UNIVERSITA DEGLI STUDI DI GENOVA
ARTIFICIAL INTELLIGENCE FOR ROBOTICS II

First Assignment Report

DIBRIS

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

Authors
Mura Alessio, Pisano Davide, Ruggero Miriam Anna, Terrile Ivan

May 2023

Contents

1 **Introduction** 3
1.1 Description of the model 3
1.2 Choice of the Planning Language 4

2 **Code description** 5
2.1 Domain 5
2.1.1 Types 5
2.1.2 Predicates 6
2.1.3 Functions 7
2.1.4 Actions 7
2.1.5 Process 11
2.1.6 Events 12
2.2 Problem instances 13
2.2.1 Initialization 13
2.2.2 Goal 13

3 **ENHSP different choices of search strategies and heuristics** 14
3.1 Basic functionality Domain-Problem 15
3.2 Optional extensions Domain-Problem 16

4 **Results and Conclusions** 18
4.1 Main Case 18
4.2 Optional Extensions 18
4.3 Delta evolution 19

5 **References** 20

Chapter 1

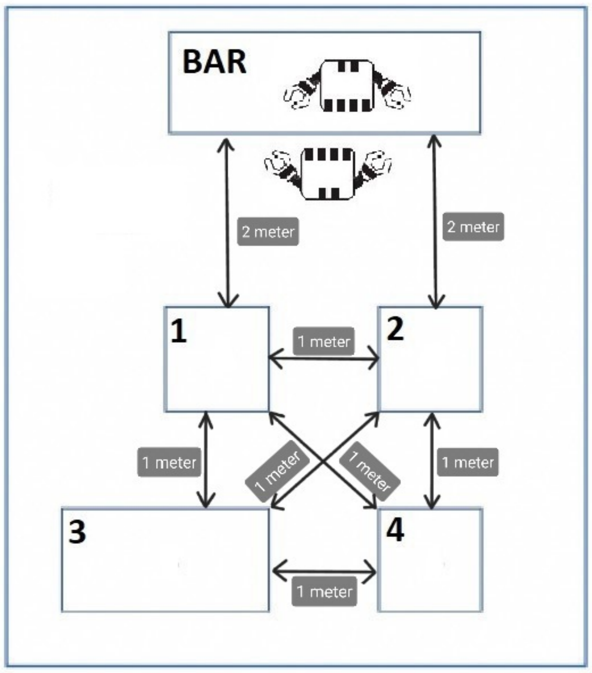
1 Introduction

1.1 Description of the model

This report describes the procedure used to solve the assignment of the Artificial Intelligence for Robotics II course. Our task is to model a Coffee shop scenario in which robots run this business. The final problem has the following structure:

- **Robots**
  There are two robots: one *barista* and one *waiter*. The barista is in charge of preparing drinks at bar counter. Cold drinks are faster to prepare than warm drinks. The waiter robot is then in charge of serving customers, and to clean tables where customers have already left the shop. The waiter robot can decide to grasp a drink with its gripper, or to use a tray to carry more drinks at once. When using the tray, the robot is moving slower to improve balance.

- **Work Environment**
  The coffee shop layout is shown in the figure below. It has the bar counter on the very top, and 4 tables for costumers. Each table is 1-meter apart from any other (let's assume Euclidean geometry does not apply here, so also tables 1 and 4 are 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.



More Details:

All the orders are assumed to be known at the start of our planning problem. It takes the barista robot 3 time units to prepare a cold drink, and 5 time units to prepare a warm drink. Once ready, the drinks are put on the bar, where the waiter can pick them up. The waiter robot 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 its moving speed is reduced in order to ensure everything is balanced. 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. 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.

Optional Extensions:

Moreover, we chose a few optional ways in order to extend our model:

1. **There are 2 waiters**:
   The coffee shop is doing well, so the owner decided to buy a second waiter. This is of course making things a bit more complicated: only one waiter can be at the bar at a given time; further, the owner does not want that a table is served by both the waiters: only one waiter can deal with the orders of a given table.

2. **Finish your drink**:
   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. The waiter robot is required to clean all the tables to complete the problem.

3. **We also serve food**:
   The coffee shop is also serving delicious biscuits. They need no preparation, and can be picked up by the waiter at the bar counter. All the customers with cold drinks will also receive a biscuit, but only after having received their drink. The waiter can not bring at the same time the drink and the biscuit, but must deliver the drink, and then go back to the counter to take a biscuit for the customer. For the sake of moving biscuits around, the same limitations as for drinks apply.

1.2 Choice of the Planning Language

To describe our model, we chose the *PDDL+* description language. This choice is mainly based on three aspects:

- The model must take into account time.
- The need for a mixed discrete-continuous domain, where some variables may change in time, such as the travelled distance of the waiter robot.
- The need to have processes and events in the domain: they describe how the model evolves and how our robots interact with the environment.

In particular the model follows the *action-process-event* schema, where:

- *Actions* are planning decisions: the planner uses them to start processes
- *Procesess* describe how the model evolves in time
- *Events* stop processes when they reach their goal.

Chapter 2

2 Code description

2.1 Domain

In this section, we are going to describe all the predicates, actions, functions, processes, and events that we have implemented. We used actions and events to start and stop processes which are associated with:

- **Robot movements**:
  The waiter robot can move in the environment thanks to a corresponding *action-process-event* schema we created. It can choose to move itself with a drink or a biscuit in its gripper, with a tray or with nothing.
- **Preparing/Finishing drinks**:
  Thanks to another *action-process-event* schema we created, the barista robot can prepare the required drinks: each one with its preparation time depending on the type. Moreover, once the drinks are served on the table, the customers can drink them in a given time unit.
- **Cleaning tables**:
  Finally, the waiter robot is in charge of cleaning the dirty tables and those that were occupied by the customers.

In the following section, we report a brief description of every single type, predicate, function, action, process and event we used.

2.1.1 Types

One important aspect of PDDL+ is the need to specify types for the objects in the domain. Specifying types in the domain is necessary to enable the planner to reason about the relationships between different objects and their properties. In addition, specifying types enables the planner to perform type checking during the planning process, which helps to prevent errors that might occur if objects of the wrong type are used in a plan. This ensures that the resulting plans are valid and executable.

Subtypes	**Types**
drink tray robot	object
biscuit	food
bar table	location
Sub-subtypes	**Subtypes**
cold warm	drink
barista waiter	robot

2.1.2 Predicates

Predicates are a crucial aspect of using PDDL+ to model complex planning problems, in fact are used to define the properties or relationships that hold between objects in the planning domain. They are an important part of the language because they enable the planner to reason about the state of the world and how it changes over time. Predicates are also used to define the preconditions and effects of actions. The preconditions of an action specify the conditions that must hold in order for the action to be executed, while the effects specify how the action changes the state of the world.

Predicate	**Description**
ready ?*d* - *drink*	Predicate to indicate if the drink is ready.
preparing ?*d* - *drink*	Predicate to indicate if the drink is being prepared.
drink-served ?*d* - *drink*	Predicate to indicate if the drink was served.
carrying-drink ?*w* - *waiter* ?*d* - *drink*	Predicate to indicate if the drink is being brought.
at-drink ?*l* - *location* ?*d* - *drink*	Predicate to indicate the location of the drink.
at-drink-tray ?*d* - *drink* ?*t* - *tray*	Predicate to indicate the location of the drink on the tray.
drink-on-tray ?*d* - *drink* ?*t* - *tray*	Predicate to indicate if the drink is on the tray.
drinking ?*d* - *drink*	Predicate to indicate if the client is drinking.
finished ?*d* - *drink*	Predicate to indicate if the client finished the drink.
free-barista ?*b* - *barista*	Predicate to indicate if the barista if free
free-waiter ?*w* - *waiter*	Predicate to indicate if the waiter is free.
at-barista ?*l* - *location*	Predicate to indicate the location of the barista.
at-waiter ?*w* - *waiter* ?*l* - *location*	Predicate to indicate the location of the waiter.
order-of ?*w* - *waiter* ?*l* - *table*	Predicate to indicate if the waiter is taking the order.
at-tray ?*l* - *location*	Predicate to indicate the location of the tray.
carrying-tray ?*w* - *waiter* ?*t* - *tray*	Predicate to indicate if the waiter is carrying the tray.
moving-with-tray ?*w* - *waiter* ?*t* - *tray*	Predicate to indicate if the waiter is moving with the tray.
moving ?*w* - *waiter*	Predicate to indicate if waiter is moving.
connected ?*l1* - *location* ?*l2* - *location*	Predicate to indicate if the locations are connected.
empty ?*from* - *location*	Predicate to indicate if the location is empty.
cleaning ?*l* - *table*	Predicate to indicate if table is being cleaned.
cleaning-waiter ?*w* - *waiter*	Predicate to indicate if the waiter is cleaning.
cleaned ?*l* - *table*	Predicate to indicate if table has been cleaned.
dirty ?*l* - *table*	Predicate to indicate if the table is dirty.
at-biscuit ?*l* - *location* ?*c* - *food*	Predicate to indicate the location of the biscuit.
carrying-biscuit ?*w* - *waiter* ?*c* - *food*	Predicate to indicate if the biscuit is being brought.
biscuit-on-tray ?*c* - *food* ?*t* - *tray*	Predicate to indicate if the biscuit is on the tray.
biscuit-served ?*c* - *food*	Predicate to indicate if the biscuit was served.
together ?*c* - *food* ?*d* - *cold*	Predicate to indicate if the drink and the biscuit are together.
unavailable ?*d* - *drink*	Predicate to indicate if the drink is unavailable.
unavailable-biscuit ?*c* - *food*	Predicate to indicate if the biscuit is unavailable.

2.1.3 Functions

Functions are an important part of using PDDL+ to model planning problems that involve quantitative aspects. They are used to represent numeric values that can be computed based on the state of the world and the actions taken. They are an important part of the language because they enable the planner to reason about quantitative aspects of the planning problem. Functions can be used in the preconditions and effects of actions to represent changes to their values. Finally, they can also be used to define the goal of the planning problem, by specifying a desired value for the function at the end of the plan.

Function	Description
duration-drink ?*d* - *drink*	Function to define the duration of the preparation of drink.
distance ?*l1* - *location* ?*l2* - *location*	Function to define the distance between tow locations.
cleaning-duration ?*l* - *table*	Function to define the duration of cleaning the table.
table-dimension ?*l* - *table*	Function to define the dimension of table.
tray-capacity ?*t* - *tray*	Function to define the capacity of the tray when carrying drinks.
tray-capacity-biscuit ?*t* - *tray*	Function to define the capacity of the tray when carrying biscuits.
real-distance ?*w* - *waiter*	Function to define the real distance that the waiter has to cover.
distance-covered ?*w* - *waiter*	Function to define the distance covered by the waiter.
finishing-drink ?*d* - *drink*	Functions to define the duration of drinking the drink.
counter-client ?*l* - *table*	Function to define the number of clients at the table.

2.1.4 Actions

Actions, In PDDL+, are used to define the basic building blocks of the planning problem, by specifying how the state of the world changes as a result of executing an action. They are an important part of the language because they enable the planner to reason about the effects of different actions and to generate plans that achieve a specified goal in the domain.

(:action prepare-drink)
This action specifies the process of the barista robot preparing a drink at the bar counter. The precondition of the action specifies that the barista robot performing the action must be free and at the bar counter, the drink must not be ready or currently being prepared. The effect of the action is to update the state of the world by setting the barista to not be free and marking the drink as being prepared. Moreover, it activates the process of preparing a new drink: *(:process preparing-drink).*

(:action pick-drink)
This action specifies the process of a waiter picking up a ready drink at the bar counter to deliver to a specified table. The precondition of the action specifies that the drink must be ready to be served, the waiter must be free and located at the bar counter where the drink is located, and the waiter cannot be currently moving. The effect of the action is to update the state of the world by having the waiter carry the drink, removing the drink from the bar counter, and marking the waiter as no longer free. This means that the waiter is now carrying the drink and cannot perform any other actions until the drink is delivered to the customer.

(:action serve-drink)
This action specifies the process of a waiter serving a drink to a customer at a table. The precondition of the action specifies that the waiter must be carrying the drink, located at the customer's table, and not currently moving. Additionally, the waiter cannot be free, meaning that they must be actively serving a customer. The effect of the action is to update the state of the world by having the waiter no longer carry the drink, becoming free to perform other actions, and marking the drink as served. The action also updates the location of the drink to be at the customer's table.

(:action pick-biscuit)
This action specifies the process of a waiter picking up a biscuit at the bar counter to deliver to a specified table. The precondition of the action specifies that the drink must be already served, the waiter must be free and located at the bar counter where the biscuit is located, and the waiter cannot be currently moving. The effect of the action is to update the state of the world by having the waiter carry the biscuit, removing the biscuit from the bar counter, and marking the waiter as no longer free. This means that the waiter is now carrying the biscuit and cannot perform any other actions until the biscuit is delivered to the customer.

(:action serve-biscuit)
This action specifies the process of a waiter serving a biscuit to a customer at a table. The precondition of the action specifies that the waiter must be carrying the biscuit, located at the customer's table, and not currently moving. Additionally, the waiter cannot be free, meaning that they must be actively serving a customer and the drink must be already served at the customer. The effect of the action is to update the state of the world by having the waiter no longer carry the biscuit, becoming free to perform other actions, and marking the biscuit as served. The action also updates the location of the biscuit to be at the customer's table.

(:action start-drinking-cold)
This action specifies the process of a customer starting a cold drink. The precondition of the action specifies that both biscuit and cold drink must be at the client table and already served. The cold drink must be in the condition that the customer is not drinking it, or has finished it. The effect is that the customer starts drinking the drink.

(:action start-drinking-warm)
This action specifies the process of a customer starting a warm drink. The precondition of the action specifies that the drink must be at the client table and already served. The warm drink must be in the condition that the customer is not drinking it, or has finished it. The effect is that the customer starts drinking the drink.

(:action finished-drink)
This action represents the process of finishing drinks. The action's precondition specifies that the drink must be finished and available at the table. Additionally, the drink must not be currently being consumed, there must be at least one client at the table, and the drink must not already be unavailable. The effect of this action is to decrease the counter of the table by one unit and to make the drink unavailable, which means that it cannot be consumed anymore.

(:action can-clean)
The action specifies that a table can be cleaned if it has no clients and has not been cleaned yet. The precondition of the action checks that the counter of clients for the table is equal to zero and that the table has not been cleaned yet. If these conditions are met, then the table is considered dirty and can be cleaned. The effect of the action is to set the "dirty" state of the table to true.

(:action start-move)
This action specifies the process of a waiter starting to move from one location to another. The precondition of the action specifies that the waiter must be currently located at the ?from location and the ?from and ?to locations must be connected. Additionally, the waiter must not already be moving. The effect of the action is to update the state of the world by marking the waiter as moving, updating the location of the waiter to the ?to location, and assigning the real distance the waiter has traveled as the distance between the ?from and ?to locations. The action also updates the location of the waiter. Moreover, it activates the process of moving: *(:process move-waiter).*

(:action start-clean)
This action specifies the process of a waiter starting to clean a table. The precondition of the action specifies that the waiter must be currently located at the table, the waiter must be free to clean the table, and the table must be dirty. Additionally, the table must not already be in the process of being cleaned, and the waiter must not already be moving or cleaning another table. The effect of the action is to update the state of the world by activating the process of cleaning the table: *(:process cleaning-table).*

(:action load-tray-drink)
This action represents the waiter robot loading a tray with a drink. The precondition of the action states that the waiter is at the bar location and is free to move, the drink is at the bar location and is ready to be served, and the tray has a capacity less than 3.0 and has no biscuit on it. It also ensures that the tray is not full with biscuits and that the waiter is not moving. The effect of the action is to add the drink on the tray and to increase the tray's capacity by one. It also ensures that the drink is no longer at the bar location and that the tray is not empty.

(:action load-tray-biscuit)
This action specifies a process for a waiter to load a tray with a biscuit. The precondition of the action specifies that the biscuit and the drink are available, the waiter is at the location of the biscuit and the tray, the waiter is not busy, and the tray has enough space to add the biscuit. The effect of the action adds the biscuit to the tray, increases the capacity of the tray by one unit for the biscuit, and removes the biscuit from its original location.

(:action pick-tray-drink)
This action represents the action of the waiter robot picking up a tray of drinks from a bar. The precondition of the action states that the tray is located at the bar counter and is not empty. The waiter must also be free and located at the bar counter. The effect of the action is that the waiter will carry the tray of drinks, and will no longer be free.

(:action pick-tray-biscuit)
This action specifies that a waiter can pick up a tray that has at least one biscuit on it, in order to serve it to a client. The precondition of this action includes that the tray must be at the bar, the waiter must be free and at the same location as the tray. Moreover, the capacity for biscuit on the tray must be greater than one unit. The effect of this action includes two conditions that will be true after the action has occurred: the waiter is now carrying the tray and no longer free.

(:action serve-drink-tray)
This action describes the process of a waiter serving a drink from a tray to a customer's table. The preconditions for the action include the waiter robot and the tray being at the location of the table, the waiter carrying the tray, the drinks being on the tray, and the tray having a positive capacity. Additionally, the waiter should not be currently moving or moving with the tray, and the waiter should not be free.The effects of the action include the removal of the drink from the tray, the placement of the drink at the table, the drink being marked as served, and a decrease in the tray's capacity by one unit.

(:action serve-biscuit-tray)
This action specifies the parameters and effects of serving a biscuit from a tray to a table. The action's preconditions specify that the waiter is not busy, has the biscuit on the tray, the tray is not full, and the biscuit is together with a cold drink that has been ordered. Additionally, the waiter is at the table with the tray, and the order of the table corresponds to the waiter. Finally, the waiter is not moving and is not moving with the tray. The action's effects specify that the biscuit is no longer on the tray, it is now at the table, and the biscuit has been served. The capacity of the tray is decreased by one unit.

(:action unload-tray)
This action represents the action of the waiter robot unloading a tray that he is carrying. The precondition for this action states that the waiter must be at the same location as the tray and he must be carrying the tray, he must not be a free waiter, he must not be moving with the tray and the tray must be empty of both drinks and biscuits. The effect of this action is that the waiter is no longer carrying the tray, he becomes a free waiter, and the tray remains at the same location.

(:action start-move-tray)
This action describes a waiter starting to move with a tray from one location to another. The precondition for this action states that the waiter robot is at the current location, there is a direct connection between the current location and the destination, the waiter is not already moving both with a tray o not, the waiter is carrying the tray and the tray is at the current location. The effect of the action includes that the waiter robot is now moving with the tray and no longer at the current location, but at the destination location with the tray and the real distance traveled by the waiter robot is updated to reflect the distance between the current location and the destination. Moreover, it activates the process of moving with tray: *(:process move-waiter-tray).*

2.1.5 Process

Processes, in PDDL+, are specified in order to allow the modeling of actions that occur over a period of time. By specifying processes in the domain of a PDDL+ problem, we can model more complex and realistic scenarios that involve concurrent activities and temporal constraints, such as deadlines, durations, and time windows. This allows us to reason about the temporal aspects of a problem, such as the order of events, the duration of activities, and the synchronization of concurrent activities, which is essential for many real-world planning applications.

(:process preparing-drink)
This process represents a drink that is being prepared and is activated by the action: *(:action prepare-drink)*. The precondition of this process is that the drink is being prepared. The effect of this process is to decrease the duration of the drink by one time unit, which suggests that the drink is being prepared over time and this process is responsible for advancing that time. The *(duration-drink ?d)* function likely represents the amount of time required to prepare the drink and is decreased by one unit for each application of this process, until it reaches zero and the drink is fully prepared.

(:process drinking)
This process models the action of a client drinking a drink and is activated by the action: *(:action start-drinking-cold)* or *(:action start-drinking-warm)*, depending on the type of the drinks. The precondition specifies that the drink must be in the "drinking" state. The effect of the process is to decrease the value of the *(finishing-drink ?d)* function by one unit, which tracks how much time is left until the drink is finished.

(:process move-waiter)
This process models the action of a waiter robot moving from one location to another in the environment and is activated by the action: *(:action start-move)*. The precondition states that the waiter is moving, which means the waiter is in the process of moving from one location to another. The effect of the process is to increase the distance covered by the waiter. This means that the distance covered by the waiter will be incremented by the specified amount, in this case two time units because it is moving without tray.

(:process cleaning-table)
This process is used to clean the tables that are dirty and is activated by the action: *(:action start-clean)*. The precondition of this process is that the table is in the state of cleaning and the waiter is not moving. The effect of this process is to decrease the dimension of the table yet to be cleaned by two time units.

(:process move-waiter-tray)
This process represents the action of the waiter robot moving from one location to another while carrying a tray and is activated by an action: *(:action start-move-tray)*. The precondition of the process requires that the waiter is carrying the tray, is not currently moving without the tray, and is currently moving with the tray. The effect of the process is to increase the distance covered by the waiter by one time unit. This is represented by the "increase" function and the *"distance-covered"* variable, which keeps track of the total distance covered by the waiter during the plan.

2.1.6 Events

Specifying events in a PDDL+ domain allows for the representation of instantaneous changes in the world state. Events can represent a wide range of actions or occurrences, such as the opening or closing of a door. By including events in the domain, planners can reason about the effects of these changes on the world state and generate plans that take them into account. Events are typically used in conjunction with processes, which represent ongoing actions that can have duration and continuous effects on the world state. Together, events and processes allow for a rich representation of dynamic systems and enable more realistic and complex planning scenarios.

(:event ready-drink)
This event signifies the completion of preparation of a particular drink by the barista robot and is activated by a process: *(:process preparing-drink).* The precondition for the event requires that the drink is currently being prepared and the duration of the drink's preparation is equal to zero. The effect of the event is that the drink is now ready, the barista robot who was preparing the drink is now free, the drink is now available at the bar counter and the drink is no longer being prepared.

(:event finish-drink)
This event specifies a finish-drink event, which is triggered when a person finishes drinking a specific drink and is activated by a process: *(:process drinking).* The event has a precondition that requires the drink to be currently being drunk and the *finishing-drink* function to be equal to zero, meaning that the client has finished drinking the whole drink. The effect of the event is to update the state by removing the drinking state of the drink and adding a finished state to the drink.

(:event arrive-waiter)
This event represents the arrival of the waiter robot to a destination location, such as a table or the bar counter, and is activated by a process: *(:process move-waiter).* The precondition of this event is that the waiter is currently in a state of motion and that the distance covered by the waiter is equal to the real distance that the waiter has to travel to reach the destination location. The effect of this event is that the waiter stops moving and the distance covered by the waiter is reset to zero.

(:event clean-table-done)
This event represents the completion of the cleaning process of a table and is activated by a process: *(:process cleaning-table).* The precondition of the event requires that the table is being cleaned, its cleaning process has been completed, namely that there is no table dimension yet to be cleaned, and the waiter is not moving. The effect of the event is to mark the cleaning process of the table as done, set the cleaning duration of the table equal to zero and mark the table as cleaned.

(:event arrive-waiter-tray)
This event models the arrival of a waiter carrying a tray to its destination location and is activated by a process: *(:process move-waiter-tray).* The precondition of the event checks that the waiter is not currently moving, but is instead moving with the tray. Additionally, the distance covered by the waiter should be equal to the actual distance between the source and destination locations of the tray, indicating that the waiter has arrived at the destination location. Finally, the waiter should be carrying the tray. The effect of the event is to update the state of the world. The waiter is no longer considered to be moving with the tray and the distance covered by the waiter is reset to zero, indicating that it has reached its destination.

2.2 Problem instances

2.2.1 Initialization

In PDDL+, the initialization section specifies the initial state of the problem. Specifying the initialization is necessary because it allows the planner to know the starting state of the problem and use it to search for a sequence of actions or events that will lead to the desired goal state. The initialization section provides the context for the planner to reason about the actions or events and their effects, and to determine if a particular plan is valid or not.

The assignment specifies four problem instances with increasing planning complexity. All problem instances define the objects and the thresholds required by the domain, in particular:

- **Drinks**:
  Since we have two different types of drinks (cold and warm), we will have two different duration time. 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. we set these time units by a specified variable: (**duration-drink** ?*d* - *drink*).
- **Map Connection**:
  In order to have a clear environment to manage, we set manually every connection between locations. For instance, we connected the bar counter to the nearest tables and these last ones to other locations. We set the environment thanks to some variables: (**connected** ?*l1* - *location* ?*l2* - *location*), (**distance** ?*l1* - *location* ?*l2* - *location*). Moreover, we initialized to zero the distance covered by the waiter robot and the real actual distance: (**distance-covered** ?*w* - *waiter*),(**real-distance** ?*w* - *waiter*).
- **Tables**:
  Another variable we had to set was that related to the tables, in particular as far as their dimensions were concerned in order to have a clear idea of the time units needed by the cleaning process. This variable is: (**table-dimension** ?*l* - *table*).
- **Initial Position**:
  Finally, the last variables that we had to set were those related to the initial position of objects: (**at-barista** ?*l* - *bar*), (**at-waiter** ?*l* - *bar*), (**at-tray** ?*l* - *bar*), (**at-biscuit** ?*l* - *bar* ?*c* - *food*).

  (?l-location ?c - food)

2.2.2 Goal

The goal in the problem of a PDDL+ specifies what the planner should achieve or accomplish. It defines a set of propositions that must hold true in order for the problem to be considered solved. The planner's job is to generate a sequence of actions and events that will transform the initial state of the problem into a state that satisfies the goal conditions. By specifying a clear and well-defined goal, the problem can be solved more efficiently and effectively, as the planner can focus on finding a solution that achieves the goal, rather than exploring unnecessary search space.

In our project, what we had to achieve was serving drinks to specified tables and cleaning all the dirty ones. In order to reach our goal, we set some variables: (**at-drink** ?*l* - *table* ?*d* - *drink*), (**cleaned** ?*l* - *table*).

Chapter 3

3 ENHSP different choices of search strategies and heuristics

Following the implementation of the domain and problem instances, our focus shifted to identifying the optimal search strategy and heuristics for our model. We proceeded by experimenting various **ENHSP** settings by appending the -planner flag to the command line. This flag enables the configuration of different heuristics and search strategies. We carefully analyzed the plan output generated by each planner setting and observed that a slight modification in the heuristic or search strategy can significantly impact the plan quality. Therefore, we executed our problems with all permissible configurations. The configurations prefixed with **sat** are intended for use in sat planning, while those prefixed with **opt** are tailored for optimal planning. In general, **sat** employs **Greedy Best First Search** as its search engine, whereas **opt** uses **A*** (although there may be exceptions to this rule).

The considered search strategies are the ones that we saw during the lessons:

- **Greedy Best First Search (GBF)**. Uses heuristic function as evaluation function: $f(n) = h(n)$. Always expands the node that is closest to the goal node. Eats the largest chunck out of the remaining distance, hence, ”greedy”.
- **A***. Here the evaluation function is: $f(n) = h(n) + g(n)$. Where h(n) is the heuristic function, g(n) is the cost to reach the node n. Instead, f(n) is the estimated cost of the cheapest solution through n. This method is used for finding optimal plans but it takes more time to find a solution concerning GBF.
- **wA***. Here the evaluation function is: $f(n) = w * h(n) + g(n)$. The evaluation function is the same as A* but the heuristic component is weighted by a parameter that is w.

The allowable planners' configurations are:

- **sat-hmrp**: Greedy Best First Search plus MRP heuristic.
- **sat-hmrph**: Same as before but with helpful actions.
- **sat-hadd**: Greedy Best First Search with numeric hadd.
- **sat-hradd**: As the previous, but every pair of numeric conditions is augmented with the implied redundant constraint.
- **sat-aibr**: A* plus AIBR heuristic.
- **opt-hmax**: A* with hmax numeric heuristic.
- **opt-hrmax**: Same as before, but with redundant constraints.
- **opt-blind**: this is a baseline blind heuristic that gives 1 to the state where the goal is not satisfied and 0 to the state where the goal is satisfied.

We are reporting the output plan of the problem instances tried with different planners for problems with basic functionality and for problems with optional extensions.

3.1 Basic functionality Domain-Problem

Planner	Plan Length	Elapsed Time	Metric	Planning Time	Heuristic Time	Search Time	Expanded Nodes	State Evaluated
sat-hmrp	53	13.5	30.0	2685	122	216	32	101
sat-hmrph	56	14.0	32.0	2705	71	116	38	89
sat-hadd	51	13.0	30.0	2379	58	135	65	192
sat-aibr	50	11.5	29.0	2625	342	452	258	443
opt-hmax	43	9.0	24.0	2452	306	597	2672	3340
opt-hrmax	43	9.0	24.0	2633	293	567	2672	3340
opt-blind	41	9.5	24.0	1804	3	560	12088	14539

Table 3.1: Different planner configuration planned on the first base problem instance.

Planner	Plan Length	Elapsed Time	Metric	Planning Time	Heuristic Time	Search Time	Expanded Nodes	State Evaluated
sat-hmrp	93	24.0	53.0	2682	142	220	72	242
sat-hmrph	115	28.5	66.0	2018	115	186	88	147
sat-hadd	94	27.0	54.0	2948	247	392	285	1004
sat-aibr	110	25.0	62.0	6598	4712	4993	1517	4001
opt-hmax	69	19.0	39.0	28327	21092	26174	285727	416856
opt-hrmax	69	19.0	39.0	28072	20624	25604	285727	416856
opt-blind	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM

Table 3.2: Different planner configuration planned on the second base problem instance.

Planner	Plan Length	Elapsed Time	Metric	Planning Time	Heuristic Time	Search Time	Expanded Nodes	State Evaluated
sat-hmrp	108	32.0	60.0	2440	180	273	76	255
sat-hmrph	109	31.5	61.0	2334	153	238	81	216
sat-hadd	98	30.0	56.0	2779	220	346	473	1581
sat-aibr	119	29.5	66.0	3892	1879	2073	472	1216
opt-hmax	77	23.0	43.0	33518	25579	31911	354822	518353
opt-hrmax	77	23.0	43.0	32992	24235	30433	354822	518353
opt-blind	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM

Table 3.3: Different planner configuration planned on the third base problem instance.

Planner	Plan Length	Elapsed Time	Metric	Planning Time	Heuristic Time	Search Time	Expanded Nodes	State Evaluated
sat-hmrp	212	56.0	121.0	2278	248	375	195	649
sat-hmrph	191	53.0	109.0	2263	307	448	167	551
sat-hadd	168	50.0	96.0	11391	7911	9204	17734	60175
sat-aibr	205	47.5	114.0	10202	7619	7832	1202	4654
opt-hmax	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM
opt-hrmax	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM
opt-blind	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM

Table 3.4: Different planner configuration planned on the fourth base problem instance.

3.2 Optional extensions Domain-Problem

Planner	Plan Length	Elapsed Time	Metric	Planning Time	Heuristic Time	Search Time	Expanded Nodes	State Evaluated
sat-hmrp	114	24.5	65.0	1969	745	1028	150	629
sat-hmrph	91	22.0	52.0	2602	906	1320	65	176
sat-hadd	93	23.0	55.0	5747	1934	2567	772	2848
sat-aibr	169	29.5	95.0	12959	11192	11791	1084	2516
opt-hmax	72	14.5	42.0	69867	57012	66618	340443	609915
opt-hrmax	72	14.5	42.0	110548	94183	109653	340443	609915
opt-blind	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM

Table 3.5: Different planner configuration planned on the first problem instance with optinal exstension.

Planner	Plan Length	Elapsed Time	Metric	Planning Time	Heuristic Time	Search Time	Expanded Nodes	State Evaluated
sat-hmrp	231	53.0	133.0	1623	790	968	3886	11737
sat-hmrph	240	57.5	140	1096	307	402	1063	2962
sat-hadd	148	43.5	85.0	1447	515	665	2431	8746
sat-aibr	255	51.5	139.0	19543	18600	18890	8586	21711
opt-hmax	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM
opt-hrmax	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM
opt-blind	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM

Table 3.6: Different planner configuration planned on the second problem instance with optinal exstension.

Planner	Plan Length	Elapsed Time	Metric	Planning Time	Heuristic Time	Search Time	Expanded Nodes	State Evaluated
sat-hmrp	155	43.5	89	880	138	213	170	951
sat-hmrph	181	49.5	104.0	891	157	241	324	1270
sat-hadd	150	42.0	85.0	1624	804	965	6714	27433
sat-aibr	170	36.0	95.0	73554	71866	72747	37323	123962
opt-hmax	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM
opt-hrmax	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM
opt-blind	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM

Table 3.7: Different planner configuration planned on the third problem instance with optinal exstension.

Planner	Plan Length	Elapsed Time	Metric	Planning Time	Heuristic Time	Search Time	Expanded Nodes	State Evaluated
sat-hmrp	424	105.5	245.0	8183	6598	7498	30576	95649
sat-hmrph	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM
sat-hadd	316	89.0	183.0	42048	38357	41363	130526	311582
sat-aibr	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM
opt-hmax	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM
opt-hrmax	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM
opt-blind	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM

Table 3.8: Different planner configuration planned on the fourth problem instance with optinal exstension.

Moreover, we made some considerations about the *word evolution delta*. This is a discretization delta that tell us how often the exogenous processes and events are checked, and their effect applied to the world. This delta need to be as small as possible, to ensure validity. By default, we use a delta value of 0.5, as it represents the minimum time required for the robot to move between tables without a tray. Below, we have included a table of quality parameters for the third problem, in which we varied the delta value.

Deltas	Plan Length	Elapsed Time	Metric	Planning Time	Heuristic Time	Search Time	Expanded Nodes	State Evaluated
delta = 0.5	155	42.5	88.0	1444	711	854	6432	27248
delta = 0.05	920	42.5	88.0	1529	780	1012	6432	27248
delta = 0.005	8570	42.5	88.0	2067	687	1436	6432	27248

Table 3.9: Third problem instance plan with different world evolution delta, with optional exstension.

Chapter 4

4 Results and Conclusions

The tables presented in chapter 3 demonstrate that there is no single planner that can provide an optimal plan for all problem instances, both for the main case and for the one with optional extensions.

4.1 Main Case

- **Problem 1**: As shown in Table 3.1, problem number 1 was executed by all planners. As we can clearly see from the table, the best planner is **opt-blind**, since it has the lowest plan length and planning time. As far as its elapsed time is concerned, there are other best planner, but by a paltry amount such that we can consider opt-blind the best planner for this problem.
- **Problem 2**: As shown in Table 3.2, problem number 2 was executed by all planners, except for **opt-blind** which resulted in a **ROOM** (Run Out Of Memory) due to an excessive number of expanded and evaluated nodes. As far as the plan length and elapsed time are concerned, the best planners are **opt-hmax** and **opt-hrmax**, but their planning time is significantly larger than other planners. In fact, all sat planners have a smaller planning time than opt ones, but have significant plan length and elapsed time.
- **Problem 3**: As we can clearly see in Table 3.3, problem number 3 was executed by all planners, except **opt-blind** which resulted in a **ROOM** (Run Out Of Memory) due to an excessive number of expanded and evaluated nodes. The final considerations are the same as the previous problem: the best planners are **opt-hmax** and **opt-hrmax** in regards to their plan length and elapsed time, whereas as far as planning time is concerned the best planners are the sat ones.
- **Problem 4**: As shown in table 3.4, problem 4 was executed by all sat planners, whereas all the opt planners ends to **ROOM** (Run Out Of Memory) state due to an excessive number of expanded and evaluated nodes. Keeping in account of plan length and elapsed time, the best planner is **sat-hadd**, but has a high value of planning time. Finally, as far as the planning time is concerned, the best planners are **sat-hmrp** and **sat-hmrph**. By these considerations, we can estimate the best planner is the **sat-hmrph**, because represents a way to meet all the needs we have to deal with.

4.2 Optional Extensions

- **Problem 1**: As we can see from table 3.5, problem number 1 was executed by all planners, except **opt-blind** which caused a **ROOM** (Run Out Of Memory) due to excessive number of expanded and evaluated nodes. The planner with the lowest plan length and elapsed time is **opt-hmax** but it has a high planning time. On the other hand, the planner with the lowest planning time is **sat-hmrp**, but it has a longer plan length and elapsed time.
- **Problem 2**: As shown in Table 3.6, problem number 2 was executed by all sat planners but for all opt planners it resulted in a **ROOM** (Run Out Of Memory) due to an excessive number of expanded and evaluated nodes. Among the four planners that successfully executed Problem 2, we note that **sat-hadd** and **sat-aibr** have significantly better values for elapsed time.

However, the planner with the lowest elapsed time and plan length is **sat-hadd**. Instead, the planner with the best planning time is **sat-hmrph**.

- **Problem 3**: As shown in Table 3.7, problem number 3 was executed by all sat planners but not by the opt planners, which resulted in a **ROOM** (Run Out Of Memory) due to an excessive number of expanded and evaluated nodes. In this instance, the best planner in terms of planning time is **sat-hmrp**, but in terms of elapsed time, the best planner is **sat-aibr**.
- **Problem 4**: As shown in table 3.8, problem number4 caused a **ROOM** (Run Out Of Memory) error for all planners except **sat-hmrp** and **sat-hadd**. Among these two planners, we can see that **sat-hadd** performed better, with lower elapsed time and lower plan length but **sat-hmrp** is better for the planning time.

4.3 Delta evolution

World Evolution Delta (WED) is a powerful feature that allows for modeling changes in the environment over time. In PDDL+, the environment is represented by a set of objects, each with their own state and properties. With WED, these objects can be changed over time, which makes it possible to model dynamic systems where the environment evolves or changes in some way.

As far as our *World Evolution Delta* is concerned, as we can see from table 3.9, the elapsed time for plan execution is the same for all tested delta values. However, the shortest plan length is achieved when the delta value is equal to 0.5. Therefore, we have chosen this value as the default.

5 References

1. ENHSP: PDDL+/ Numeric planning (https://sites.google.com/view/enhsp/);
2. ENHSP: How to use it (https://sites.google.com/view/enhsp/home/how-to-use-it);
3. PDDL+ - Planning.wiki: The AI Planning and PDDL wiki (https://planning.wiki/ref/pddlplus);
4. Vallati, Mauro. Search and Heuristics for Classical Planning.