UNIVERSITAT ROVIRA i VIRGILI

Assignment 1: Cleaner Robotic Task

Planning and Approximate Reasoning (MAI)

Team members
BEJARANO SEPULVEDA, Edison Jair
edison.bejarano@estudiantat.upc.edu
GUENDOUZ, Wafaa
wafaa.guendouz@estudiantat.upc.edu

Universitat Rovira i Virgili
Avinguda dels Països Catalans, 26, 43007 Tarragona, Spain
Master's degree in Artificial Intelligence
Tarragona, 2022

Contents

1 Introduction

The planning domain definition language is a set of rules and definitions to facilitate AI planning by providing a set of declarative actions and conditions that an AI system can iterate through trial and error to arrive at the most optimal solution. PDDL is composed of a domain, or a set of possible states, and certain actions with required states and effects, and a problem, which consists of a set of objects, an initial state, and a desired (goal) state.

A PDDL system will take such a domain, and attempt to solve the problem in the most optimal way. The result is a planning list, a set of actions with parameters that must be executed in that order to get the optimal result.

In this essay, we're trying to solve the problem of visiting cells of a matrix, given obstacles ahead. The matrix is a NxN office composing of N offices, in this case 9. Each office can be clean or dirty, and can contain an obstacle (a box). Limitations on offices is that each office might only contain a maximum of 1 box, and the whole matrix can only contain a maximum of N2 - 1 boxes, in this case, between 0 or 8. Each room is connected orthogonally (vertically or horizontally) via other rooms. We have a robot that can traverse rooms and push boxes from a room to another. A robot can only push one box at a time, one room at a time. A robot can only clean a room if it has no box in it (empty).

Given N=3 and an arbitrary number of boxes in different rooms and an arbitrary number of clean/dirty rooms, write a PDDL that can produce an optimized plan to achieve this task.

2 Methodology description

A well-defined plan is the cornerstone of a successful task, and for this project, a detailed plan has been defined, divided into different parts among the team members. Likewise, GitHub and its environment were used as complementary tool for version control.

For this project, the process was structured in the following parts:

- **Tools and software:**

  Github: As mentioned above, this tool facilitates code version control and the exchange of different solutions, allowing quick sharing, iteration and change and increasing development productivity. Figure 1 shows the different versions created for the resolution of this exercise. https://github.com/EjbejaranosAI/PA

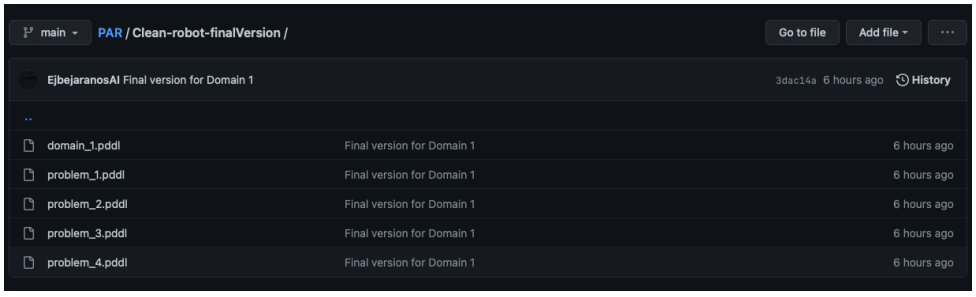

Figure 1 – GitHub repository

PDDL: This is a family of languages that allow to define planning problems and for this task it was used as the primary language to create the solutions.

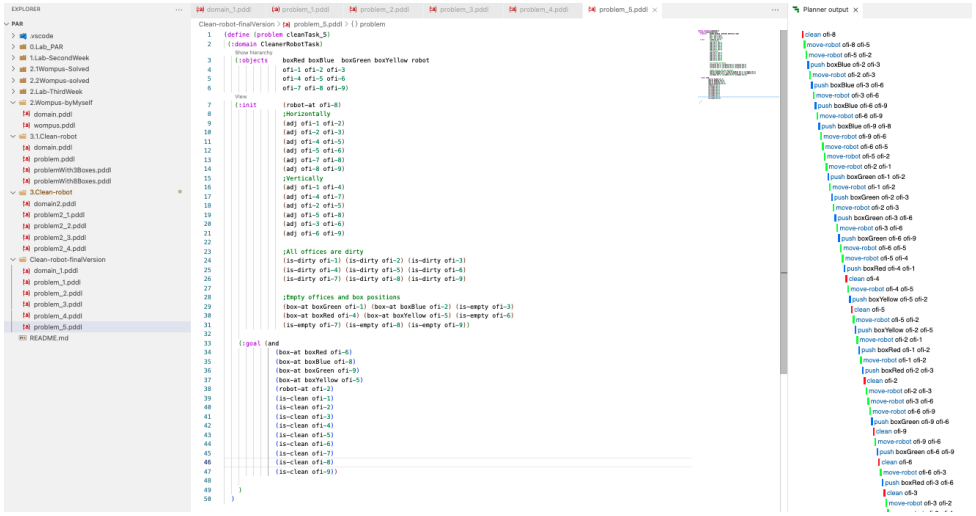

Figure 2 – Planning Domain Definition Language

- **Coordination:**

  At the time of starting this work, it was decided to carry out a division of labor, according to the main components of the delivery. That is why, it was important to coordinate and divide the different tasks, such as the code development, where by techniques as pair programming it was attached the planning solution. Other tasks, as the report it was divided in an equivalent way.

- **Decisions:**

  As a fundamental piece for the development of the project, constant communication was necessary to be able to transmit doubts, possible solutions and the most important, to facilitate the decision in many opportunities, where two different point of views it was helpful.

  One of the most significant decisions for this project was how to address this problem from the statement and how to aboard the preconditions and effects in the actions, which had a big impact in the final solution.

3 Analysis of the problem

As it is well known, in all problems the best strategy is to simplify the actors and actions as much as possible, and one way, to have a logical representation about the knowledge in the world that it is desire to explore is by CWA (Closed-World Assumption), where by a formal system of logic is used for knowledge representation, it is possible define a conceptual expression for all the assumptions in our problem.

In the challenge presented for this work, there is a world that is composed of 9 offices which can be represented as a 3x3 matrix. In this scenario, there is a robot in charge of cleaning every office. However, the existence of a box inside an office interferes with the robot's task, preventing it from cleaning the offices that contain such boxes. To solve this issue, the robot must push the boxes to other offices in order to do the cleaning. Once the cleaning task of all offices has been completed, both the robot and the boxes must reach a particular pre-established location.

With this small outline about the problem to be addressed, it was understood that the first step is to specify the **components of the PDDL planning task**, which are the following:

1. **Objects:** Things in the world that interest us.
2. **Predicates:** Properties of objects that we are interested in; they can be true or false.
3. **Initial state:** The state of the world that we are starting from.
4. **Goal specification:** Things that we want to be true.
5. **Actions/Operators:** Ways of changing the state of the world.

Another important statement to take into account are the rules that our world must follow in order to function correctly and to help meet the proposed objectives.

In the case of the robot that needs to clean all the offices and in turn move the boxes, the rules that were analyzed and used to create and develop the solution were the following:

- The robot's main mission is to clean each office and move the boxes to where it is indicated.
- The robot is not allowed to clean an office if it contains a box inside. If that is the case it is necessary to move the box.
- To perform the movement of pushing a box out from an office, it can only be done to another office that is in an empty state. That is, it is not allowed to have several boxes within the same office.
- At the end of the cleaning task the robot must leave each one the boxes at the locations indicated in the goal state.

3.1 Objects

For problem 1:

- Offices: ofi-1, ofi-2, ofi-3, ofi-4, ofi-5, ofi-6, ofi-7, ofi-8, ofi-9.
- Boxes: boxBlue, boxRed.
- Robot

For problem 2, 3 and 4:

- Offices: ofi-1, ofi-2, ofi-3, ofi-4, ofi-5, ofi-6, ofi-7, ofi-8, ofi-9.
- Boxes: boxRed, boxBlue, boxGreen
- Robot

3.2 Predicates

- (robot-at ?x): true if x is an office and the robot is at x
- (box-at ?x ?y): true if x is a box, y is an office, and x is in y
- (is-dirty ?x): true if x is an office, and is dirty
- (is-clean ?x): true if x is an office, and is clean
- (is-empty ?x): true if x is an office and x does not contain any boxes
- (adj ?x ?y)): true if x is an office, y is another office, x and y are horizontally or vertically adjacent

3.3 Initial State

The init declaration is where we specify our initial problem. Here, we must be explicit about which rooms are clean and which are not, which office contains the robot, which offices contain a box and which are empty. This is because none of this information is derivative of each other, so we can not make such assumptions, or else our plan could end up with double states (an empty room that also contains a box). In the Initial state we also define the adjacent rooms and the state of the offices (dirty or clean).

In all 4 problems, we have that:

- is-dirty (ofi-1), is-dirty (ofi-2), ... is-dirty (ofi-9) are true (we considered the case of all offices being initially dirty)
- adj (ofi-1, ofi-2), adj (ofi-2, ofi-3), adj (ofi-4, ofi-5), adj (ofi-5, ofi-6), ...adj (ofi-8, ofi-9) are true (to define the offices that are adjacent to each other horizontally) *
- adj (ofi-1, ofi-4), adj (ofi-4, ofi-1), ... adj (ofi-9, ofi-6). Are true (to define the offices that are adjacent to each other vertically) *

* As a first approach, we decided to explicitly define the reciprocity of adjacent offices in the "Initial State", since given two offices A and B, office #A being adjacent to office #B is equivalent to office #B being adjacent to office #A. Thus we had defined in the initial state that both *adj (ofi-A, ofi-B)* **and** *adj (ofi-B, ofi-A)* were true. However, we later found through testing that adding the condition of these expressions being equivalent when we build the "Actions", rather than in the "Initial State" is more efficient performance-wise. That is, we specified the adjacency condition as *((adj ?A ?B) OR (adj ?B ?A))*, and this provided a solution which took fewer steps to complete the cleaning task.

For problem 1:

- robot-at (ofi-4) is true (The robot is at office 4)
- is-empty (ofi-1),is-empty (ofi-2),is-empty (ofi-3),is-empty (ofi-4),is-empty (ofi-7),is-empty (ofi-8)is-empty (ofi-9) are true(the offices 1,2,3,4,7,8,9 have no boxes)
- box-at (box-1,ofi-5) ,box-at (box-2,ofi-6) are true (box-1 is in office 5 and box-2 is in office 6)

Everything else is false.

For problem 2:

- robot-at (ofi-4) is true (The robot is at office 4)
- is-empty (ofi-2), is-empty (ofi-3), is-empty (ofi-4), is-empty (ofi-7), is-empty (ofi-8), is-empty (ofi-9) are true(the offices 1,2,3,4,7,8,9 have no boxes)
- box-at (boxGreen,ofi-1) ,box-at (boxBlue,ofi-5), box-at (boxRed ,ofi-6) are true (boxGreen is in office 1 , boxBlue is in office 5, boxRed is in office 6)

Everything else is false.

For problem 3:

- robot-at (ofi-8) is true (The robot is at office 8)
- is-empty (ofi-4), is-empty (ofi-5), is-empty (ofi-6), is-empty (ofi-7), is-empty (ofi-8), is-empty (ofi-9) are true(the offices 4,7,8,9 have no boxes)
- box-at (boxGreen,ofi-1) ,box-at (boxBlue,ofi-2), box-at (boxRed ,ofi-3) are true (boxGreen is in office 1 , boxBlue is in office 2, boxRed is in office 3)

Everything else is false.

For problem 4:

- robot-at (ofi-8) is true (The robot is at office 8)
- is-empty (ofi-3), is-empty (ofi-5), is-empty (ofi-6), is-empty (ofi-7), is-empty (ofi-8), is-empty (ofi-9) are true(the offices 3,6,7,8,9 have no boxes)
- box-at (boxGreen,ofi-1) ,box-at (boxBlue,ofi-2), box-at (boxRed ,ofi-4) are true (boxGreen is in office 1 , boxBlue is in office 2, boxRed is in office 4)

Everything else is false.

For problem 5:

- robot-at (ofi-8) is true (The robot is at office 8)
- is-empty (ofi-3), is-empty (ofi-6), is-empty (ofi-7), is-empty (ofi-8), is-empty (ofi-9) are true(the offices 3,6,7,8,9 have no boxes)
- box-at (boxGreen,ofi-1) ,box-at (boxBlue,ofi-2), box-at (boxRed ,ofi-4), (boxYellow ,ofi-5) are true (boxGreen is in office 1 , boxBlue is in office 2, boxRed is in office 4 and boxYellow is in office 5)

Everything else is false.

3.4 Goal specification:

The goal describes our end state, we can choose what we want, and what we can ignore. In all problems, we want all rooms to be clean (this is static between problems), and we might want the boxes and robots is specific places or not. Note that by default, the plan will not attempt to preserve the location, as it tries to optimize for the shortest amount of execution. For every problem we need to specify a desired state (the goal).

For problem 1:

- robot-at (ofi-9) must be true
- box-at((box-1, ofi-6) and box-at(box-2, ofi-8) must be true
- is-clean(x) while x is from [ofi-1,ofi-2, ofi-3, ofi-4, ofi-5,ofi-6, ofi-7,ofi-8,ofi-9] must be true]

For problem 2:

- robot-at (ofi-9) must be true
- box-at((boxRed, ofi-6), box-at((boxGreen, ofi-1) and box-at(boxBlue, ofi-8) must be true
- is-clean(x) while x is from [ofi-1,ofi-2, ofi-3, ofi-4, ofi-5,ofi-6, ofi-7,ofi-8,ofi-9] must be true]

For problem 3:

- robot-at (ofi-2) must be true
- box-at((boxRed, ofi-7), box-at((boxGreen, ofi-2) and box-at(boxBlue, ofi-9) must be true
- is-clean(x) while x is from [ofi-1,ofi-2, ofi-3, ofi-4, ofi-5,ofi-6, ofi-7,ofi-8,ofi-9] must be true]

For problem 4:

- robot-at (ofi-2) must be true
- box-at((boxRed, ofi-6), box-at((boxGreen, ofi-8) and box-at(boxBlue, ofi-9) must be true
- is-clean(x) while x is from [ofi-1,ofi-2, ofi-3, ofi-4, ofi-5,ofi-6, ofi-7,ofi-8,ofi-9] must be true]

3.5 Actions:

Following the description of the previous problem below, it was understood that at least 3 actions are necessary to solve the problem, this action is:

1. **Movement operator:** Is one of the simplest and lightest actions we have, it takes 2 parameters, the office we're going from, and the office we're going to. The move precondition is that the robot is in the current office, and the office we want to go to is adjacent to this one. Its effect is that the robot is now in office 2.

   Action/Operator:

   - Description: The robot can move from x to y.
   - Precondition: robot-at(x), adj (x,y) are true. Meaning the robot is at office x and office x is adjacent to office y.
   - Effect: robot-at (y) becomes true. (robot-at x) becomes false.
     Everything else does not change.

2. **Clean operator:** This is the main action of our exercise, it takes a single parameter which is the office, and checks if we're in it, if the office is empty, and is dirty. Its effects are that the room is now not dirty. Note: because predicates aren't derivatives of each other, if we were using clean and dirty, we would need to negate one and assert the other.

   Action/Operator:

   - Description: The robot cleans x.
   - Precondition: robot-at(x), is-empty (x), is-dirty(x) are true. Meaning the robot is at office x and office x is empty(contains no boxes) and the office x is dirty.
   - Effect: is-clean(x) becomes true. is-dirty(x) becomes false.
     Everything else does not change.

3. **Push:** One of the complex actions we have, push takes 3 conditions, which box we want to push? from where? and where to?.

   It's precondition is that we need to make sure that the robot is in the current office, the box we want to push is in the current office, we're pushing to an adjacent office, and that office is empty.

   Its effect is that the box is now at office 2, so is the robot, and that office 2 is not empty anymore. Furthermore, to avoid 2 places bugs, we need to mention that room 1 is now empty, and that neither the box, nor the robot, are in that room.

   Action/Operator:

   - Description: The robot pushes a box from an office x to an office y.
   - Precondition: box-at(box,x), adj (x,y), robot-at(x),is-empty(y), is-dirty(x) are true. is-empty(x) is false.
   - Effect: box-at (box,y), is-empty (x) becomes true.box-at (box, x), is-empty (y)becomes false.
     Everything else does not change.

4 PDDL Implementation

As a proposed solution, and as can be seen in the image Figure 7, we created one single domain.pddl file, and 5 problemN.pddl files.

4.1 Domain

Clean-robot-finalVersion > domain_1.pddl > ...

```
(define (domain CleanerRobotTask)
    (:requirements :strips :equality)
    (:predicates    (robot-at ?x) 3 1 1
                    (box-at ?x ?y) 1 1 1
                    (is-dirty ?x) 2 1
                    (is-clean ?x) 1
                   (is-empty ?x) 3 1 1
                    (adj ?x ?y) 4)
    (:action clean
        :parameters (?ofi)
        :precondition (and
                            (robot-at ?ofi)
                            (is-empty ?ofi)
                            (is-dirty ?ofi)
                        )
        :effect (and
                    (not (is-dirty ?ofi))
                    (is-clean ?ofi)
                )
    )

    (:action move-robot
        :parameters (?from ?to)
        :precondition (and
                            (robot-at ?from)
                            (or (adj ?from ?to)
                                (adj ?to ?from))
                        )
        :effect (and
                    (not(robot-at ?from))
                    (robot-at ?to)
                )
    )

    (:action push
        :parameters (?box ?from ?to)
        :precondition (and
                            (box-at ?box ?from)
                            (or (adj ?from ?to)
                                (adj ?to ?from))
                            (robot-at ?from)
                            (not (is-empty ?from))
                            (is-empty ?to)
                            (is-dirty ?from)
                        )
        :effect (and
                    (not (box-at ?box ?from))
                    (box-at ?box ?to)
                    (is-empty ?from)
                    (not (is-empty ?to))
                )
    )
)
```

Figure 3 – PDDL domain script implemented.

4.1.1 Predicates

```
(:predicates    (robot-at ?x)
                (box-at ?x ?y)
                (is-dirty ?x)
                (is-clean ?x)
                (is-empty ?x)
                (adj ?x ?y))
```

Figure 4 – Predicates implemented.

The predicates ‘is-clean‘ and ‘is-dirty‘ are boolean and opposite of each other, so we can reduce the amount of possible bugs and memory footprint by using ‘(not (is-dirty ?office))‘ instead of ‘(is-clean ?office)‘, or ‘(not (is-clean ?office))‘ instead of ‘(is-dirty ?office)‘.

4.1.2 Actions

1. **Clean:** For the Clean action has only one parameter that is office(ofi), it has three preconditions, the robot must be in the office(robot-at ?ofi), the office must be empty(is-empty ?ofi) and the office must be dirty(is-dirty ?ofi). Its effect is the office won't be dirty anymore(not(is-dirty ?ofi)) and the office is clean(is-clean ?ofi).

```
(:action clean
    :parameters (?ofi)
    :precondition (and
                    (robot-at ?ofi)
                    (is-empty ?ofi)
                    (is-dirty ?ofi)
                )
    :effect (and
                (not (is-dirty ?ofi))
                (is-clean ?ofi)
            )
)
```

Figure 5 – Clean Action implemented.

   Note: because predicates aren't derivatives of each other, if we were using clean and dirty, we would need to negate one and assert the other

2. **Move to:** The action move-robot needs two parameters the source (?from) and the destination (?to). It needs three preconditions, the robot must be in the source location(robot-at ?from) and both offices must be adjacents (adj ?from ?to)or(adj ?to ?from).

```
(:action move-robot
    :parameters (?from ?to)
    :precondition (and
                        (robot-at ?from)
                        (or (adj ?from ?to)
                            (adj ?to ?from))
                    )
    :effect (and
                    (not(robot-at ?from))
                    (robot-at ?to)
                )
)
```

Figure 6 – move-robot Action implemented.

Note: we must also be explicit that the robot is no longer at room 1, this is to avoid bugs in which the robot is in 2 places at the same time.

3. **Push:** In this action we need 3 parameters, the box(box?), the source office(?from) and the destination office(?to). Its preconditions are, the box must be in the source location (box-at ?box ?from), the two offices must be adjacent (adj ?from ?to), the robot must be at the source office(robot-at ?from), the source location can't be empty(not)(is-empty ?from), the destination office must be empty(is-empty ?to)and the source location must be dirty. Its effects are, the box won't be at the source office(not(box-at ?box ?from)), the box will be at destination office(box-at box? ?to), the source office will become empty(is-empty ?from) and the destination office will become not empty(not(is-empty ?to)).

```
(:action push
    :parameters (?box ?from ?to)
    :precondition (and
                        (box-at ?box ?from)
                        (adj ?from ?to)
                        (robot-at ?from)
                        (not (is-empty ?from))
                        (is-empty ?to)
                        (is-dirty ?from)
                    )
    :effect (and
                    (not (box-at ?box ?from))
                    (box-at ?box ?to)
                    (is-empty ?from)
                    (not (is-empty ?to))
                )
)
)
```

Figure 7 – Push Action implemented.

Note: in PDDL 2.2, we can use derived predicates, this essentially means that we can turn a predicate by turning another one, for example, pushing a box from office 1 to 2, can have a derived predicate that robot is now at office 2 (instead of doing that manually).

4.2 Problem

5 Testing cases and results

Sometimes, reaching an optimal solution can take longer than you imagine, for which good and logical assumptions must be proposed, so that our rules, objects and predicates can work in a harmonious way, complying with the rules and satisfying the objectives. However, in the real world, getting all of this in the first few tries can be a bit tricky (unless you're an expert in the field).

5.1 Steps to arrive for the optimal solution

The way to develop the solution was to write two scripts, one of them the domain in which the predicates and actions were reflected, with which an attempt was made to start with those that were given in the statement of the problem, but as Good results were obtained, an attempt was made to enter other predicates as shown in the figure 8, where, thinking of a solution, more complexity was provided than was needed, leading to problems such as the one for which no solutions were found by the PDDL.

```
(define (domain CleanerRobotTask)
    (:requirements :typing :strips)
    (:predicates

        (at ?what ?square)   ;ok
        (adj ?from ?to)  ;ok
        (is-boxRed ?box)   ;ok
        (is-robot ?who)        ;ok
        (box-in ?square ?box)   ;ok
        (is-clean ?square)   ;ok
        (is-dirty ?square)  ;ok
```

Figure 8 – logical errors adding unnecessary predicates

That is why to develop this project, it was necessary to restart with the previously assumptions explained by the professor, however, it was realized that if the preconditions or the effects in each of the actions were not established correctly, it could lead to a wrong logic and therefore the planning obtained as a result would be wrong as observed in the figure 9, where the action clean it was executed before the push. This kind of issues are very common if a preconditions in the action are wrong or the effects are not well established. After this, it was noticed that on some occasions the error of the functions came from not updating the position states in the effects, assuming that they would be updated autonomously.

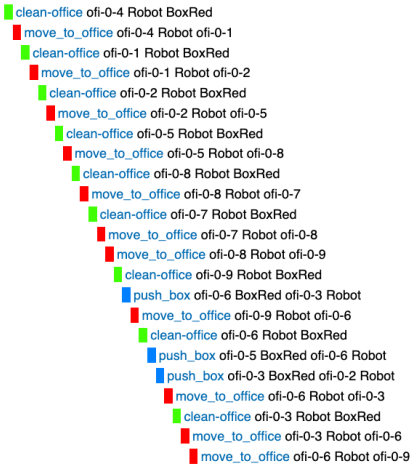

Figure 9 – Logical error in the cleaning action

Another interesting fact that was perceived in the development of the practice was the fact of understanding that what was not established as positive in the initial stages of the problems would be left as false, which meant some failures.

At the time of obtaining the first solutions that satisfied the objectives, it was perceived that, by establishing the adjacency between all the offices, this presented an abundance in the way it was described as shown in the figure, so we looked for a way to improve this, becoming as shown in the figure by using OR in the move and push actions, allowing to reduce initial parameters in the "init" of the problem, reaching an optimization as shown in the figure 11, decreasing the nodes generated from 200 to 176.

```
Total time: 0.004
Nodes generated during search: 200
Nodes expanded during search: 61
Plan found with cost: 33
BFS search completed
```

Figure 10 – Solution without the use of OR

```
Total time: 0.004
Nodes generated during search: 176
Nodes expanded during search: 54
Plan found with cost: 33
BFS search completed
```

Figure 11 – Solution with the use of OR

5.2 Analysis of the results

With the purpose of testing the solution developed in the domain, different problems were developed, 5 to be specific, where it is tested if the computed solutions are correct by varying some components in the planning. Some of the modified parameters from one problem to another are the location of the objects, the number of boxes, where it was tested with 2, 3 and 4 boxes. Below are 5 cases where the initial state and assumptions of our world are found, along with the desired target. On the other hand, the results obtained by the domain when presenting these problems are presented.

5.2.1 Case 1

Description: In this case, the proposed case of the original problem was used, where as objects we have two boxes in office 5 and 6 as shown in the figure 12 and the robot in office number 4 in the initial state . The objective in this case is for the robot to clean all the offices and also finish its journey in office 9, previously leaving the red and blue blue boxes in offices 6 and 8 correspondingly.

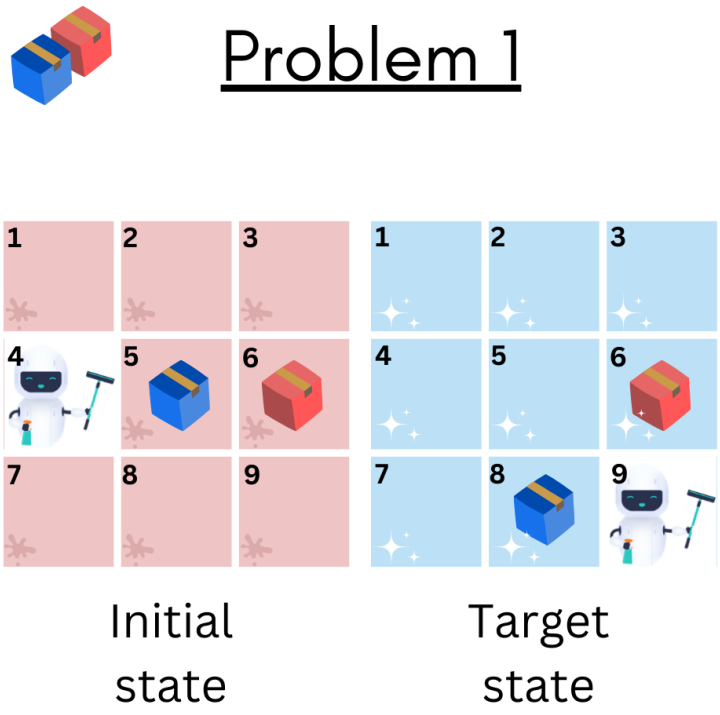

Figure 12 – Cleaner Robotic Task - Problem 1

Planning results In other hand, as is show in the figure 14, the planning result for this problem was solved in a time of 0.016 seconds, generating 1015 nodes during its result, of which only 317 were expanded and in the end it cost 29 actions to solve the task.

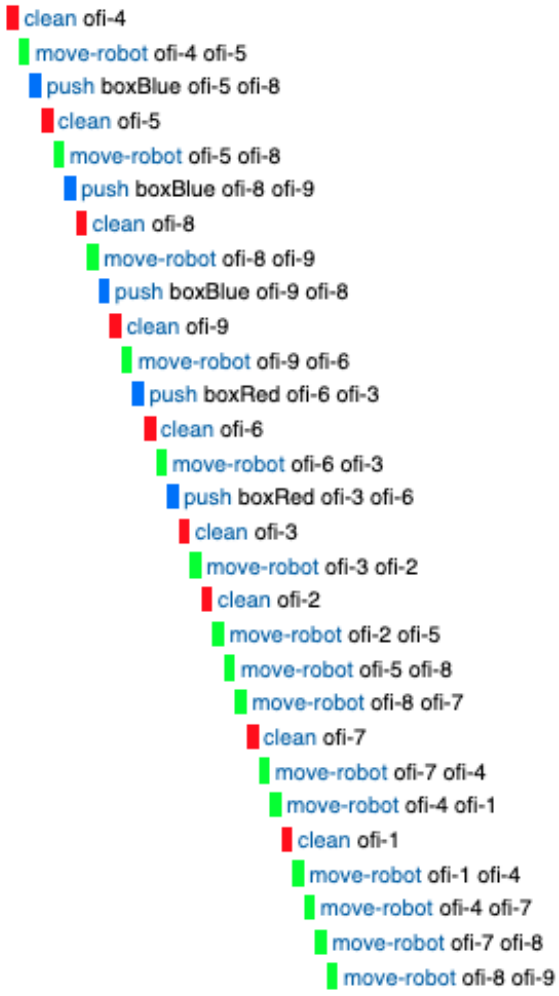

Figure 13 – Planning solution for problem 1

```
Total time: 0.016
Nodes generated during search: 1015
Nodes expanded during search: 317
Plan found with cost: 29
BFS search completed


Plan found:
0.00100: (clean ofi-4)
0.00200: (move-robot ofi-4 ofi-5)
0.00300: (push boxblue ofi-5 ofi-8)
0.00400: (clean ofi-5)
0.00500: (move-robot ofi-5 ofi-8)
0.00600: (push boxblue ofi-8 ofi-9)
0.00700: (clean ofi-8)
0.00800: (move-robot ofi-8 ofi-9)
0.00900: (push boxblue ofi-9 ofi-8)
0.01000: (clean ofi-9)
0.01100: (move-robot ofi-9 ofi-6)
0.01200: (push boxred ofi-6 ofi-3)
0.01300: (clean ofi-6)
0.01400: (move-robot ofi-6 ofi-3)
0.01500: (push boxred ofi-3 ofi-6)
0.01600: (clean ofi-3)
0.01700: (move-robot ofi-3 ofi-2)
0.01800: (clean ofi-2)
0.01900: (move-robot ofi-2 ofi-5)
0.02000: (move-robot ofi-5 ofi-8)
0.02100: (move-robot ofi-8 ofi-7)
0.02200: (clean ofi-7)
0.02300: (move-robot ofi-7 ofi-4)
0.02400: (move-robot ofi-4 ofi-1)
0.02500: (clean ofi-1)
0.02600: (move-robot ofi-1 ofi-4)
0.02700: (move-robot ofi-4 ofi-7)
0.02800: (move-robot ofi-7 ofi-8)
0.02900: (move-robot ofi-8 ofi-9)
Metric: 0.029
Makespan: 0.029
States evaluated: undefined
Planner found 1 plan(s) in 0.958secs.
```

Figure 14 – Results problem 1

It is observed for this case that all the objectives were set by the plan satisfactorily.

5.2.2 Case 2

Description: Unlike the previous case, where only 2 boxes were used, in this one more object was attached, which was a green box in the office with location 1, with the intention of complicating and testing the solution developed in the domain.pddl . The objective for this case is very similar to that in the previous case, only adding that the green box must be left in the initial position 1 in the target as is show in the picture 15.

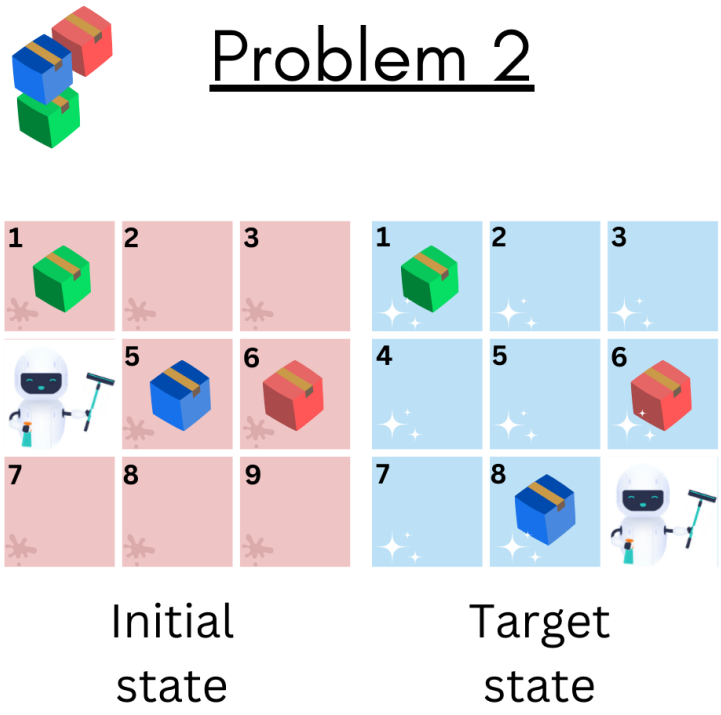

Figure 15 – Cleaner Robotic Task - Problem 2

Planning results:

As planned for this case, and how is show in the figure 17, the complexity would increase, causing the time to solve it to expand almost six times, becoming 0.06 seconds, generating 3 times more nodes in the search, which is equivalent to 3013 nodes, and in turn expanding 815 , which means a big difference with respect to the 317 obtained in case 1. The cost for this solution was 33 actions, generating 4 more than before.

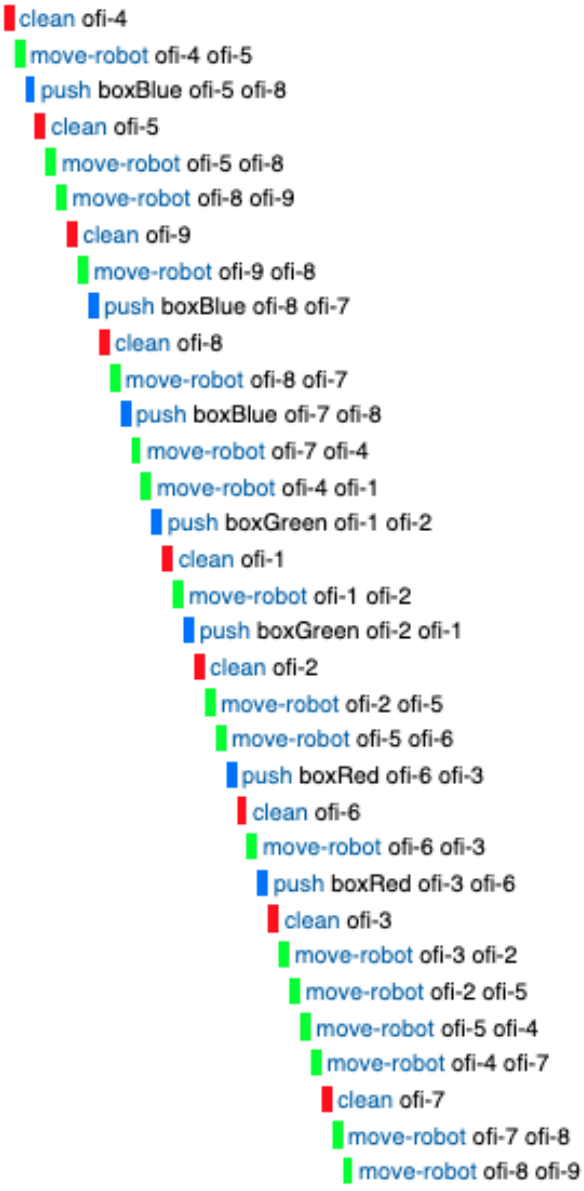

Figure 16 – Planning solution for problem 1

```
Total time: 0.06
Nodes generated during search: 3013
Nodes expanded during search: 815
Plan found with cost: 33
BFS search completed


Plan found:
0.00100: (clean ofi-4)
0.00200: (move-robot ofi-4 ofi-5)
0.00300: (push boxblue ofi-5 ofi-8)
0.00400: (clean ofi-5)
0.00500: (move-robot ofi-5 ofi-8)
0.00600: (move-robot ofi-8 ofi-9)
0.00700: (clean ofi-9)
0.00800: (move-robot ofi-9 ofi-8)
0.00900: (push boxblue ofi-8 ofi-7)
0.01000: (clean ofi-8)
0.01100: (move-robot ofi-8 ofi-7)
0.01200: (push boxblue ofi-7 ofi-8)
0.01300: (move-robot ofi-7 ofi-4)
0.01400: (move-robot ofi-4 ofi-1)
0.01500: (push boxgreen ofi-1 ofi-2)
0.01600: (clean ofi-1)
0.01700: (move-robot ofi-1 ofi-2)
0.01800: (push boxgreen ofi-2 ofi-1)
0.01900: (clean ofi-2)
0.02000: (move-robot ofi-2 ofi-5)
0.02100: (move-robot ofi-5 ofi-6)
0.02200: (push boxred ofi-6 ofi-3)
0.02300: (clean ofi-6)
0.02400: (move-robot ofi-6 ofi-3)
0.02500: (push boxred ofi-3 ofi-6)
0.02600: (clean ofi-3)
0.02700: (move-robot ofi-3 ofi-2)
0.02800: (move-robot ofi-2 ofi-5)
0.02900: (move-robot ofi-5 ofi-4)
0.03000: (move-robot ofi-4 ofi-7)
0.03100: (clean ofi-7)
0.03200: (move-robot ofi-7 ofi-8)
0.03300: (move-robot ofi-8 ofi-9)
Metric: 0.033
Makespan: 0.033
States evaluated: undefined
Planner found 1 plan(s) in 0.448secs.
```

Figure 17 – Results problem 1

5.2.3 Case 3

Description: For this case, the number of boxes was kept at 3, however their positions were set to 1 for the green box, 2 for the blue box and 3 for the red box, leaving the robot in office 8 for its status. initial. The intention of this case is to observe how effective the solution can become when moving objects within the established world. On the other hand, the goal in this problem is for the red box to move from location 3 to 7 and the green box from office 1 to 9, also leaving the blue box in position 8 and the robot in the initial position. of the blue one that was office 2. With this, the robot will have to push the red and blue boxes moving from one corner to another, **figure 18**, with which the complexity is expected to increase with respect to the previous case .

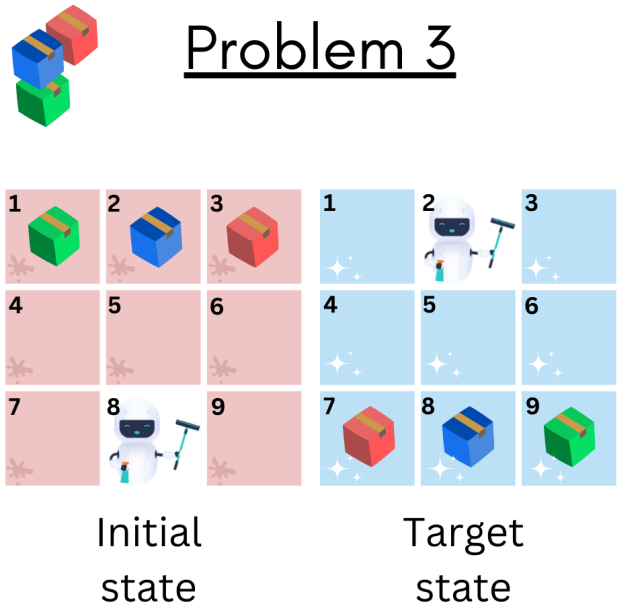

Figure 18 – Cleaner Robotic Task - Problem 3

Planning results:

For this example, the solution took 0.76 seconds as it is shown in the picture **20**, which allows us to observe that it increased about 1.2 times in time with respect to case two.On the other hand, the nodes generated by this case were 3677, which shows an increase of 664 nodes with respect to the previous case.

In the same way, the expanded nodes increased by 207, for a total of 1022 and the cost went from 33 to 44 shares. With this we understand that the position assumes great importance in the cost of a solution.

Also for this case the solution obtained satisfies the objectives proposed in this case.

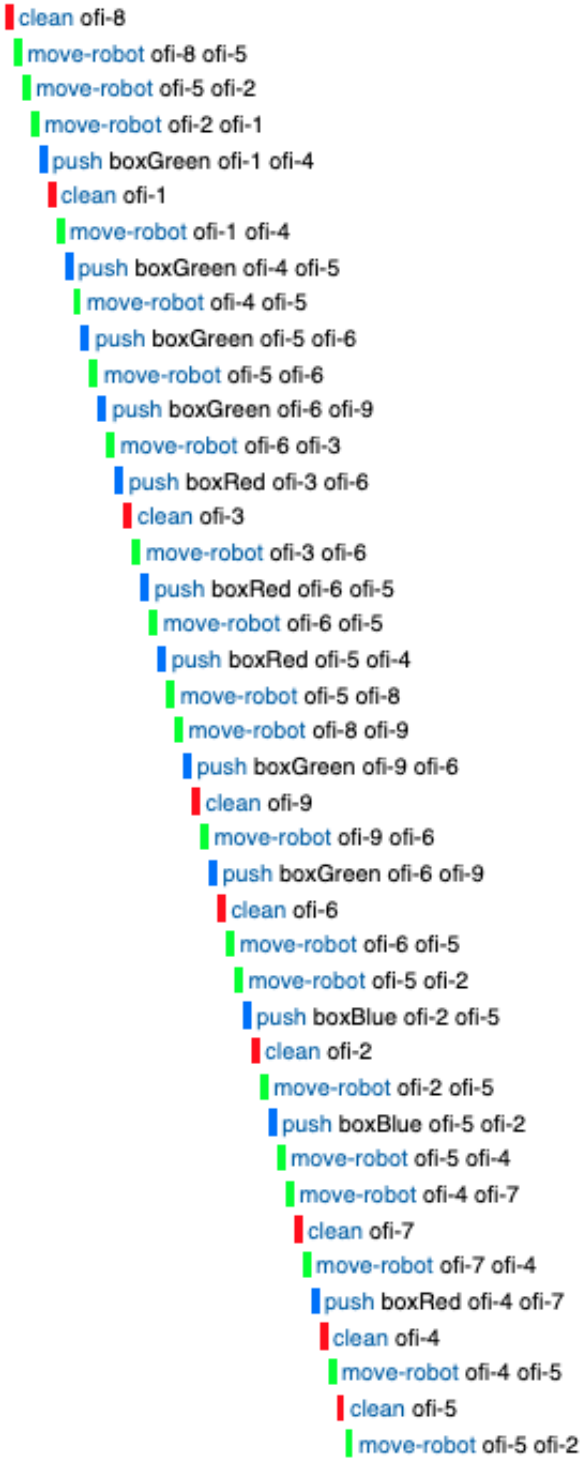

Figure 19 – Planning solution for problem 1

```
Total time: 0.076
Nodes generated during search: 3677
Nodes expanded during search: 1022
Plan found with cost: 41
BFS search completed


Plan found:
0.00100: (clean ofi-8)
0.00200: (move-robot ofi-8 ofi-5)
0.00300: (move-robot ofi-5 ofi-2)
0.00400: (move-robot ofi-2 ofi-1)
0.00500: (push boxgreen ofi-1 ofi-4)
0.00600: (clean ofi-1)
0.00700: (move-robot ofi-1 ofi-4)
0.00800: (push boxgreen ofi-4 ofi-5)
0.00900: (move-robot ofi-4 ofi-5)
0.01000: (push boxgreen ofi-5 ofi-6)
0.01100: (move-robot ofi-5 ofi-6)
0.01200: (push boxgreen ofi-6 ofi-9)
0.01300: (move-robot ofi-6 ofi-3)
0.01400: (push boxred ofi-3 ofi-6)
0.01500: (clean ofi-3)
0.01600: (move-robot ofi-3 ofi-6)
0.01700: (push boxred ofi-6 ofi-5)
0.01800: (move-robot ofi-6 ofi-5)
0.01900: (push boxred ofi-5 ofi-4)
0.02000: (move-robot ofi-5 ofi-8)
0.02100: (move-robot ofi-8 ofi-9)
0.02200: (push boxgreen ofi-9 ofi-6)
0.02300: (clean ofi-9)
0.02400: (move-robot ofi-9 ofi-6)
0.02500: (push boxgreen ofi-6 ofi-9)
0.02600: (clean ofi-6)
0.02700: (move-robot ofi-6 ofi-5)
0.02800: (move-robot ofi-5 ofi-2)
0.02900: (push boxblue ofi-2 ofi-5)
0.03000: (clean ofi-2)
0.03100: (move-robot ofi-2 ofi-5)
0.03200: (push boxblue ofi-5 ofi-2)
0.03300: (move-robot ofi-5 ofi-4)
0.03400: (move-robot ofi-4 ofi-7)
0.03500: (clean ofi-7)
0.03600: (move-robot ofi-7 ofi-4)
0.03700: (push boxred ofi-4 ofi-7)
0.03800: (clean ofi-4)
0.03900: (move-robot ofi-4 ofi-5)
0.04000: (clean ofi-5)
0.04100: (move-robot ofi-5 ofi-2)
Metric: 0.041
Makespan: 0.041
States evaluated: undefined
Planner found 1 plan(s) in 0.513secs.
```

Figure 20 – Results problem 1

5.2.4 Case 4

Description: For case number 4 and as shown in the figure **21**, a configuration of is added. different box locations, where in the initial state, the three boxes will be grouped in the upper left corner being position 1 for the green box, position 2 for the blue box and position 4 for the red box, leaving the robot in the same location as in the previous case, position8.

As an objective of this case, it is established that the boxes must end up grouped in the lower right corner, leaving them in positions 8, 6 and 9 for the blue, red and green boxes in that respective order, leaving the robot in office 2. This case represents a similarity with the previous one and the intention is that the results are similar to case 3.

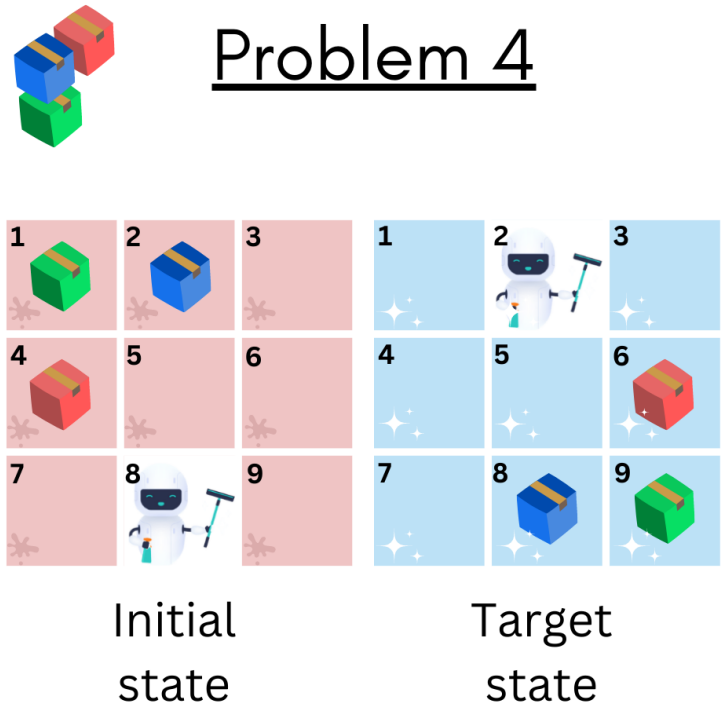

Figure 21 – Cleaner Robotic Task - Problem 4

Resutls

In this case, figure **23**, the results with respect to time varied, thus leaving a time of 0.036, which means that it was reduced by 2.1 times the time of the previous case. Another important aspect that varied were the nodes generated during the search, which were reduced by 1,909 nodes, with which only 1,768 were generated in this case, which explains the reduction in time by almost half. Likewise, the expanded nodes were 484 reducing 538 expanded nodes from the previous case. However, the cost did increase by 3 actions, thus making a total of 44. This indicates that although the difficulty in generating the planning may be less, this may generate a higher cost of actions.

Figure 22 – Planning solution for problem 1

```
Total time: 0.036
Nodes generated during search: 1768
Nodes expanded during search: 484
Plan found with cost: 43
BFS search completed


Plan found:
0.00100: (clean ofi-8)
0.00200: (move-robot ofi-8 ofi-5)
0.00300: (move-robot ofi-5 ofi-2)
0.00400: (push boxblue ofi-2 ofi-5)
0.00500: (move-robot ofi-2 ofi-5)
0.00600: (push boxblue ofi-5 ofi-8)
0.00700: (move-robot ofi-5 ofi-4)
0.00800: (push boxred ofi-4 ofi-5)
0.00900: (move-robot ofi-4 ofi-1)
0.01000: (push boxgreen ofi-1 ofi-2)
0.01100: (move-robot ofi-1 ofi-2)
0.01200: (push boxgreen ofi-2 ofi-3)
0.01300: (move-robot ofi-2 ofi-3)
0.01400: (push boxgreen ofi-3 ofi-6)
0.01500: (move-robot ofi-3 ofi-6)
0.01600: (move-robot ofi-6 ofi-9)
0.01700: (clean ofi-9)
0.01800: (move-robot ofi-9 ofi-6)
0.01900: (push boxgreen ofi-6 ofi-9)
0.02000: (clean ofi-6)
0.02100: (move-robot ofi-6 ofi-5)
0.02200: (push boxred ofi-5 ofi-6)
0.02300: (move-robot ofi-5 ofi-2)
0.02400: (clean ofi-2)
0.02500: (move-robot ofi-2 ofi-5)
0.02600: (clean ofi-5)
0.02700: (move-robot ofi-5 ofi-2)
0.02800: (move-robot ofi-2 ofi-1)
0.02900: (clean ofi-1)
0.03000: (move-robot ofi-1 ofi-4)
0.03100: (clean ofi-4)
0.03200: (move-robot ofi-4 ofi-5)
0.03300: (move-robot ofi-5 ofi-2)
0.03400: (move-robot ofi-2 ofi-3)
0.03500: (clean ofi-3)
0.03600: (move-robot ofi-3 ofi-2)
0.03700: (move-robot ofi-2 ofi-5)
0.03800: (move-robot ofi-5 ofi-8)
0.03900: (move-robot ofi-8 ofi-7)
0.04000: (clean ofi-7)
0.04100: (move-robot ofi-7 ofi-8)
0.04200: (move-robot ofi-8 ofi-5)
0.04300: (move-robot ofi-5 ofi-2)
Metric: 0.043000000000000003
Makespan: 0.043000000000000003
States evaluated: undefined
Planner found 1 plan(s) in 0.532secs.
```

Figure 23 – Results problem 1

5.2.5 Case 5

Description:

For the last case, a last object was added, this being a yellow box as shown in the figure **24**, positioning it in office number 5 in its initial and final state, leaving the other boxes in the same positions as the previous case. to identify the impact of another object on the problem.

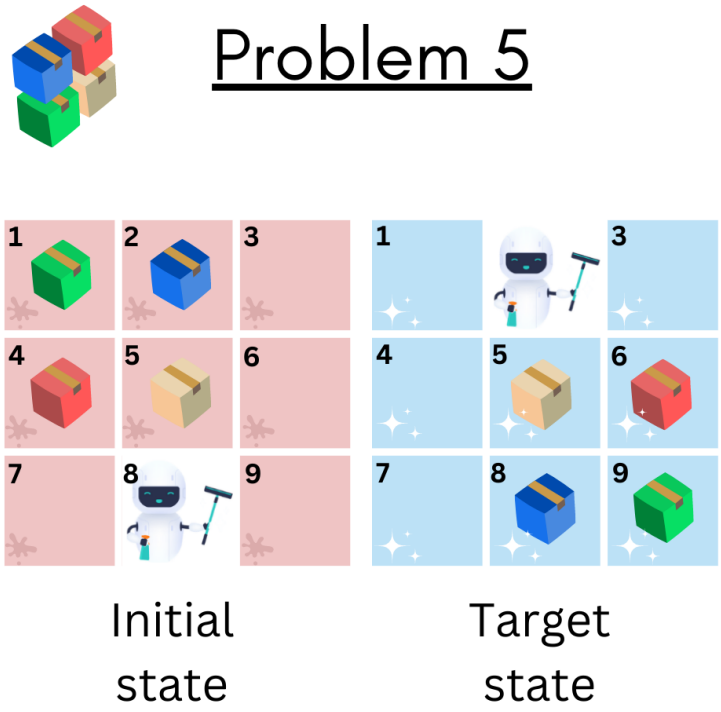

Figure 24 – Cleaner Robotic Task - Problem 5

Planning results:

For this case and as we can see in figure **26**, the time obtained was 0.08, which is equivalent to 2.2 times more than the previous case, coming to resemble the time in case 3.

Regarding the nodes generated in this search, they were 3240, 1472 more than in the previous case and 437 less than case 3. With respect to the expanded nodes, they were 880, 396 more than case 4 and 142 less than case 3. As cost of actions, it was observed that 55 were needed to be able to solve the problem satisfactorily, exceeding 43 and 41 actions in case 4 and 3 respectively.

From this last experiment it was possible to show that the positions in which the objects are placed can be equivalent to more in the cost of nodes than the same number of objects. On the other hand, the necessary actions increase as you increase the objects in the assumptions of our problems.

As a final conclusion, it is observed that the planning can improve as the approaches of the predicates and actions are modified, becoming appropriate to use some logic gates such as XOR, OR and functions such as forall. This could mean possible improvements in the logic and consequently in the planning results.

Figure 25 – Planning solution for problem 1

```
Total time: 0.08
Nodes generated during search: 3240
Nodes expanded during search: 880
Plan found with cost: 55
BFS search completed


Plan found:
0.00100: (clean ofi-8)
0.00200: (move-robot ofi-8 ofi-5)
0.00300: (move-robot ofi-5 ofi-2)
0.00400: (push boxblue ofi-2 ofi-3)
0.00500: (move-robot ofi-2 ofi-3)
0.00600: (push boxblue ofi-3 ofi-6)
0.00700: (move-robot ofi-3 ofi-6)
0.00800: (push boxblue ofi-6 ofi-9)
0.00900: (move-robot ofi-6 ofi-9)
0.01000: (push boxblue ofi-9 ofi-8)
0.01100: (move-robot ofi-9 ofi-6)
0.01200: (move-robot ofi-6 ofi-5)
0.01300: (move-robot ofi-5 ofi-2)
0.01400: (move-robot ofi-2 ofi-1)
0.01500: (push boxgreen ofi-1 ofi-2)
0.01600: (move-robot ofi-1 ofi-2)
0.01700: (push boxgreen ofi-2 ofi-3)
0.01800: (move-robot ofi-2 ofi-3)
0.01900: (push boxgreen ofi-3 ofi-6)
0.02000: (move-robot ofi-3 ofi-6)
0.02100: (push boxgreen ofi-6 ofi-9)
0.02200: (move-robot ofi-6 ofi-5)
0.02300: (move-robot ofi-5 ofi-4)
0.02400: (push boxred ofi-4 ofi-1)
0.02500: (clean ofi-4)
0.02600: (move-robot ofi-4 ofi-5)
0.02700: (push boxyellow ofi-5 ofi-2)
0.02800: (clean ofi-5)
0.02900: (move-robot ofi-5 ofi-2)
0.03000: (push boxyellow ofi-2 ofi-5)
0.03100: (move-robot ofi-2 ofi-1)
0.03200: (push boxred ofi-1 ofi-2)
0.03300: (move-robot ofi-1 ofi-2)
0.03400: (push boxred ofi-2 ofi-3)
0.03500: (clean ofi-2)
0.03600: (move-robot ofi-2 ofi-3)
0.03700: (move-robot ofi-3 ofi-6)
0.03800: (move-robot ofi-6 ofi-9)
0.03900: (push boxgreen ofi-9 ofi-6)
0.04000: (clean ofi-9)
0.04100: (move-robot ofi-9 ofi-6)
0.04200: (push boxgreen ofi-6 ofi-9)
0.04300: (clean ofi-6)
0.04400: (move-robot ofi-6 ofi-3)
0.04500: (push boxred ofi-3 ofi-6)
0.04600: (clean ofi-3)
0.04700: (move-robot ofi-3 ofi-2)
0.04800: (move-robot ofi-2 ofi-1)
0.04900: (clean ofi-1)
0.05000: (move-robot ofi-1 ofi-4)
0.05100: (move-robot ofi-4 ofi-7)
0.05200: (clean ofi-7)
0.05300: (move-robot ofi-7 ofi-8)
0.05400: (move-robot ofi-8 ofi-5)
0.05500: (move-robot ofi-5 ofi-2)
Metric: 0.055
Makespan: 0.055
States evaluated: undefined
Planner found 1 plan(s) in 0.557secs.
```

Figure 26 – Results problem 1

References

[1] Planning.Wiki - The AI Planning PDDL Wiki https://planning.wiki/guide/whatis/pddl

[2] "Closed-world assumption". Wikipedia, 2022, https://en.m.wikipedia.org/wiki/Closed-world$_assumption$