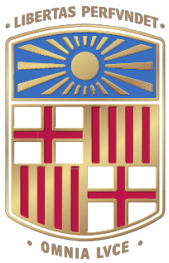

Practical Exercise 1: Cleaner Robotic Task

Planning & Approximate Reasoning

Master in Artificial Intelligence

Students:
- Alam López
- Mario Rosas

Professor: Hatem A. Rashwan

Academic Year: 2023-1

Laboratory Group: 3

October 18, 2023

Contents

1

Cleaner Robotic Task

1.1 | Introduction

In this task, we have a robot that needs to clean a floor of a building that contains n^2 offices arranged in a matrix with n rows and n columns. The robot's primary goal is to clean every office that is dirty and leave the boxes in their designated locations when necessary. There are numerous states in which the environment, including the robot, maybe at the beginning and at the end of each run.

The robot will begin cleaning an office given an initial configuration, which includes some clean and unclean offices as well as a box in some rooms. All tests will conclude when all rooms are clean and the boxes are in their final indicated positions. The initial state, the final state, or both may differ for each run and must be explicitly indicated in the problem definition.

The important rules of the environment are:

- The robot can only move between adjacent offices in a horizontal or vertical direction.
- An office may be either empty or contain only one box at a time.
- An office may be either clean or dirty, but never both.
- A box can be pushed by the robot only between adjacent offices.
- The number of boxes may be between 0 and $n^2 - 1$.

In this project, we create and implement a planner using PDDL to address the cleaning issue and investigate how the solver method finds effective ways for the robot to move from the starting world state to the desired state. We tested our implementation with 5 cases of increasing complexity and environmental conditions to analyze the solutions provided by the solver method.

1.2 | Analysis of the problem

To solve the task, we decided to use STRIPS. In order to do so, we first modeled this problem by defining the predicates, then with the predicates defined, the actions to be performed with their corresponding parameters, preconditions, and effects, as follows:

Predicates:

- Robot-location(*of*): the robot is in the office *of*.

- Box-location(*b*, *of*): box *b* is located in office *of*.
- Dirty(*of*): office *of* is dirty.
- Clean(*of*): office *of* is clean.
- Empty(*of*): there isn't any box in office *of*.
- Adjacent(*of1*, *of2*): offices *of1* and *of2* are horizontally or vertically adjacent.

With these simple predicates defined, we can express any state of the environment without ambiguity, by setting the relationships of the objects in a general way, independently of the value of "n".

Now, the next step is to determine how any actual state can be modified to the next one, in this specific case by the robot. We established **three actions**, that the robot can perform in order to alter the environment. Each of them, with its proper restrictions or so-called preconditions, and effects once the action is applied.

Actions:

- Move(*of1*, *of2*): Move from office of1 to office of2
- Clean-office(*of*): Clean the office of
- Push(*b*, *of1*, *of2*): Push a box b, from of1 to of2

For example, the simplest of the actions performed by the robot is to move from one office (of1) to another one (of2). So we can call the action: **move**. In order to move from one office, the robot needs to be inside that office itself. Finally reminding the rules given, the robot only can move to an adjacent office. Then we have **two preconditions** for the action **move**, which can be expressed in terms of the predicates we have already defined:

Description	Predicate
"To move from office A to B, the robot needs to be inside office A"	*Robot-location(of1)*
"The robot only can move to an adjacent office"	*Adjacent(of1, of2)*

Table 1.1: Predicates representing **preconditions** for the robot to move

By moving, from one office to another, there will be a change in the environment. The robot now will be in the office it moved to, and will no longer be in the room it moved from. We have **two effects** that also can be expressed in terms of the predicates:

Description	Predicate
"The robot will be in the office it moved to"	*Robot-location(of2)*
"The robot will no longer be in the room it moved from"	N*ot(Robot-location(of1))*

Table 1.2: Predicates representing **effects** for the robot after it moves

1.3 | PDDL Implementation

The implementation in PDDL, a specific language for task planning in artificial intelligence, includes 2 main parts: the domain and the problem definition. In the problem definition we have created different scenarios of increasing complexity so one file is used for each tested case.

1.3.1 | Domain

In the domain, we include **requirements** that help to indicate to the solver different built-in functionalities that it can use to find a solution. Also defined in this part are the **predicates**, these indicate the facts that describe the conditions of the environment along the process of finding a solution, the predicates are usually properties of the objects involved in the problem and receive parameters to indicate the concrete object or the state of the property. Finally the set of **actions** is added, an action can receive *parameters*, which are used to validate some predicates defined in its *preconditions* subsection, the last part of an action is the *effect*, in which the results of the environment after applying that action are defined, this mostly related to the defined preconditions and the parameters passed to it. For our implementation the declared actions are:

- **clean-office** receives the office ID as a parameter; as preconditions the robot must be in that office to be able to clean it and the office must be dirty; the result is that the office is no longer dirty but clean.
- **move** receives the IDs of two different offices as parameters, one is the origin and the other one the destination, as preconditions the origin must be adjacent to the destination and the robot must be located in the origin office, the effect produced is that the destination office is no the new location of the robot.

- **push** receives as parameters the IDs of a box and two offices, as preconditions the destination office must be adjacent to the origin office and be empty, also the robot and the box must be in the origin office, as a result the box ends up in the destination office which is also the new location of the robot, the origin office acquires the property of 'empty' while the destination office loses that property.

The code for our domain can be found in the domain.pddl file in our repository, it contains comments that explain in some specific steps the logic behind our statements more specifically

1.3.2 | Problem

For the problem definition, each case has its own file where in the **objects** section we set the existing objects for each own case as well as the initial properties of the objects and their relationships in the **init** section. Finally, in the **goal** section we declare all the predicates we need to represent the final conditions in our environment, i.e. the final states of the objects in it.

An important consideration during the definition of the problem so that the solver is able to understand the rules of the environment, is that the definitions of the states of the objects must be made complete and explicit, even if it seems that things are being left out and that you are being redundant, because if they are not established in this way the solver may not comply with the rules of the environment or may not even find a solution at all. An example of this is the definition of the interoffice adiaciencies in the problem under attack.

The files describing each of the cases we have tested can be found in our repository with the prefix *problem* and a number reflecting their position according to the complexity of the problem.

1.4 | Testing cases, results and analysis

Six case studies were applied, and the concept of each "problem" to be solved in all of them is different, so the conditions are also different. We limit ourselves to classifying them by the size of the grid representing the offices, however, we do not fully determine their level of complexity by this characteristic. The first two problems have a 2×2 grid, while the next two have a 3×3 grid, the last 2 problems are represented with a 4×4 grid. A different combination in the amounts of boxes, dirty rooms and initial locations for all objects are represented in the problems, but probably, more importantly, **the concept we try to prove in each of them is different**, in some of them we leave some definitions unclear **to understand the sensitivity of the solver to reach goals with less or more details, either in the goal itself, in the initial conditions, or even in the definition of the actions**.

1.4.1 | Test Case: 0

1.4.1.1 | Problem Definition

This is probably the easiest of the problems we have defined, where the robot is on a 2×2 floor with 4 offices 1 of which is dirty and 3 boxes. The initial and final distribution of objects is as shown in the code below, as well as in Figure [1.1].

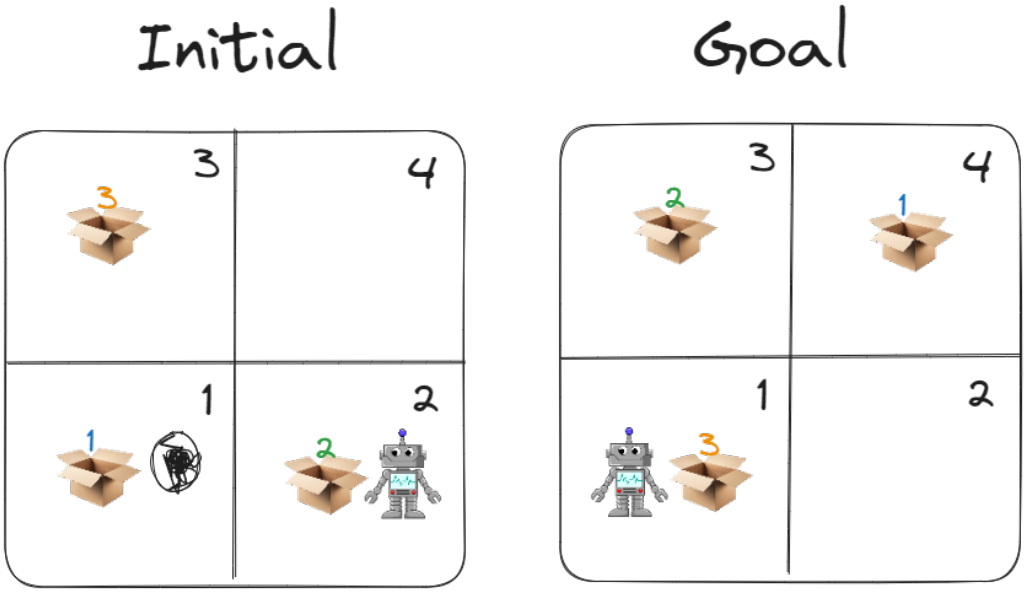

Figure 1.1: Easy problem with 3 boxes, one dirty room and the robot initially at office 2.

```
(define (problem cleaning2x2) (:domain office-robot)
(:objects of1 of2 of3 of4 b1 b2 b3)
(:init
      (robot-location of2)
      (box-location b1 of1)(box-location b2 of2)(box-location b3 of3)
      (dirty of1)
      (empty of4)
      (adjacent of1 of2)(adjacent of1 of3)
      (adjacent of2 of1)(adjacent of2 of4)
      (adjacent of3 of1)(adjacent of3 of4)
      (adjacent of4 of2)(adjacent of4 of3)
)
(:goal (and
       (robot-location of1)
       (box-location b1 of4)(box-location b2 of3)(box-location b3 of1)
       (clean of1)
)))
```

1.4.1.2 | Solver results

- Match tree built with 33 nodes.
- Landmarks found: 5
- Plan found with cost: 18
- Nodes generated: 85
- Nodes expanded: 58
- 1 plan found in 0.306 secs.

After running the solver we found the plan and tested it using the drawings presented in Figure [1.1] by checking both the steps and the trajectory of the robot, the resulting plan (Figure [1.2]) successfully completes the task according to the defined goal. In a nutshell, the robot moves to the dirty office, cleans it, and then moves to the second office to start opening space to accommodate the boxes in a circular motion.

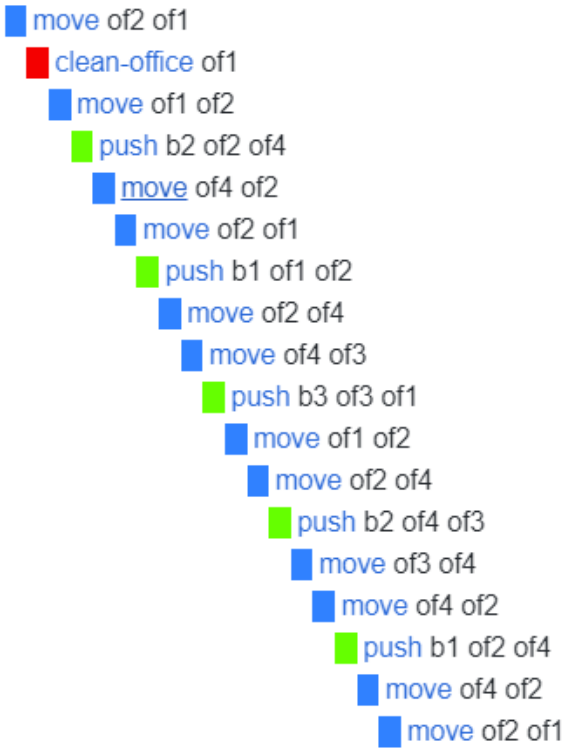

Figure 1.2: Plan found steps (TC0)

The interesting thing about this solution is that, although the circular trajectory is efficient, the robot being initially in the $of2$, where it starts pushing boxes, it does not do it as a first step to then keep moving boxes until it passes through the dirty room to clean it, this could save at least two steps to be performed in the overall plan. Although the environment can be considered simple, and the number of generated and expanded nodes is not too large compared to others, the cost of the found plan (steps) is not too low and this is probably due to the limitation of possible empty spaces to be able to put a box in each step.

1.4.2 | Test Case: Impossible problem

1.4.2.1 | Problem Definition

In this second experiment, and by observing test case 0, we wondered what would happen by giving the planner an impossible problem to solve according to the constraints.

By having the same 2 by 2 matrix and 3 boxes, it is impossible to revert the order in which the boxes are given. For instance, we wanted to review if this will output an actual error.

Then the initial and goal states were defined as stated in the Figure [1.3].

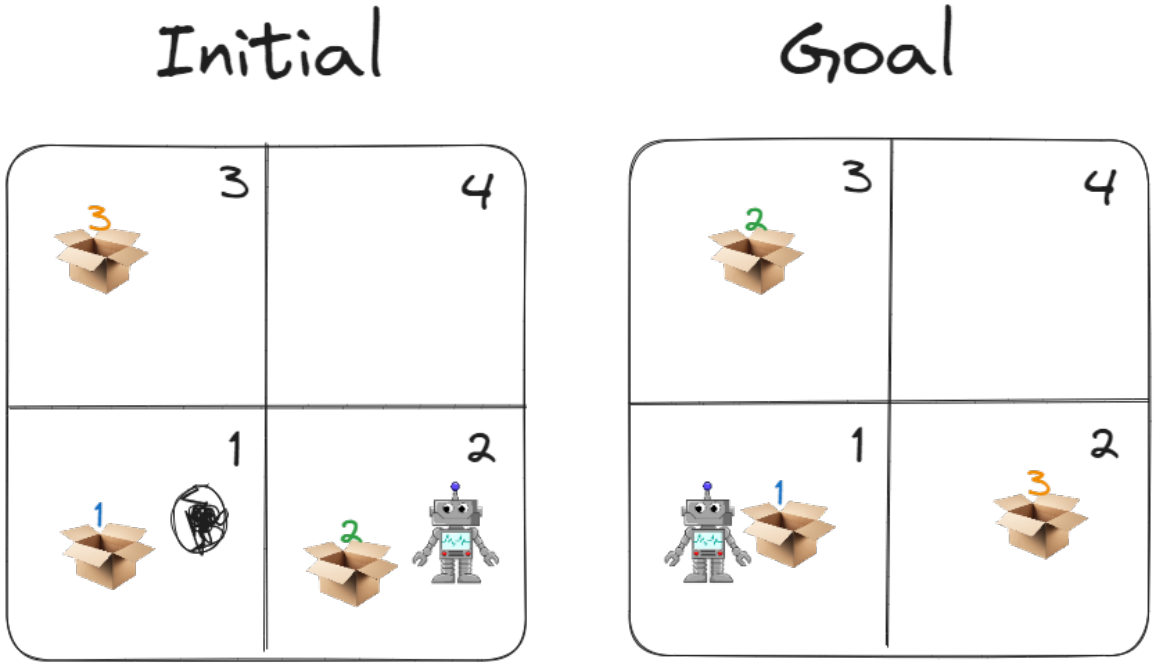

Figure 1.3: Impossible problem with 3 boxes. Main task to change the order of box 2 and box 3.

As seen and previously described, it is impossible to change the order of boxes 2 and 3, since there is only one empty office, there is no center and the movements are restricted to be only vertical and horizontal.

```
(define (problem cleaning2x2_1) (:domain office-robot)
(:objects of1 of2 of3 of4 b1 b2 b3)
(:init
    (robot-location of2)
       (box-location b1 of1)(box-location b2 of2)(box-location b3 of3)
       (dirty of1)
       (empty of4)
       (adjacent of1 of2)(adjacent of1 of3)
       (adjacent of2 of4)(adjacent of2 of1)
       (adjacent of3 of1)(adjacent of3 of4)
       (adjacent of4 of2)(adjacent of4 of3)
)
(:goal (and
 (robot-location of1)
```

```
15 (box-location b1 of1)(box-location b2 of3)(box-location b3 of2)
16 (clean of1)  ))))
```

1.4.2.2 | Solver results

- Match tree built with 33 nodes.
- Landmarks found: 4
- Plan found with cost: 14
- Nodes generated: 87
- Nodes expanded: 61
- 1 plan found in 0.326 secs.

However, when computing the solution, it did not output an error. It outputted an actual plan, that of course it did not solved the problem, but it tried to. We can see in the green steps that represents the push actions in the Figure [1.4] how it tried for each of the boxes to reach the goal state, and at some point the boxes were in the goal, but not simultaneously.

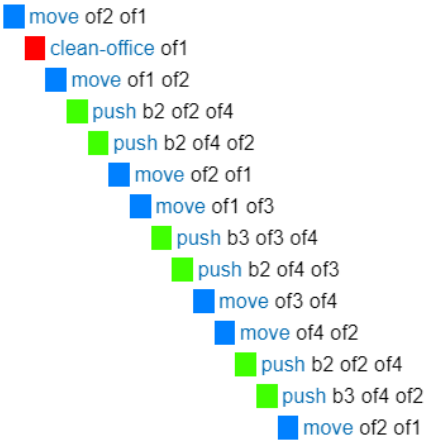

Figure 1.4: Plan found steps (Impossible problem)

We find this experiment very interesting and we tried to understand why instead of outputting an error it gave us a plan that did not solve the problem. So we tried adding an additional effect inside the push action in the domain, that was redundant (or at least from our point of view), and we verified that the problem this time did not outputted any plan which was correct.

The specific predicate we added was:

```
1 (:
2  not(box-location ?box ?office1)
3 )
```

With the solver indicating that no plan was found we realized that adding the above showed effect on the push action we gave more information to the solver so that it determined from the beginning that this was an impossible task.

1.4.3 | Test case: 1

1.4.3.1 | Problem Definition

In the third experiment, we attempted to solve the problem stated in the lab. The Figure [1.5] describes the initial and goal states given.

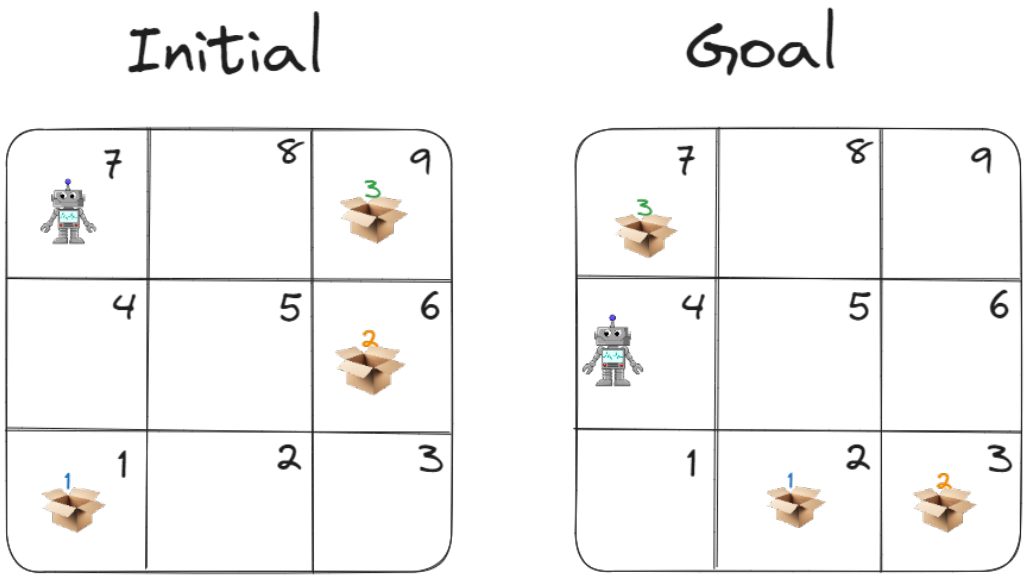

Figure 1.5: Task 1. Problem with 3 boxes and no dirt. Goal: relocate the boxes.

In this case, We have a 3 by 3 matrix of offices, so we needed to establish the objects and predicates that will describe this environment. The complexity here was increased by adding 5 more offices. However, the predicates were the same, and we only needed to define the additional relationships between the objects, but using the same predicates. It was straightforward to add the objects like the new offices.

However, what changed the most, were the relationships between offices: the adjacent predicates.

Since we realized from previous experiments, that is not enough to define the adjacent predicate between of1 and of2 as: *adjacent(of1, of2)* , but instead, we needed to also add the reciprocal of it: *adjacent(of2, of1)* we needed to define 24 adjacent predicates, in order to have the space well-defined and without possible errors.

```
1 (define (problem cleaning3x3_1) (:domain office-robot)
2 (:objects of1 of2 of3 of4 of5 of6 of7 of8 of9 A B C)
3 (:init
4         (robot-location of7)
```

```
        (box-location A of1)(box-location B of6)(box-location C of9)
        (empty of2) (empty of3) (empty of4) (empty of5) (empty of7)(empty of8)
        (adjacent of1 of2)(adjacent of1 of4)
        (adjacent of2 of1)(adjacent of2 of5)(adjacent of2 of3)
        (adjacent of3 of2)(adjacent of3 of6)
        (adjacent of4 of1)(adjacent of4 of5)(adjacent of4 of7)
        (adjacent of5 of4)(adjacent of5 of8)(adjacent of5 of6)(adjacent of5 of2)
        (adjacent of6 of3)(adjacent of6 of5)(adjacent of6 of9)
        (adjacent of7 of4)(adjacent of7 of8)
        (adjacent of8 of7)(adjacent of8 of5)(adjacent of8 of9)
        (adjacent of9 of8)(adjacent of9 of6)
)
(:goal (and
(robot-location of4)
(box-location A of2)(box-location B of3)(box-location C of7)
(empty of5) ;These predicate was added because without it, the final position of the
;box A was he of5 even when it was explicitly declared as of2 in the goal
)))
```

1.4.3.2 | Solver results

- Match tree built with 96 nodes.
- Landmarks found: 5
- Plan found with cost: -505
- Nodes generated: 126
- Nodes expanded: 51
- 1 plan found in 0.46 secs.

After running the solver we found the plan and tested it using the drawings presented in Figure [1.5] by checking both the steps and the trajectory of the robot, the resulting plan (Figure [1.6]) successfully completes the task according to the defined goal.

In this case, there were no complications and basically the planner performed as expected, moving each of the 3 boxes to the desired position.

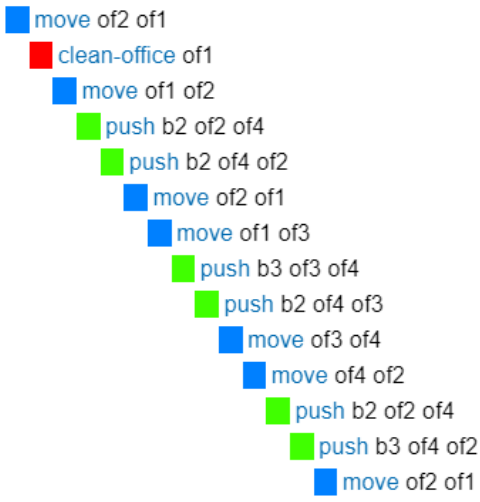

Figure 1.6: Plan found steps (TC1)

1.4.4 | Test case: 2

1.4.4.1 | Problem Definition

In this problem, although there seems to be no obvious complexities to reach the goal from the initial state, a significant number of steps must be taken to solve the task. The environment is set up with a 3^2 grid, there are 4 randomly placed boxes and 3 dirty rooms, including the one in which the robot is in the initial position. In the objective the robot must be placed in the $of3$, there must be a box in each of the corners of the grid and all the rooms must be clean, this is shown graphically in Figure [1.7].

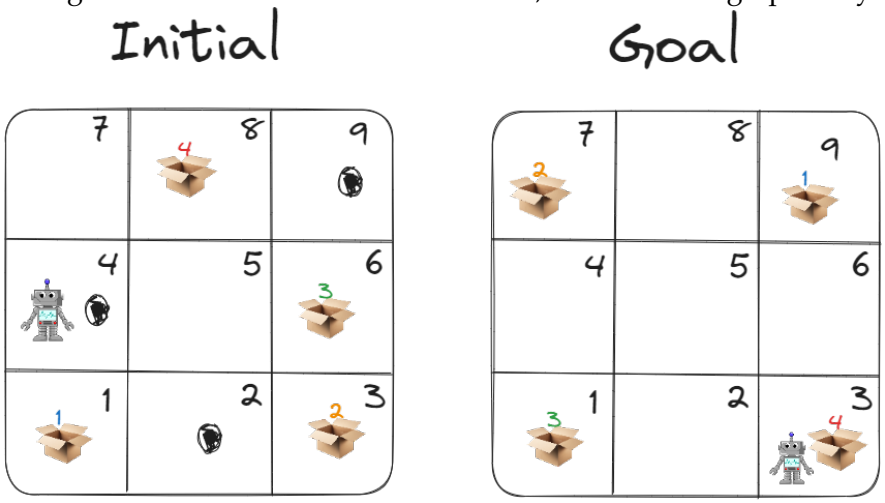

Figure 1.7: Medium level problem with 4 boxes, 3 dirty rooms in a grid of 3×3.

```
(define (problem cleaning3x3_2) (:domain office-robot)
(:objects of1 of2 of3 of4 of5 of6 of7 of8 of9 b1 b2 b3 b4)
(:init
(robot-location of4)
(box-location b1 of1)(box-location b2 of3)(box-location b3 of6)(box-location b4 of8)
(dirty of2)(dirty of4)(dirty of9)
(empty of2)(empty of4)(empty of5)(empty of7)(empty of9)
(adjacent of1 of2)(adjacent of1 of4)
(adjacent of2 of1)(adjacent of2 of5)(adjacent of2 of3)
(adjacent of3 of2)(adjacent of3 of6)
(adjacent of4 of1)(adjacent of4 of5)(adjacent of4 of7)
(adjacent of5 of4)(adjacent of5 of8)(adjacent of5 of6)(adjacent of5 of2)
(adjacent of6 of3)(adjacent of6 of5)(adjacent of6 of9)
(adjacent of7 of4)(adjacent of7 of8)
(adjacent of8 of7)(adjacent of8 of5)(adjacent of8 of9)
(adjacent of9 of8)(adjacent of9 of6))
(:goal (and
(robot-location of3)
(box-location b1 of9)(box-location b2 of7)(box-location b3 of1)(box-location b4 of3)
(clean of2) (clean of4) (clean of9))))
```

.

1.4.4.2 | Solver results

- Match tree built with 123 nodes
- Landmarks found: 8
- Plan found with cost: 30
- Nodes generated: 547
- Nodes expanded: 299
- 1 plan found in 0.413 secs

With the analysis of the obtained solution plan, shown in Figure [1.8], we checked that it was a correct solution, and indeed following the steps completes the task successfully. In summary, at first the robot cleans the room where it is located since it is dirty, then it moves to the nearest dirty room and cleans it, the robot repeats the pattern for the last dirty room, after that the robot moves to where $box2$ is located and pushes it to its target office, and then repeats the pattern for the other 3 boxes with this order $box4 \rightarrow box1 \rightarrow box3$.

Figure 1.8: Plan found steps (TC2)

The time in which the plan was found, the number of nodes generated and expanded seems consistent with the complexity of the problem. The number of critical points (landmarks) to find the solution seems proportional to the number of steps found, and as initially hypothesized, although there are no particular complexities for the solution of the problem, the number of actions to be performed is considerable. Analyzing the trajectory followed by the robot, some points that could optimize the work of the robot are identified. According to the steps generated by the solver, it seems that the type of activity to be performed is grouped in stages, i.e. first it cleans all the dirty rooms, then it places the boxes in their final positions one by one and finally it moves to their final position. However, at the moment of moving to clean the rooms the robot could push some of the boxes already being in that room to save future movements.

1.4.5 | Test case: 3

1.4.5.1 | Problem Definition

In the fifth experiment, we increased the difficulty of the problem by *n=4* At the same time, we wondered if the solver would perform correctly by not setting the positions of the boxes in the goal state, not giving the robot a final state, but instead only setting the empty offices desired and the offices that needed to be cleaned.

So, we defined the initial and goal states as shown in Figure [1.9]

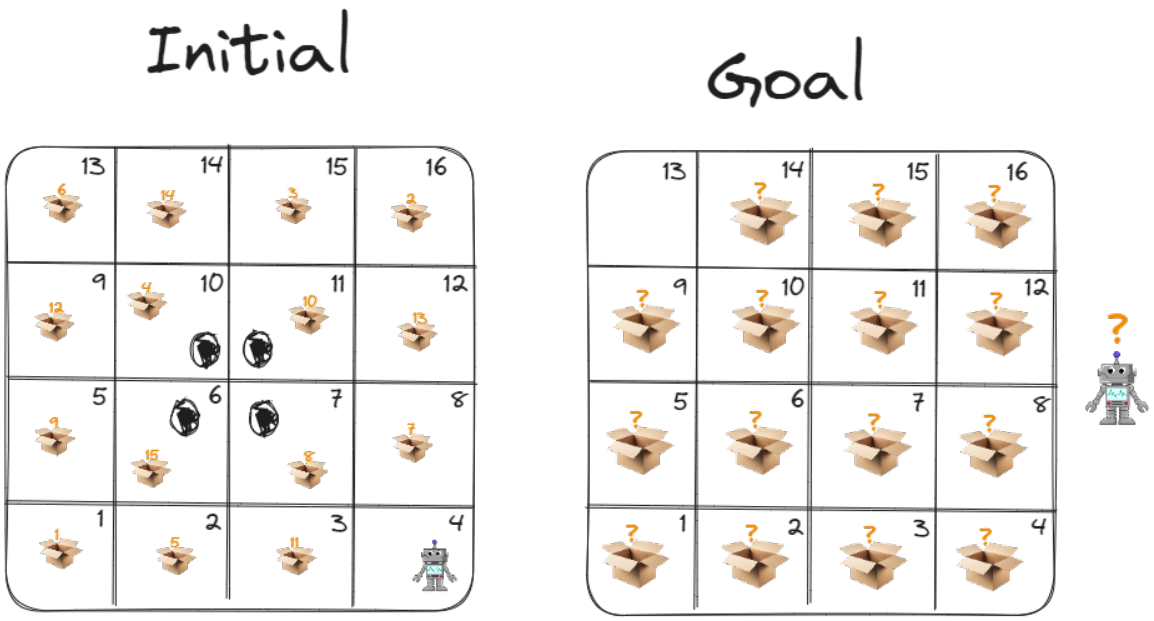

Figure 1.9: Task 3. Problem with 15 boxes, dirt. Goal: relocate the boxes.

In this case, We have a 4 by 4 matrix of offices, so we needed to restablish the objects and predicates that will describe this environment, as we did in the transition from 2 by 2 to 3 by 3. We can see the implementation of PDDL code below.

```
(define (problem cleaning4x4_1) (:domain office-robot)
(:objects of1 of2 of3 of4 of5 of6 of7 of8 of9 of10 of11 of12 of13 of14 of15 of16 b1
    b2 b3 b4 b5 b6 b7 b8 b9 b10 b11 b12 b13 b14 b15)

(:init
        (robot-location of4)
        (box-location b1 of1) (box-location b2 of16)(box-location b3 of15)(
           box-location b4 of10)(box-location b5 of2)(box-location b6 of13)(
           box-location b7 of8)(box-location b8 of7)
        (box-location b9 of5) (box-location b10 of11)(box-location b11 of3)(
           box-location b12 of9)(box-location b13 of12)(box-location b14 of14)(
           box-location b15 of6)
        (dirty of6)(dirty of7)(dirty of10)(dirty of11)
        (empty of4)
        (adjacent of1 of2)(adjacent of1 of5)
        (adjacent of2 of1)(adjacent of2 of3)(adjacent of2 of6)
        (adjacent of3 of2)(adjacent of3 of4)(adjacent of3 of7)
        (adjacent of4 of3)(adjacent of4 of8)
        (adjacent of5 of6)(adjacent of5 of9)(adjacent of5 of1)
        (adjacent of6 of5)(adjacent of6 of7)(adjacent of6 of10)(adjacent of6 of2)
        (adjacent of7 of6)(adjacent of7 of8)(adjacent of7 of11)(adjacent of7 of3)
        (adjacent of8 of7)(adjacent of8 of12)(adjacent of8 of4)
        (adjacent of9 of10)(adjacent of9 of13)(adjacent of9 of5)
        (adjacent of10 of9)(adjacent of10 of11)(adjacent of10 of14)(adjacent of10
           of6)
        (adjacent of11 of10)(adjacent of11 of12)(adjacent of11 of15)(adjacent of11
           of7)
        (adjacent of13 of9)(adjacent of13 of14)
        (adjacent of14 of13)(adjacent of14 of15)(adjacent of14 of10)
        (adjacent of15 of14)(adjacent of15 of16)(adjacent of15 of11)
        (adjacent of16 of15)(adjacent of16 of12)
)
(:goal (and
 (empty of13)
 (clean of6)
 (clean of7)
 (clean of10)
 (clean of11)
))))
```

1.4.5.2 | Solver results

- Match tree built with 637 nodes.
- Landmarks found: 5
- Plan found with cost: 27
- Nodes generated: 288
- Nodes expanded: 253
- 1 plan found in 0.305 secs.

By testing the plan given by the solver using the drawings presented in Figure [1.9] we were able to check and analyze the efficiency of both the steps and the trajectory the robot followed to achieve the goal, the resulting plan (Figure [1.10]) completes the task according to the defined goal.

It was very interesting to note that even if we did not define the position of the boxes as goal states, but only the office that needed to be empty, it was enough for the planner to find a solution.
Also, since we did not define the goal position of the robot, it remained in the same position that it needed to be in order to complete the last goal.

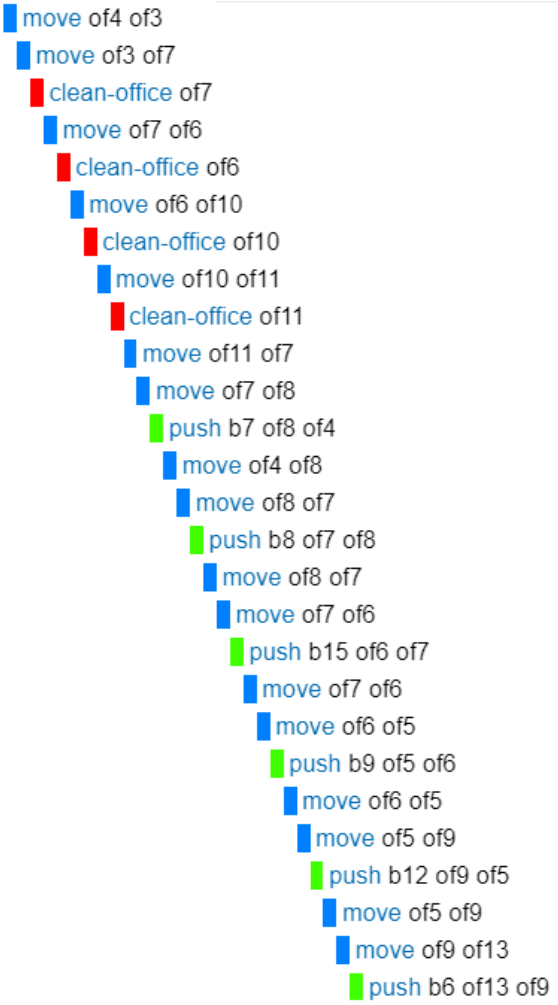

Figure 1.10: Plan found steps (TC3)

1.4.6 | Test case: 4

1.4.6.1 | Problem Definition

In this last test case, as shown inFigure [1.11], we have an environment defined by a 4×4 grid, all the odd-numbered offices are dirty and there are three boxes that must be placed in explicitly indicated locations that are different from the origin position.

```
1 (define (problem cleaning4x4_2) (:domain office-robot)
2 (:objects of1 of2 of3 of4 of5 of6 of7 of8 of9 of10 of11 of12 of13 of14 of15 of16 b1
     b2 b3)
3 (:init
4     (robot-location of2)
5     (box-location b1 of1) (box-location b2 of10)(box-location b3 of15)
```

```
    (dirty of1)(dirty of3)(dirty of5)(dirty of7)
    (dirty of9)(dirty of11)(dirty of13)(dirty of15)
    (empty of2)(empty of3)(empty of4)(empty of5)(empty of6)(empty of7)(empty of8)
    (empty of9)(empty of11)(empty of12)(empty of13)(empty of14)(empty of16)
    (adjacent of1 of2)(adjacent of1 of5)
    (adjacent of2 of1)(adjacent of2 of3)(adjacent of2 of6)
    (adjacent of3 of2)(adjacent of3 of4)(adjacent of3 of7)
    (adjacent of4 of3)(adjacent of4 of8)
    (adjacent of5 of6)(adjacent of5 of9)(adjacent of5 of1)
    (adjacent of6 of5)(adjacent of6 of7)(adjacent of6 of10)(adjacent of6 of2)
    (adjacent of7 of6)(adjacent of7 of8)(adjacent of7 of11)(adjacent of7 of3)
    (adjacent of8 of7)(adjacent of8 of12)(adjacent of8 of4)
    (adjacent of9 of10)(adjacent of9 of13)(adjacent of9 of5)
    (adjacent of10 of9)(adjacent of10 of11)(adjacent of10 of14)(adjacent of10 of6)
    (adjacent of11 of10)(adjacent of11 of12)(adjacent of11 of15)(adjacent of11 of7)
    (adjacent of13 of9)(adjacent of13 of14)
    (adjacent of14 of13)(adjacent of14 of15)(adjacent of14 of10)
    (adjacent of15 of14)(adjacent of15 of16)(adjacent of15 of11)
    (adjacent of16 of15)(adjacent of16 of12)
)
(:goal (and
 (robot-location of16)
 (box-location b1 of5)(box-location b2 of1)(box-location b3 of7)
 (clean of1)(clean of3)(clean of5)(clean of7)
 (clean of9)(clean of11)(clean of13)(clean of15)
)))
```

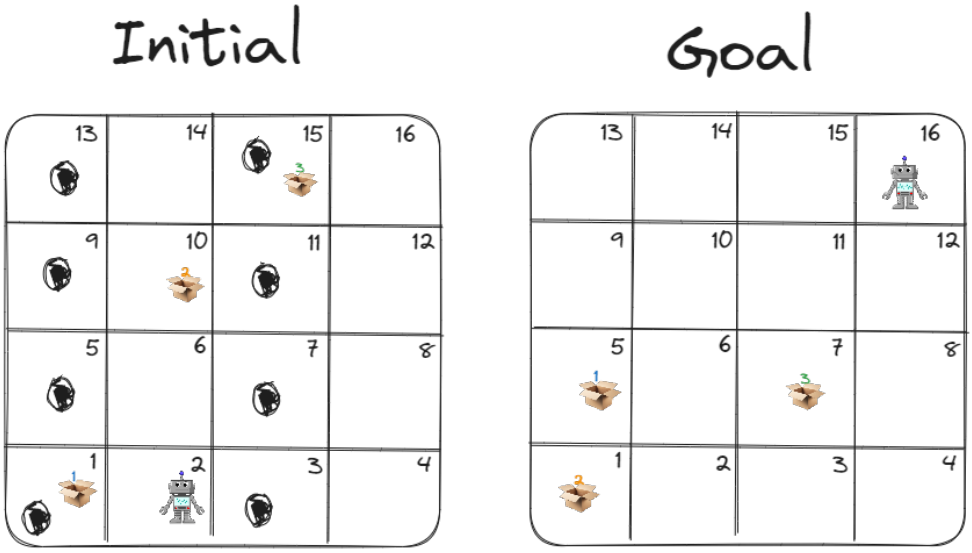

Figure 1.11: Easy problem with 3 boxes, one dirty room and the robot initially at office 2.

.

1.4.6.2 | Solver results

- Match tree built with 188 nodes
- Landmarks found: 12
- Plan found with cost: 35
- Nodes generated: 657
- Nodes expanded: 352
- 1 plan found in 0.775 secs.

After obtaining the problem solution, shown in Figure [1.12], it was confirmed that it is a valid solution, indeed the goal is reached through the indicated steps.

Analyzing the solution plan, both the steps and the trajectory of the robot, it can be identified at the beginning of it a pattern of action different from those shown in solutions to previous problems, because although the general sets of actions are grouped, and in the order of action the robot always cleans the dirty rooms first, this time the robot takes advantage of having to go through an office where it only has to make a push to put one of the boxes in its final position and executes it. Showing that possibly in previous problems it did not do that behavior, since it deviated it from the route that it takes to fulfill the group of goals that it desires. However, at this same point, the office from which he pushes the box is also dirty, but he does not approach to clean it, which implies extra steps later to do so. This gives some indication that there may be optimization opportunities for the plan generated. In the rest of the steps the plan follows general patterns shown in solutions to other problems already discussed.

move of2 of1
push b1 of1 of5
clean-office of5
move of5 of9
clean-office of9
move of9 of13
clean-office of13
move of13 of14
move of14 of15
clean-office of15
move of15 of11
clean-office of11
move of11 of7
clean-office of7
move of7 of3
clean-office of3
move of3 of2
move of2 of1
clean-office of1
move of1 of2
move of2 of6
move of6 of10
push b2 of10 of6
push b2 of6 of2
push b2 of2 of1
move of1 of2
move of2 of3
move of3 of7
move of7 of11
move of11 of15
push b3 of15 of11
push b3 of11 of7
move of7 of11
move of11 of15
move of15 of16

Figure 1.12: Plan found steps (TC4)

In addition, although the cost of the plan is not too high even when the grid is larger, the amount of nodes generated, expanded and used in the tree is much higher than in environments with smaller grids.

1.5 | Conclusions

In this assignment, we implemented a domain and multiple test cases for the Cleaner Robotic Task using PDDL. The task involved a grid environment with offices that could contain boxes and dirt. The goal was to clean the offices, move the boxes to designated locations, and place the robot in a specific final position. To do this PDDL solver helped us to apply the STRIPS strategy to find a plan for the world rules and each problem conditions.

We defined the PDDL domain, which included actions, predicates, and requirements. Implementing this domain required careful consideration of the complexities of the problem and a clear representation of the problem space. We created six distinct test cases to evaluate our PDDL implementation. These test cases progressively increased in complexity, providing a comprehensive assessment of our system's capabilities.

Our tests revealed several key insights into the performance of our implementation. We observed some of the considerations that must be taken into account when choosing the order of tasks, how repetitive clusters of tasks to be performed emerge, and how the involvement of the objects in each task and their relationship to the final state also influence the solution approach, which helped us to understand the strengths and weaknesses of our implementation.

Through this task, we learned important aspects about domain and problem definition using PDDL. It is important to take into account the level of specificity used in the preconditions and effects of the actions, as well as in the description of the problem, including the objects, the initial state, and the final state, because if not defined correctly, some rules may be violated, or a solution may not be found by the solver.

Although our implementation of PDDL was successful in solving the Robotic Cleaning Task on all problems with a possible solution, there is room for improvement and further exploration. Some areas for future work involve trying to include more predicates, more actions, or more advanced logics for robot navigation, which may make it possible to facilitate the definition of problem conditions or optimize the steps necessary to reach the goal.

In conclusion, this assignment allowed us to gain hands-on experience in applying PDDL to solve a complex planning problem. We successfully implemented the Cleaner Robotic Task domain and tested it across six progressively challenging scenarios. Our findings and experiences will contribute to our understanding of planning and approximate reasoning in this class.