UNIVERSITÀ DEGLI STUDI DI GENOVA

DIBRIS

DEPARTMENT OF COMPUTER SCIENCE AND TECHNOLOGY,
BIOENGINEERING, ROBOTICS AND SYSTEM ENGINEERING

ARTIFICIAL INTELLIGENCE FOR ROBOTICS 2

First Assignment

Modeling a robotic coffee shop

Author:

Giulia Berettieri
Francesca Corrao
Marco Tabita

Student ID:

s4816056
s5471094
s4653859

Professors:

Fulvio Mastrogiovanni
Mauro Vallati

April 30, 2023

Contents

1 Introduction

1.1 Description

The assignment requires to model a robotic coffee shop scenario from the description given. There exist different ways to implement this problem, it is asked to find the most efficient one.
In particular, to run this coffee shop, the barista robot needs to prepare the drinks ordered, which can be cold or warm. The waiter robot is then in charge of serving customers, using its gripper or a tray, and to clean the tables when customers have already left the shop. It is possible to assume that all the orders are known.

1.2 Constraints

More detailed information are then given about the barista, Ernesto:

- uses 3 time units to prepare a cold drink;
- uses 5 time units to prepare a warm drink;

about the waiter, Ambrogio:

- can grasp a single drink using one of its grippers;
- if it is not using a tray can only bring a drink at a time;
- if it decides to use a tray can carry up to 3 drinks at the same time;
- is not allowed to leave the tray on a table;
- moves at 2 meters per time unit;
- if it is using the tray moves 1 meter per time unit;
- takes 2 time units per square meter to clean a table;
- cannot clean a table while carrying the tray;
- when the tray is taken from the bar it must be returned there;

and about the bar itself, it is given the planimetry of the room and the dimension of the table here reported:

- each table is 1 meter apart from any other;
- the Bar is 2 meters away from tables 1 and 2;
- Table 3 is the only table of 2 square meters;
- all the others are of 1 square meter.

1.3 Problems

At the end 4 different problems of increasing difficulty must be solved:

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

2 Base Model

2.1 Initialization

The first thing to do is to understand how the requirements need to be translated into the right implementation. This is done by understanding which type of objects need to be modeled, what are the predicates needed, which actions and durative-actions the planner can do and what are, on the other hand, the processes and the events outside the planner control.
Another important step is the choice of a planning engine because not all the engines support every PDDL functionality. In this case the engine chosen is ENHSP, which supports PDDL+ and numeric Planning, but does not support durative-actions. However, they can be implemented using a succession of action-process-event.

2.2 PDDL

The first approach based on the specification involved to model 5 types, but after some more in depth reading and some trials 2 types have been deleted. In fact only one instance of them is used and the introduction of some predicates was enough to substitute them and to handle their functionalities. In the end the 3 types left are *waiter*, *location* and *drink* while the tray and the barista have been removed.
Then the focus goes on the actions the robot has to perform; in this first step the temporal constraints are ignored and the focus is on the predicates needed to perform the actions. Working in this way let the test be possible using a simple PDDL engine such as planning.domain. In this stage the actions detected and implemented are:

- `take_drink`
- `put_on_tray`
- `take_tray`
- `leave_tray`
- `serve_drink_g`
- `serve_drink_t`
- `move`
- `clean`
- `prepare`

This step was very important because the predicates needed in order to execute the different actions have emerged. In particular, the predicates implemented are needed to know the followings:

- in which *location* a *drink* or a *waiter* are (**at_location**);
- if a *location* is the bar (**bar**);
- if a *waiter* is grabbing something or not (**free**);
- which *drink* a *waiter* is grabbing (**on_robot**);
- if the tray is in a *location* or on the *waiter*(**tray_at**);
- if a *drink* is on the tray (**on_tray**);
- if a *location* needs to be cleaned (**table_to_clean**).

2.3 PDDL+

After the simplest implementation of the problem is developed and a first solution is found, the next step is to add the temporal aspect and to use some numeric resources.
Firstly the numeric aspect is taken into account by adding the function **number_drink_on_tray**, which is used to know how many drinks are one the tray. This makes possible to easily check that the *waiter* does not put more than 3 drinks on the tray just by adding a precondition on the action `put_on_tray`. There are other functions that have been added to the model but their use is more related to the temporal aspect, so they will be explicated when used.
Then the focus goes on adding the temporal constraints: this means that some of the actions previously implemented need to be changed and, in order to do so, some more predicates and functions need to be added. Since the engine chosen is ENHSP and it does not support durative-actions, the ideally modeled durative-actions have been implemented using a sequence of action-process-event. This leads the agent to start the process with an action, then the process itself will model the time passing and when the time needed to execute the action reaches the 0 the event is triggered and it ends the process.

The actions modified to introduce the temporal aspect are the following:

- Move: before it was an action, `move`, that instantaneously changed the *location* in which a *waiter* is, now it is a sequence that takes into account the distance that the *waiter* has to travel in order to arrive in a location and the speed of the *waiter*. In particular the structure is:
  - action (`start_move`): sets a function **distance**, representing the distance to travel, to the actual distance between the starting *location* and the end *location* (**distance_location**), and then the end *location* is stored thanks to the predicate **going_to**.
  - process (`move`): each time instant the **distance** from the end *location* is decreased by the **speed** of the *waiter* expressed as distance travel over time.
  - event (`stop_move`): when the value of **distance** of the end *location* becomes 0, or less, the event is triggered and the *waiter* is now at the end *location*. The value of the predicates **at_location** is set according to it.

  Moreover, the predicate **work_at** is introduced, which helps the robot to move only when strictly necessary. In fact, the robot before starting to move needs to be in a location where it has done something, so useless moving are kept out of the planning.
- Clean: while before this was an instantaneous action, now it will take into account how big is the table to clean and how many time instants are needed to clean it. In particular the structure is:
  - action (`start_clean`): when the *waiter* is not grabbing anything and it is in a *location* that has some **square_2_clean**, it can start to clean it by setting to True the predicate **busy**.
  - process (`cleaning`): each time instant the **square_2_clean** of the *location* is decrease by 0.5 square units.
  - event (`stop_clean`) : when **square_2_clean** of the *location* is less then 0 the **busy** precondition is negated.
- Prepare: a *drink* takes some time to be prepared according to the fact that it is warm or cold, this implemented as follows:
  - action (`shake_it`): when a *drink* is **order** and the barista is not **busy_shake**, the *drink* is set to be **prepared**, the state of the barista is set to **busy_shake** and the function **time_2_prep** of the *drink* is set either to 5 or 3.
  - process (`prepare_drink`): each time instant the **time_2_prep** a *drink* is decreased by 1.
  - event (`pour_it`): once the **time_2_prep** of a *drink* becomes 0 the *drink* does not need to be **prepared** anymore. It can be now found at the bar (**at_location**) and the barista is no longer in the **busy_shake** state.

It is important to keep as action only the start, because the planner can only decide when an action starts not when it ends, the end is then triggered automatically that is why it is an event. To avoid an event to be continuously triggered, at least one predicate checked in the precondition must be negated in the effect.
With this base implementation using the ENHSP engine satisfiable plans in reasonable time are found for all the problems.

3 Extra Functionalities

After the basic problems were solved it has been possible to work on the extra functionalities proposed. Notably, all these functionalities has been added progressively to the domain and to the problems in the order given by the text.

3.1 Warm drinks cool down

The first possible extension requires to serve the warm drinks to the customer before they get cold. This means that the *waiter* has 4 times unit to serve the beverage to the customers. The idea occurred in order to satisfy this requirement is to add a `cooling` process which is started by the event `pour_it` that assigns the **time_2_cool** of the *drink* to the default 4 time units, and ended by the event `drink_cold`. In order to start the process the predicate **is_cooling** is used, by being set to True by the event `pour_it`. The process decreases the **time_2_cool** of the *drink*, and the event makes it not warm if the **time_2_cool** is expired. When the *drink* is served, if it is hot, and it must be warm to be served, one of the effect is to end the `cooling` process.

3.2 There are 2 waiters

In order to help the *waiter* Ambrogio, it is now added a second waitress, Gertrude, but few more constraints are introduced:

- only one *waiter* can be at the bar at a given time;
- only one *waiter* can deal with the orders of a given table.

To solve the first issue a new predicate **loc_used** is added, in this way the *waiter* needs the *location* where it wants to go to be free to start moving there. However the bar situation was still a little bit tricky, since Ambrogio was starting to move from the bar, so two new *location* are then introduced as shown in the figure below:

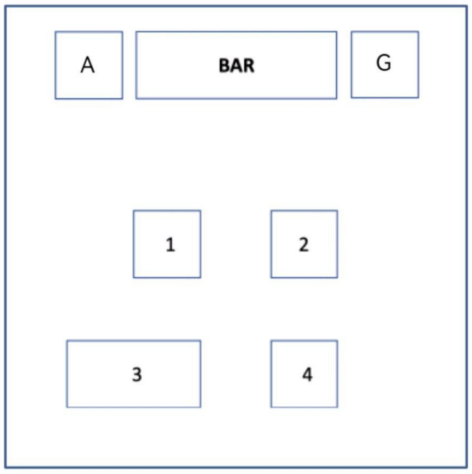


Figure 1: Bar map

These two stations have the same distance from the table as the bar and have an ideally 0-distance from the bar itself. In this way each *waiter* starts from its own station and in order to make all the procedure cleaner in the problem's goal it is specified that the waiters must return to their stations.
In order to overcome the second constraint of this functionality, the solution used involves the static attribution of the single *drink* to the *waiter* in the problem definition. This is because in a real scenario the customers would order the *drinks* with an interface, which would assign a *waiter* to the *drinks* ordered by a table and provide all the informations at the beginning of the problems. This is done taking advantage of the predicate **assigned**.

3.3 Finish your drink

The following extra functionality requires that, after receiving the *drink*, a customer will finish it in 4 time units and will leave after that. When all the customers of a table have left, the waiter robot can clean it.
Aimed at achieving this points it has been added a new process `drinking` which starts as a *drink* is **served** and decreases the **time_2_drink** needed to finish it. In the problems are indicated the total number of drinks ordered from a certain table, which could be easily done in reality too. This is important because two events can then be triggered:

- `end_drinking` which, as soon as **time_2_drink** is expired, decreases the number of drinks ordered on that table. This number is given by the function **drinks_assigned**;
- `table_dirty` which declares that the table needs to be cleaned, turning to False the predicate **dirty**, and assigns **square_2_clean** to its dimension, with the function **dim_table**, when the **drinks_assigned** are 0 and the table has been served.

Moreover the waiters are required to clean all the tables to complete the problem, so it is added in the problems that the tables which are dirty from the beginning and the one which orders drinks must be cleaned.

3.4 We also serve food

The last functionality to implement aims to serve to all the customers who receive a cold *drink* a biscuit which can be picked up by the waiter at the bar counter. A few restrictions are here applied as well:

- The *waiter* can not bring at the same time the drink and its biscuit;
- For the sake of moving biscuits around, the same limitations as for drinks apply.

Aimed to overcame these restrictions the following procedure has been implemented: when the cold *drink* is served the newly introduced predicate **need_biscuit** became True, this triggers two sequential actions `take_biscuit_for` and then `give_biscuit_to`. The biscuit itself is not modeled. The waiter can also use the tray to serve the biscuit, and this thanks to the actions `put_biscuit_on_tray` and `give_biscuit_t`. To take count of the number of the biscuits on the tray is used the function **biscuit_on_tray**. The sequentiality of the actions is given by setting to False and True the predicates **need_biscuit** and **give_biscuit**. Moreover a newly introduced predicate is **biscuit_on_robot** which states if the robot has a biscuit in its gripper.
Lastly to make the procedure of cleaning the tables more realistic and not having a waiter who cleans the table while you are eating, the 4 time instance of the previous extra functionality starts after the biscuit is served.

4 Final Model

4.1 Domain

- TYPES: **waiter** - **location** - **drink**
- PREDICATES:
  - **at_location**: a drink or a waiter is at a location;
  - **bar**: identifies the bar;
  - **free**: a waiter is holding something with its gripper or not;
  - **on_robot**: a drink is grasped by a waiter;
  - **busy**: the state of a waiter: if it is True means that it is cleaning a table;
  - **going_to**: a waiter is going to a location;
  - **work_at**: a waiter worked at a location;
  - **busy_shake**: the state of the barman: if it is True means it is preparing a drink;
  - **warm**: differentiates the warm drinks from the cold ones;
  - **order**: a drink does not exist and must be prepared;
  - **prepared**: a drink is in the preparation process;
  - **on_tray**: drink is on the tray;
  - **tray_at**: the tray location (the robot has it or it is at the bar);
  - **is_cooling**: a warm drink is in cooling state;
  - **assigned**: attribution drink-waiter;
  - **loc_used**: there is a waiter in a location;
  - **served**: a drink is served to a table;
  - **dirty**: a table needs to be cleaned;
  - **need_biscuit**: a drink needs a biscuit;
  - **biscuit_on_robot**: a biscuit is grasped by a waiter;
  - **give_biscuit**: a waiter is serving the biscuit.
- FUNCTIONS:
  - **speed**: waiter's velocity;
  - **distance**: distance to travel to reach a location, it is used for the process *move*;
  - **distance_location**: distance between locations, it is used for mapping the coffee shop;
  - **number_drink_on_tray**: number of drinks on tray;
  - **square_2_clean**: table's surface to clean, it is used by the the process *clean*;
  - **time_2_prep**: time needed to prepare a drink, it is used by the process *shaking*;
  - **time_2_cool**: time remaining before a warm drink becomes cold; it is used by the process *cooling*;
  - **time_2_drink**: time needed to finish a drink, it is used by the process *drinking*;
  - **drinks_assigned**: number of drinks ordered at a table;
  - **dim_table**: table's surface, it is used to set the initial surface to clean;
  - **biscuit_on_tray**: number of biscuits on tray.

- ACTIONS:
  - **put_on_tray** (waiter location drink): corresponds to grasp a drink with the gripper and put it on the tray.

    PRECONDITIONS:
    * The waiter, the drink and the tray must be at the bar;
    * The waiter must be free;
    * The waiter must be assigned to the drink;
    * The tray must have less than 3 drinks.

    EFFECTS:
    * The drink is on the tray and no more at the bar;
    * The number of drinks on tray is incremented by 1;
    * The waiter worked at the bar.

  - **take_drink** (waiter location drink): corresponds to grasp a drink with the gripper.

    PRECONDITIONS:
    * The waiter and the drink must be at the bar;
    * The waiter must be free;
    * The waiter must be assigned to the drink.

    EFFECTS:
    * The waiter is no longer free;
    * The drink is on the waiter and no more at the bar;
    * The waiter worked at the bar.

  - **take_biscuit_for** (waiter location drink): corresponds to grasp the biscuit needed by a served cold drink.

    PRECONDITIONS:
    * The waiter must be at the bar;
    * The waiter must be free;
    * The drink must need a biscuit;
    * The waiter must be assigned to the drink.

    EFFECTS:
    * The waiter is no longer free;
    * The waiter has the biscuit;
    * The waiter worked at the bar;
    * The biscuit is no more needed and must be given to the drink.

  - **give_biscuit_to** (waiter location drink): corresponds to serve the biscuit to its drink.

    PRECONDITIONS:
    * The waiter must be at the table where the drink was previously served;
    * The waiter must have the biscuit;
    * The biscuit must be given to the drink;
    * The waiter must be assigned to the drink.

    EFFECTS:
    * The waiter has no more the biscuit;
    * The waiter worked at the table;
    * The waiter is free;
    * The biscuit has not to be given to the drink anymore.

  - **put_biscuit_on_tray** (waiter location drink): corresponds to grasp a biscuit with the gripper for a drink and put it on the tray.

    PRECONDITIONS:
    * The waiter and the tray must be at the bar;
    * The waiter must be free;
    * The waiter must be assigned to the drink;
    * There must be less than 3 biscuits on the tray;
    * The drink must need a biscuit.

    EFFECTS:
    * The number of biscuit on tray is increased by 1;
    * The waiter worked at the bar.
    * The biscuit is no more needed and must be given to the drink.

– **give_biscuit_t** (waiter location drink): corresponds to serve a biscuit to its drink with the tray.

PRECONDITIONS:
* The waiter must be at the table where the drink was previously served;
* The waiter must have the tray;
* The number of biscuits on tray must be more than 0;
* The biscuit must be given to the drink;
* The waiter must be assigned to the drink.

EFFECTS:
* The waiter worked at the table;
* The number of biscuits on tray is decreased by 1;
* The biscuit has not to be given to the drink anymore.

– **serve_drink_t** (waiter location drink): corresponds to serve a drink at the table with the tray.

PRECONDITIONS:
* The waiter must be at the table;
* The waiter must have the tray;
* The drink must be on tray;
* The waiter must be assigned to the drink.

EFFECTS:
* The number of drinks on tray is decremented by 1;
* The drink is no more on tray but on the table;
* The drink is served;
* The waiter worked at the table;
* The time to drink is assigned to 4;
* If the drink is warm the cooling process stops;
* If the drink is cold it needs a biscuit.

– **serve_drink_g** (waiter location drink): corresponds to serve a drink at the table with the gripper.

PRECONDITIONS:
* The waiter must be at the table;
* The waiter must have the drink;
* The waiter must be assigned to the drink.

EFFECTS:
* The waiter has no longer the drink so it is free;
* The drink is at the table;
* The drink is served;
* The waiter worked at the table;
* The time to drink is assigned to 4;
* If the drink is warm the cooling process stops;
* If the drink is cold it needs a biscuit.

– **leave_tray** (waiter location): corresponds to leave the tray at the bar.

PRECONDITIONS:
* The waiter must be at the bar;
* The waiter must have the tray;
* The number of drinks *or* the number of biscuits on tray must be 0.

EFFECTS:
* The tray is at the bar;
* The waiter has no longer the tray so it is free;
* The waiter's velocity is set to 2;
* The waiter worked at the bar.

– **take_tray** (waiter location): corresponds to take the tray from the bar.

PRECONDITIONS:

* The waiter and the tray must be at the bar;
* The waiter must be free;
* The number of drinks *or* the number of biscuits on tray must be 0.

EFFECTS:

* The waiter has the tray, so it is no longer free;
* The tray is no longer at the bar;
* The waiter worked at the bar;
* The waiter's velocity is set to 1.

– **start_clean** (waiter location): triggers the cleaning process.

PRECONDITIONS:

* The waiter must be at the table;
* The waiter must be free;
* The table must have a surface to clean greater than 0;
* Other waiter related processes must not be running.

EFFECTS:

* The waiter has the cleaning process running;
* The waiter has not worked at the table yet.

– **start_move** (waiter location_from location_to): triggers the moving process.

PRECONDITIONS:

* The waiter must be at location_from;
* The waiter must have worked at location_from;
* The location_to must not be occupied.

EFFECTS:

* The waiter has the moving process running.
* The waiter is no more at location_from, which is now no more occupied;
* The distance to travel is set to the distance between the 2 locations;
* The waiter is going to the destination;
* The waiter finished to work at location_from;
* The location_to is now occupied.

– **shake_it** (drink): triggers the drink's preparing process.

PRECONDITIONS:

* The drink must be ordered;
* Other drinks must not being prepared.

EFFECTS:

* The drink is no more ordered;
* The preparing process is now running;
* If the drink is warm the time to prepare it is set to 5;
* If the drink is cold the time to prepare it is set to 3.

- PROCESSES:
  - **move** (waiter location): represents the motion of the waiter.

    PRECONDITIONS:
    * The waiter must be going to the destination;
    * The distance must be greater than 0.

    EFFECTS:
    * Decrements the distance to travel by the velocity, which is expressed as the distance travelled in 1 time instant.

  - **clean** (waiter location): represents the cleaning of the table.

    PRECONDITIONS:
    * The waiter must have a process running;
    * The waiter must be at the table;
    * The surface to clean must be greater than 0.

    EFFECTS:
    * Decrements the surface to clean by the surface cleaned in 1 time instant, which is 0.5 square meter.

  - **drinking** (drink): represents the consuming of the drink.

    PRECONDITIONS:
    * The drink must be served;
    * The time to drink must be greater than 0;
    * The drink must have its biscuit.

    EFFECTS:
    * Decrements the time to drink the beverage by 1 unit each time instant.

  - **cooling** (drink): represents the cooling of the drink.

    PRECONDITIONS:
    * The drink must be cooling;
    * The time to cool the drink must be greater than 0.

    EFFECTS:
    * Decrements the time to cool the drink by 1 unit each time instant.

  - **prepare_drink** (drink): represents the preparation of the drink.

    PRECONDITIONS:
    * The preparing process is running;
    * The time to prepare the drink must be greater than 0.

    EFFECTS:
    * Decrements the time to prepare the drink of 1 unit each time instant.

- EVENTS:
  - **stop_clean** (waiter location): when occurs the waiter stops to clean the table.

    PRECONDITIONS:
    * The waiter must be at the table;
    * The waiter must have the cleaning process running;
    * The waiter must be free;
    * The table must have a surface to clean lower or equal than 0.

    EFFECTS:
    * The waiter has no longer the cleaning process running;
    * The waiter has worked at the table.

  - **stop_move** (waiter location): when occurs the waiter reaches the destination.

    PRECONDITIONS:
    * The waiter must be going to the destination;
    * The distance to travel must be lower or equal than 0.

    EFFECTS:
    * The waiter is at the destination;
    * The waiter is no longer going to the destination.

  - **end_drinking** (drink location): when occurs the drink is finished.

    PRECONDITIONS:
    * The drink must be served;
    * The time to drink must be lower or equal than 0;
    * The drink must be at the table.

    EFFECTS:
    * The drink has not to be served anymore;
    * The number of drink assigned to the table is decreased by 1;
    * The table is dirty.

  - **table_dirty** (location): when occurs the table needs to be cleaned.

    PRECONDITIONS:
    * The drinks assigned to the table must be lower or equal than 0;
    * The table must be dirty.

    EFFECTS:
    * The surface to clean is set to the dimension of the table;
    * The table is no longer dirty.

  - **drink_cold** (drink): when occurs the drink becomes cold.

    PRECONDITIONS:
    * The drink must be cooling;
    * The time to cool it must be lower or equal than 0.

    EFFECTS:
    * The drink is no longer warm;
    * The cooling process ends.

  - **pour_it** (drink location): when occurs the drink is ready.

    PRECONDITIONS:
    * The preparation process must be running;
    * The drink must be prepared;
    * The time to prepare it must be lower or equal than 0.

    EFFECTS:
    * The preparation process is no longer running;
    * If the drink is warm assigns the time to cool it to 4 and starts the cooling process;
    * The drink is no more prepared and it is at the bar.

4.2 Problems

Accordingly to the model proposed, each time a problem is defined, the same set of predicates and assignment of functions is initialized in order to model the bar properly.

- Ambrogio and Gertrude are the waiters;
- All the tables are defined as locations;
- There are two special locations which are the waiter's stations;
- The waiters are at their stations and they have worked there;
- The waiters are **free**;
- The waiters' velocity is set to 2;
- In order to map the coffee shop, the distance between each pair of locations is set;
- The dimension of tables: 1, 2 and 4 is set to 1;
- The dimension of table 3 is set to 2;
- The bar is defined as a location with the unique **bar** location predicate;
- The tray is at the bar and it has neither biscuits or drinks on it.
- The **time_2_drink** and **time_2_prep** is set to 0 for each drink ordered in the specific problem;
- The **square_2_clean** of the tables not used in each specific problem is set to 0.

Problem 1: There are 2 customers at table 2: they ordered 2 cold drinks. Tables 3 and 4 need to be cleaned.

- INIT:
  - Drinks 1 and 2 are **ordered**;
  - Ambrogio is **assigned** to drinks 1 and 2;
  - The **drinks_assigned** to table 2 are set to 2;
  - The **square_2_clean** of table 3 is assigned to 2;
  - The **square_2_clean** of table 4 is assigned to 1.
- GOAL:
  - Drinks 1 and 2 are at table 2;
  - The **drinks_assigned** to table 2 are 0;
  - Drinks 1 and 2 do not need a biscuit;
  - The **square_2_clean** of tables 2, 3 and 4 is equal to 0;
  - The waiters are not **working**;
  - The waiters are at their stations.

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

- INIT:
  - Drinks 1, 2, 3 and 4 are **ordered**;
  - Drinks 1 and 2 are **warm**;
  - Ambrogio is **assigned** to drinks 1, 2, 3 and 4;
  - The **drinks_assigned** of table 3 is equal to 4;
  - The **square_2_clean** of table 1 is assigned to 1.
- GOAL:
  - Drinks 1, 2, 3 and 4 are at table 3;
  - Drinks 1 and 2 are **warm**;
  - The **drinks_assigned** to table 3 are equal to 0;
  - Drinks 3 and 4 do not need a biscuit;
  - The **square_2_clean** of tables 1 and 3 is equal to 0;
  - The waiters are not **working**;
  - The waiters are at their stations.

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.

- INIT:
  - Drinks 1, 2, 3 and 4 are **ordered**;
  - Drinks 1, 2, 3 and 4 are **warm**;
  - Gertrude is **assigned** to drinks 1 and 2;
  - Ambrogio is **assigned** to drinks 3 and 4;
  - The **drinks_assigned** of tables 1 and 4 are equal to 2;
  - The **square_2_clean** of table 3 is assigned to 2.
- GOAL:
  - Drinks 3 and 4 are at the table 1;
  - Drinks 1 and 2 are at the table 4;
  - Drinks 1, 2, 3 and 4 are **warm**;
  - The **drinks_assigned** to tables 1 and 4 is equal to 0;
  - The **square_2_clean** of tables 1, 3 and 4 is equal to 0;
  - The waiters are not **working**;
  - The waiters are at their stations.

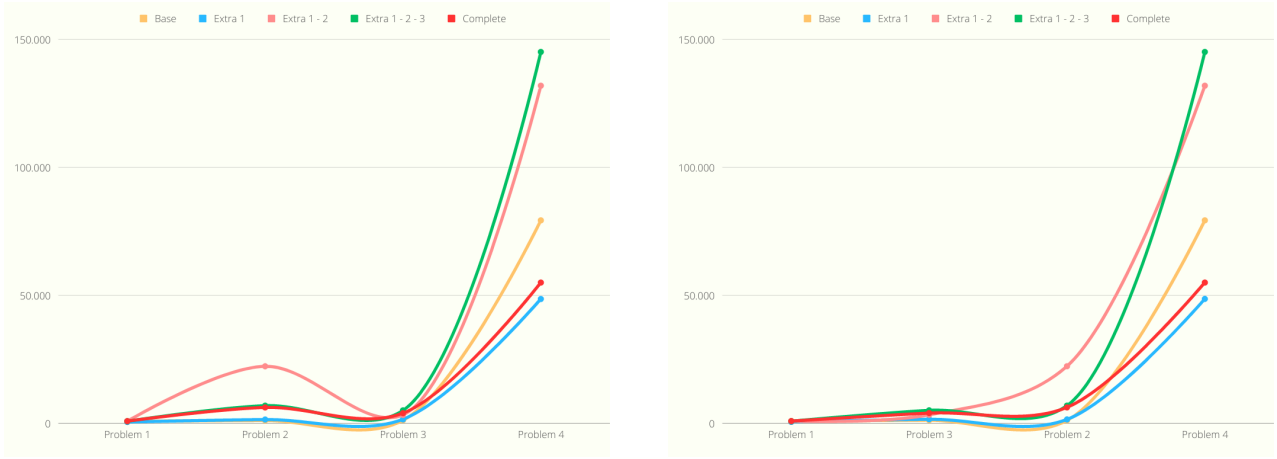
Problem 4: There are 2 customers at table 4 and 2 customers at table 1: they all ordered cold drinks. There are also 4 customers at table 3: they all ordered warm drinks. Table 4 needs to be cleaned.

- INIT:
  - Drinks 1, 2, 3, 4, 5, 6, 7 and 8 are **ordered**;
  - Drinks 1, 2, 3 and 4 are **warm**;
  - Ambrogio is **assigned** to drinks 1, 2, 3 and 4;
  - Gertrude is **assigned** to drinks 5, 6, 7 and 8;
  - The **drinks_assigned** to tables 1 and 4 are equal to 2;
  - The **drinks_assigned** to table 3 are equal to 4;
  - The **square_2_clean** of table 4 is assigned to 1.
- GOAL:
  - Drinks 1, 2, 3 and 4 are at the table 3;
  - Drinks 5 and 6 are at the table 4;
  - Drinks 7 and 8 are at the table 1;
  - Drinks 1, 2, 3 and 4 are **warm**;
  - The **drinks_assigned** to tables 1, 3 and 4 are equal to 0;
  - Drinks 5, 6, 7 and 8 do not need a biscuit;
  - The **square_2_clean** of tables 1, 3 and 4 is equal to 0;
  - The waiters are not **working**;
  - The waiters are at their stations.

In the implementation of this problem, it is possible to notice that the table 4 needs to be cleaned twice. The first time is due to the problem specifications, and the second one because some drinks are served on that table.
Since it is not specified that the table must be clean to serve the drinks there, no restrictions has been applied and the waiter can decide to clean it whenever it wants.
However, the requirement of having a clean table before serving the drinks, could be achieved by adding in the domain an additional precondition in the serve drink actions.

5 Performance Analysis

Once all the outputs have been produced, it has been possible to compare the performance of the engine to solve the different problems.
The analysis concerned in particular the time used by the engine to find a satisfactory plan. It is expected to see an exponential growth of the time while the difficulty of each problem increases, taking as granted that the problems are given in an increasingly order of difficulty, independently from the extra task implemented.
One of the main aspects to take into account when looking at the time used by the engine is how the domain is built. Since each extra functionality has been developed based on the precedent, in the plot should be possible to see how the addition of each task effects the performance of the engine.
The expected result then should include 5 curves with an exponential behaviour.



(a) Ordered behaviour

(b) Increasing behaviour

Figure 2: Planning Time (ms)

In the Figure above is expressed:

- in yellow the trend of the time used to create the base plan for each problem;
- in blue the trend of the time used to create the plan with the 1st extra functionality for each problem;
- in pink the trend of the time used to create the plan with the 1st and the 2nd extra functionality for each problem;
- in green the trend of the time used to create the plan with the 1st, the 2nd and the 3rd extra functionality for each problem;
- in red the trend of the time used to create the plan with the complete domain for each problem;

Focusing on the Figure 2a is possible to see the exponential growth expected. One outstanding result is, on the other hand, the peak reached in the 2nd problem with respect to the third one. While the exponential behaviour is visible in the first Figure, by reordering the problem on the x-axis in the Figure 2b it becomes crystal clear. Differently from what was expected, this result suggests that the second problem presents more issue in the exploring part.
Here is given a possible explanation of this event: despite the number of drinks to serve, which is the same in both cases, in the 2nd problem all the drinks are ordered at the same table, while in the 3rd one are not. This implies that having 2 different waiters to manage the orders is more useful in the 3rd problem due to the restrictions given by the second extra functionality. Another impacting fact is that in the third problem all the drinks are warm, this means that no biscuits needs to be served.

Focusing on the 4th problem, is possible to see that the vertical order of the domains is not what logically expected:

- OBTAINED:
  - extra 1 - 2 - 3;
  - extra 1 - 2;
  - base;
  - complete;
  - extra 1.
- EXPECTED:
  - complete;
  - extra 1 - 2 - 3;
  - extra 1 - 2;
  - extra 1;
  - base.

This is due to the improvement that have been added during the implementation process. In fact in two moments the domain sustained substantial changes, as the ones mentioned in this paper, in order to increase its efficiency: after the base plan has been found and in its final version.
In fact the complete domain has not the highest planning time. It is, instead, remarkably lower than the ones with less functionalities.
The changes that have majorly contributed to lower the planning time are:

- using **busy** only for the `clean` process;
- using **work_at** only for the `move` process;
- do not model the tray as a type;
- reducing the number of predicates and functions;
- avoiding redundant preconditions.

Other possible improvements in efficiency could have been added by ordering the preconditions that allow the actions and the actions themselves, and ideally merging some sequential actions into macro-operators. This has not been done because, after some small trials, the benefits shown were not enough with respect to what should have been implemented.
In conclusion, it is possible to affirm that the final solution proposed is quiet efficient since it solves the most difficult problem in less than 1 minute. Furthermore all the extra functionalities have been implemented and the plans found by the engine are intuitive, functional and similar to the logical implementation that could occur to the human mind.