTo t2 bar w1
PutDownOneBiscuit u3 w2 t4 c7
ConsumeBiscuit u3 c7 t4
PickOne-HOT-Drink d3 w1 bar
MoveFromTo bar t3 w1
PrepareIcyDrink d1 b
LeaveTable t4 c7
PutDownOneDrink d3 w1 t3 c3
MoveFromTo t3 bar w1
ConsumeHotDrink d3 c3 t3
PickOne-ICY-Drink d1 w1 bar
MoveFromTo bar t1 w1
PrepareIcyDrink d8 b
PutDownOneDrink d1 w1 t1 c1
MoveFromTo t1 t2 w1
ConsumeIcyDrink d1 c1 t1
MoveFromTo t2 bar w1
PickOneBiscuit u1 w1 bar c1
MoveFromTo bar t1 w1
LeaveTable t3 c3
PutDownOneBiscuit u1 w1 t1 c1
MoveFromTo t1 bar w1
ConsumeBiscuit u1 c1 t1
PickOneBiscuit u2 w1 bar c2
MoveFromTo bar t3 w1
MoveFromTo t4 bar w2
LeaveTable t1 c1
MoveFromTo t3 t1 w1
PickOne-ICY-Drink d8 w2 bar
MoveFromTo bar t4 w2
PrepareHotDrink d5 b
PutDownOneBiscuit u2 w1 t1 c2
MoveFromTo t1 t2 w1
ConsumeBiscuit u2 c2 t1
MoveFromTo t2 t3 w1
PutDownOneDrink d8 w2 t4 c8
ConsumeIcyDrink d8 c8 t4
MoveFromTo t4 t1 w2
LeaveTable t1 c2
EmptyTable t1
MoveFromTo t1 bar w2
PickOneBiscuit u4 w2 bar c8
MoveFromTo bar t4 w2
MoveFromTo t3 bar w1
PutDownOneBiscuit u4 w2 t4 c8
ConsumeBiscuit u4 c8 t4
PickOne-HOT-Drink d5 w1 bar
MoveFromTo bar t3 w1
PrepareHotDrink d4 b
LeaveTable t4 c8
EmptyTable t4
CleanTable w2 t4
PutDownOneDrink d5 w1 t3 c5
MoveFromTo t3 bar w1
ConsumeHotDrink d5 c5 t3
MoveFromTo t4 t2 w2
MoveFromTo bar t3 w1
CleanTable w2 t2
MoveFromTo t3 bar w1
LeaveTable t3 c5
PickOne-HOT-Drink d4 w1 bar
MoveFromTo bar t3 w1
PutDownOneDrink d4 w1 t3 c4
ConsumeHotDrink d4 c4 t3
LeaveTable t3 c4
EmptyTable t3
CleanTable w1 t3
MoveFromTo t3 t1 w1
CleanTable w1 t1

5 Conclusion

In this section all the data gathered by executing all the problems are analyzed and discussed. To get enough information about the planner's performance on these different problems, we tested each of them 100 times.

5.1 Usage of the tray

For each problem, we recorded the number of times the planner thought it would be more efficient for the waiter to deliver the drinks without using a tray (i.e. with a gripper), the number of times it decided to take the tray to deliver the drinks, and the number of times the tray was taken by both waiters.

	NoTray	***OneTray***	***TwoTrays***	***Tot***
Problem1	82	18	0	100
Problem2	51	49	0	100
Problem3	100	0	0	100
Problem4	52	45	3	100

As seen in the table above, the waiter frequently uses the gripper when preparing hot drinks in order to adhere to the cooling down deadline.
The waiter tends to favor the tray if there are only a few drinks or if every order is for a single table.
The usage of the gripper and tray is not out of balance in any of the other situations, and it cannot be determined which is more effective to utilize.

5.2 Problem 1

The computing cost to complete problem1's goal is quite cheap because the search only takes a few decimal seconds. The performance of the planner in the first problem is pretty stable and produces results that are similar since the ratio between the mean and standard deviation of the data produced is typically low. The number of flips, on the other hand, is often modest since the complexity is low but the standard deviation is big, which means that the planner finds many solutions to this problem with various computational costs.

5.3 Problem 2

The reasoning of the second problem is quite identical to the first, with the exception that two additional warm drinks have been added, and only one table needs to be cleaned as opposed to two.
Although the waiters might choose to use the tray in this situation to transport the cold drinks, the performance analysis of the issue would still be essentially the same: search time would be slow, the number of actions would increase slightly in number, as would the duration and the number of flips.

5.4 Problem 3

The third problem displays an average outcome of actions, duration, and search time that is quite similar to the results of the previous problems, but with the difference in place of the standard deviation.
In all parameters, the latter remains extremely low, demonstrating the remarkable homogeneity of these plans, which have a tendency to converge on one solution or a group of related ones.
This outcome is also a result of the fact that there are only four hot drink orders to be filled, which the waiters choose to take without the tray in order to prevent the latter from becoming cold before being served to the customers.

5.5 Problem 4

Due to the fact that it has a significantly higher number of orders and tasks than the previous assignment, the last problem uses a lot more resources than the others.
In spite of this, we observe that the standard deviation continues to be quite low, with values that are comparable to those of the other problems.
This outcome is advantageous since it assures us of good consistency in the planner's choices.
The usage of the tray is still relatively evenly distributed in terms of resources, and only occasionally do both waiters choose to deliver the beverages on trays. The table clearly shows that there are far more flips made by the planner when executing the task than were previously retrieved values. This is unquestionably due to the fact that there are more orders than previously and that, with one exception, tables are occupied by customers. Additionally, there are many orders, mixed with hot and cold drinks, adding to the complexity. This resulted in a continuous change in the output value of the Q-function, which is the reward, and the planner accomplishes this task with a num filp value 100 times greater than the one in problem 2, that up until now had the gretest value from all the problem up now.
It is also evident that the number of flips and the search duration are connected,as the Q-function (that helps to guide the search towards the goal state and to find a plan that achieves the desired goal) is subjected to numerous "flips" of its values before achieving the goal state and therefore it asses that the execution of the problem as a certain degree of complexity.

6 Further Improvements

The cooling down deadline is not met 35 times, according to an analysis of the 100 tests that were run for problem 2. So it is evident that the cooling down feature doesn't work correctly. This might be a limit of the planner that doesn't support constraint or processes that might be useful for fulfilling this task. So maybe a different planner that supports *PDDL+* or *PDDL3* could be beneficial.