Universitat Rovira i Virgili

Planning and Aproximate Reasoning

M.Sc. in Computer Engineering: Computer Security and Intelligent Systems

# Practical Exercise 1: Planner: The coffee server

*Authors:*
Mónica del Carmen Muñoz
Pulak Roy

# Contents

**1 Introduction 2**

**2 Analysis of the problem 3**

**3 Planning Algorithm and Heuristics 4**

3.1 Heuristic 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

3.2 Heuristic 2 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

**4 Implementation design 6**

4.1 Methods . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

4.2 Entry File . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

**5 Testing cases and results 8**

5.1 Case 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

5.2 Case 2 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

5.3 Case 3 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

5.4 Comparison test cases . . . . . . . . . . . . . . . . . . . . . . . . . 10

**6 Intructions to execute the program 11**

**Appendices 12**

## 1 Introduction

Intelligent strategies are compulsory for solving problems arising in our daily life. Knowledge based intelligent strategies are prerequisite to solve such problems efficiently and to achieve the desired result. Planning refers to a sequence of actions that lead from the initial state to the goal state.

Problem solving and planning are deeply related to Artificial Intelligence. For instance in the fields of game playing, robots utilize problem solving and planning techniques to accomplish their tasks.

The Standford Research Institute Problem Solver **(STRIPS)** is an automated planning technique that works by executing a domain and problem to find a goal. A characteristic of this planning systems is their linear approach, i.e. only one aim is followed and attempted to be solved – interleaving of goals is not allowed. Goals are maintained on a stack and dealt with one by one. Additionally, only totally ordered action sequences are allowed, i.e. the order of actions cannot be changed later on and must be strictly determined once planning is complete. With STRIPS you provide objects, actions, preconditions, and effects. Once the domain is described, you then provide the initial state and final state. Strips can then search all possible states, starting from the initial one, executing various actions, until it reaches the goal.[1]

In this exercise our problem description is like, there is a squared building composed by 36 offices, which are located in a matrix of 6 rows and 6 columns. From each office it is possible to move (horizontally or vertically) to the adjacent offices. The building has some coffee machines in some offices that can make 1, 2 or 3 cups of coffee at one time. The people working at the offices may ask for coffee and a robot called "Clooney" is in charge of serving the coffees required. Each office may ask for 1, 2 or 3 coffees but not more. The petitions of coffee are done all at early morning so that the robot can plan the service procedure. Each petition has to be served in a single service. The goal is to serve all the drinks to all the offices in an efficient way (minimizing the travel inside the building, in order to not disturb the people working). The robot will start with a given initial state. Initial state contains petitions of coffee, positions of the coffee machines and the initial position of the robot. In goal state no more petitions are pending to be served. Following figures depict our problem scenario more clearly.

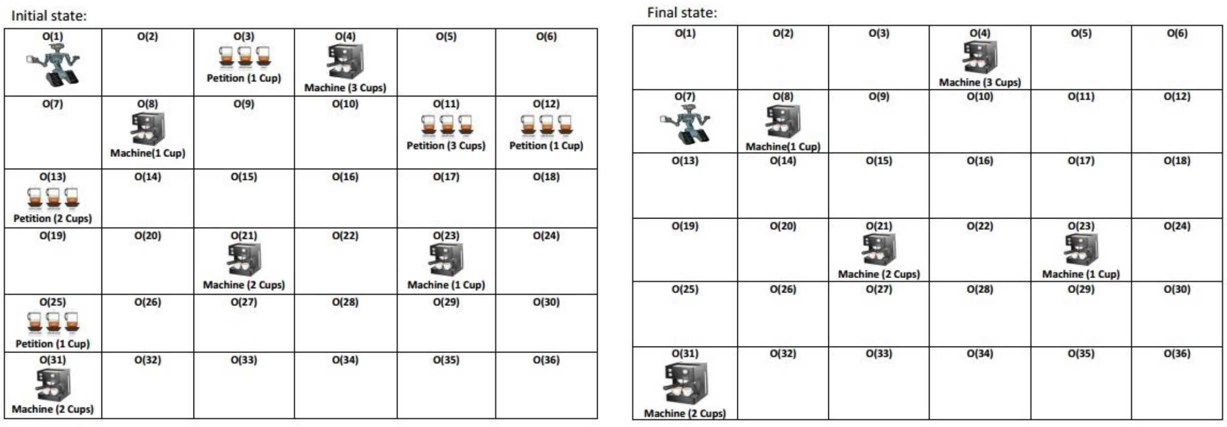


*Figure 1: Problem Example*

## 2 Analysis of the problem

Our problem is represented by a matrix. We have a robot that is in an initial location inside the matrix and need to fulfil all the requests made by the offices. There are some offices that have a coffee machine, which can serve 1, 2 or 3 coffees.

For making our plan we have three types of operators:

- Make(o,n)
- Move(o1, o2)
- Serve(o,n)

The Make(o,n) operator lets the robot make n cups of coffee in the machine located in office o. For applying this operator we need to have some preconditions like the robot need to be located in the office o, the robot need to be free and there must be a machine in the office o that can serve n cups of coffee. When we apply this operator we loose the property robot-free and now our robot is loaded with n cups of coffee.

The Move(o1, o2) operator allows the robot to move between the offices. For applying this operator we need the robot to be in office o1 and the number of steps is x. After this operator is applied the robot is no more in office o1 and the number of steps is not x any more. Now our robot is in location o2 and the number of steps is the previous plus the distance between office o1 and o2.

The Serve(o,n) operator allows the robot to serve the petition of n cups of coffee in the office o. For applying this operator we need the robot to be in office o, we need the robot loaded with n cups and the petition of the office o to be n cups of coffee. When we apply this operator we can say that there is no petition for this office o, and the robot is not loaded with n cups. Now our robot is free and we can say that the office was served.

## 3 Planning Algorithm and Heuristics

The key component of STRIPS algorithm is the Operator which contains the three elements: preconditionList, addList and deleteList. The STRIPS algorithm also requires the list of goals and the initial state of the world as inputs. At the starting point initial state is considered as current state of the planner.

The planner first checks whether the goals are already satisfied by looking at the current state. If some goals are not satisfied, a backwards chaining procedure is applied. Each operator's addList enumerates the goals that it will achieve. By looking through the operator's addLists, the planner finds an operator that will result in the goal being achieved.

Since each operator has a set of preconditions (preconditionList), it may not be possible to execute that action if those predicates are not true in the current state. The preconditions for desired operator become the goalList for another iteration of the planner, and planner attempts to find operators which will satisfy this new list of goals. By iterating backwards through the chain of events, the agent may eventually arrive at a sequence of operators which will satisfy all the intermediate goals and the original goal.

STRIPS planning Operators support predicates which take arguments. These arguments can then be unified with facts in the current state in order to form bindings for the arguments. Following is the pseudocode for STRIPS:

**Algorithm 1** STRIPS algorithm

```
1: currentstate ← $e_i$
2: plan ← null
3: push predicates in $e_f$ to stack
4: push each predicate x in $e_f, \forall_x$
5: loop: $i >$ stack is empty then terminate;
6: unstack top element from stack
7: switch element do
8:     case operator
9:         currentstate - delList - addList; add operator to plan
10:    case listofpredicates
11:        validate all; anyofpredicateisfalse push this predicatelist again; push false predicate;
12:    case singlepredicate
13:        partiallyinstantiated find constant c; x = c in all predicate in stack; totallyinstantiated if true do nothing; if false then searchoperator(),push operator, push operator's preconditionList, push each predicate
14: end loop:
15: return plan
```

### Heuristics

A heuristic, is any approach to problem solving, learning, or discovery that employs a practical method not guaranteed to be optimal or perfect, but sufficient for the immediate goals. The objective of a heuristic is to produce a solution in a reasonable time frame that is good enough for solving the problem at hand. This solution may not be the best of all the actual solutions to this problem, or it may simply approximate the exact solution. But it is still valuable because finding it does not require a prohibitively long time.

### 3.1 Heuristic 1

In our exercise we also implemented heuristic method to improve the performance of our planning technique. Firstly we applied heuristic method to rearrange our final state's predicates, because we need to serve all the petitions by optimizing the number of moves of our robot.

This method takes initial state(String) and final state(String) as input and returns the rearranged final state so that robot can serve all petition with optimal number of moves. Inside that method we have Hashtable<Integer, Integer >coff_machine which stores office number(o) as it's key and machine capacity(n) as it's value. In addition, there is a Hashtable <Integer, Vector<Integer >>petition which has key = number of coffee(n) and value = list of office number(o). By this data structure we will be able to know all the petitions. So that we will be able to know which offices has same number of petition. We take the robot location from the initial state and maintain its position correctly throughout serving all the petitions. Following steps reveal how our heuristic method works:

**Algorithm 2** Heuristic method

```
1: loop:iterate following steps until all petitions has been processed
2: first: find out nearest machine from robot-location.(use manhatan distance)
3: second: get that nearest machine's capacity and then check is there any petition with
   same capacity
4: If there is no such petition then remove these machine from coff_machine
5: If Yes(if there are multiple number of petition with same quantity) then find nearest
   petition
6: add that petition at first position in final state
7: remove that served petition's offce number from petition hashtable
8: change robot location with the value from served petition's office number
9: end loop:
10: return final state
```

This algorithm is implemented inside Strips. The method is named public static String heuristicMethod(String initialState, String goalState).

### 3.2 Heuristic 2

There is another section in our implementation where we give some intelligence to our robot. When STRIPS planner unstack a predicate name robot-loaded(n) then it needs to Make(o,n)(n cups of coffee from office o). In that situation our robot can find out nearest machine location with capacity of n.

This method is implemented in Strips.java, inside public int getKeyOfValue(int value) method. This method takes a integer value as input and returns an integer(nearest machine location). In this procedure we have used Map<Integer, Integer >map_machine_coffe from Strips.java class which can provide us machine location(key) and machine capacity(value). Alongside, we have Vector<Integer>vectOffices which stores office numbers that have machine with same capacity. Following steps depicts the working step of this method:

**Algorithm 3** Heuristic method

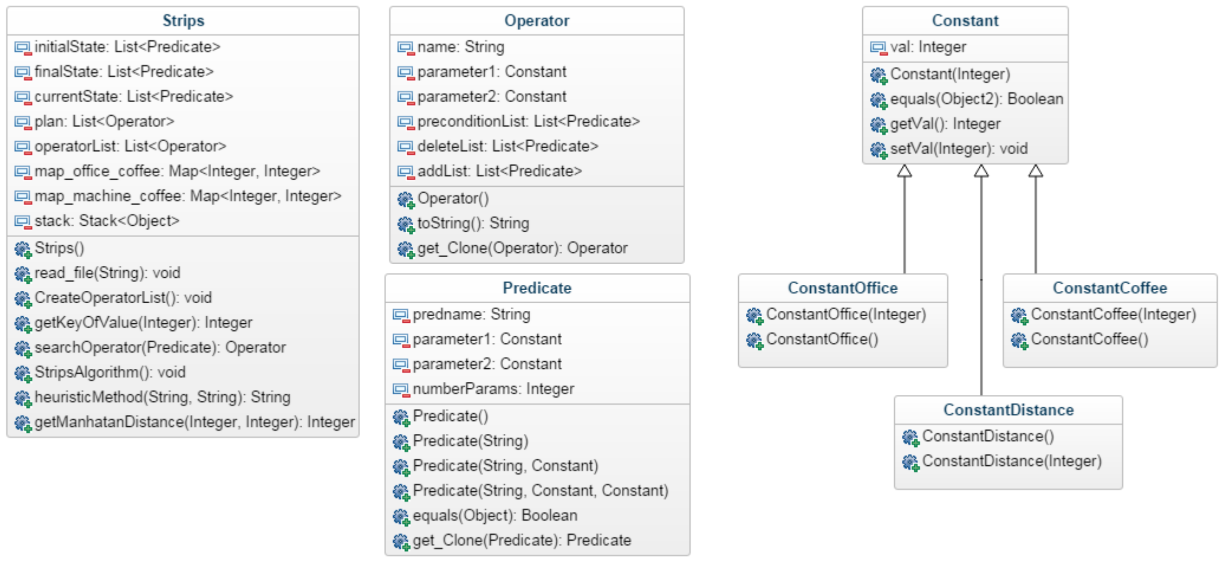
```
1: first: search inside map_machine_coffe to get office location that has machine with
   capacity(input parameter: int value) and add to vector vectOffices
2: second: find current location of robot from current state
3: third: check all elements of vectOffices and calculate nearest office location (use ma-
   hatan distance)
4: return nearest office number with capacity(n)
```

## 4 Implementation design

In our implementation we define the following classes:

- Predicate
- Operator
- Constant (ConstantOffice, ConstantCoffee, ConstantDistance)
- Strips (The main class that implements the algorithm)

The following picture is the class diagram of our project.



Our Strips class at the begging fills the necessary variables to run the algorithm. We have a list of initial predicates, final predicates and we maintain the current state. At the beggining we fill two HashMap, one gives us the petition information (number_office,number_petitions). The other HashMap lets us know the office where the coffe machines are and the number of cups it can serve(number_office, number_cups).

Finally we have an stack of objects which is the structure that helps us in the implementation of the main STRIPS algorithm. Inside of our stack we can have three types of objects (Operators, List of Predicates and Single Predicates). Depending on the pop object we analyze each case.

## 4.1 Methods

In this section we are going to describe the most important methods that we implemented apart of the heuristic methods that were explined before.
First we have the ReadFile method that receives the address of the file where the initial state and final state are described. In this method we parse the file that we called objectives.txt and fill the variables initial state, final state and the hashmaps described before.

In the Strips contructor we also called the CreateOperatorList method. This method fills the List of Operators and initially the predicates of the Add, Delete and Precondition lists are simple constants without value. Later in the algorithm these lists are going to be instanciated with real values of office number and number of cups of coffee.

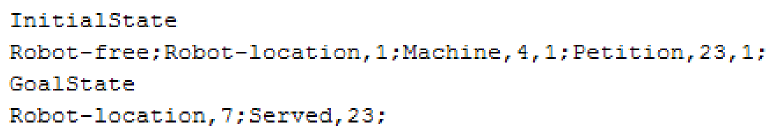
Another interesting method that we define is SearchOperator. This method receive a predicate and search in the List of Operators for an operator that has a predicate with this name. In our specific domain we do not have in the add list a predicate that leads us to more than one operator. This is more simple because we do not need to apply heuristic to choose between operators. In this method we return a fully instanciated operator, this means that all the parameters and the predicates inside the add, delete and precondition list have real values. For doing this we use the current state to know the location of the robot, the hashtables of petitions and the hashtables of the machines with the numbers of cups of coffee that can serve.

The main method is called StripsAlgorithm. Initially we create a list of operators that we call plan to print. While the stack is not empty we pop an element for analyzing. This element can be an Operator, a List of predicates or a single predicate. Depending on the element we pop we do some treatment following the STRIPS algorithm.

## 4.2 Entry File

Our entry file differs from the original proposed in the task. In the first line we place the text InitialState, the second line is the list of predicates separated by semicolon. Inside each predicate we separate the parameters by comma. In the third line the text GoalState and in the fourth line the list of final predicates separated by semicolon and the parameters by comma.

The following image shows an example with the structure defined before:



```
InitialState
Robot-free;Robot-location,1;Machine,4,1;Petition,23,1;
GoalState
Robot-location,7;Served,23;
```

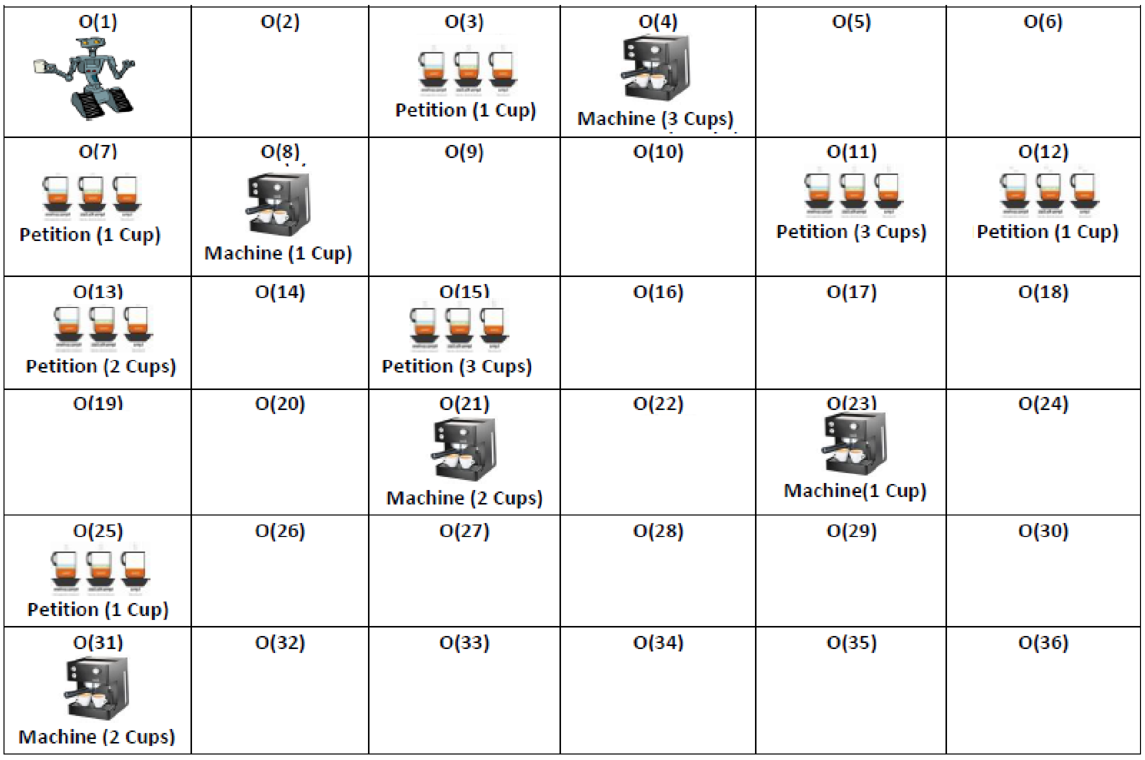
## 5 Testing cases and results

We analize three test cases with different complexity.

### 5.1 Case 1

The first case that we analyze has the following structure and image for the **initial state**.

Robot-free;Robot-location,1;Machine,4,3;Machine,8,1;Machine,21,2;Machine,23,1;
Machine,31,2;Petition,3,1;Petition,7,1;Petition,11,3;Petition,12,1;Petition,13,2;
Petition,15,3;Petition,25,1;



| | | | | | |
|---|---|---|---|---|---|
| O(1) | O(2) | O(3) Petition (1 Cup) | O(4) Machine (3 Cups) | O(5) | O(6) |
| O(7) Petition (1 Cup) | O(8) Machine (1 Cup) | O(9) | O(10) | O(11) Petition (3 Cups) | O(12) Petition (1 Cup) |
| O(13) Petition (2 Cups) | O(14) | O(15) Petition (3 Cups) | O(16) | O(17) | O(18) |
| O(19) | O(20) | O(21) Machine (2 Cups) | O(22) | O(23) Machine(1 Cup) | O(24) |
| O(25) Petition (1 Cup) | O(26) | O(27) | O(28) | O(29) | O(30) |
| O(31) Machine (2 Cups) | O(32) | O(33) | O(34) | O(35) | O(36) |

The **final state** is defined as:

Robot-location,2;Served,3;Served,7;Served,11;Served,12;Served,13;Served,15;Served,25;

After applying the Strips algorithm we obtain the following **plan** :

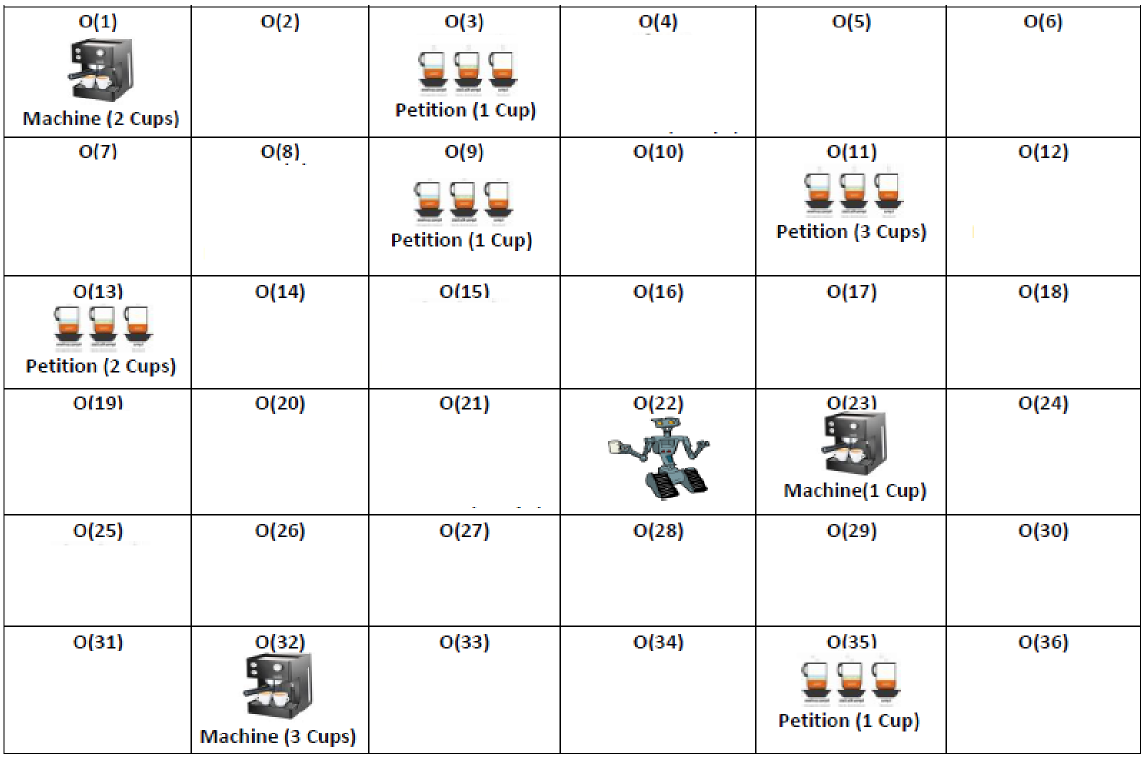
Move(1,8); Make(8,1); Move(8,7); Serve(7,1); Move(7,8); Make(8,1); Move(8,3); Serve(3,1);
Move(3,4); Make(4,3); Move(4,11); Serve(11,3); Move(11,4); Make(4,3); Move(4,15); Serve(15,3);
Move(15,21); Make(21,2); Move(21,13); Serve(13,2); Move(13,8); Make(8,1); Move(8,12);
Serve(12,1); Move(12,23); Make(23,1); Move(23,25); Serve(25,1); Move(25,2);

The number of **steps** travelled by the robot for this case is 37.

## 5.2 Case 2

For our test case 2 the **initial state** is:

Robot-free;Robot-location,22;Machine,1,2;Machine,23,1;Machine,32,3;Petition,3,1;Petition,9,1;
Petition,11,3;Petition,13,2;Petition,35,1;



The **final state** is the following:

Robot-location,2;Served,3;Served,9;Served,11;Served,13;Served,35;

The **plan** obtained is the following:

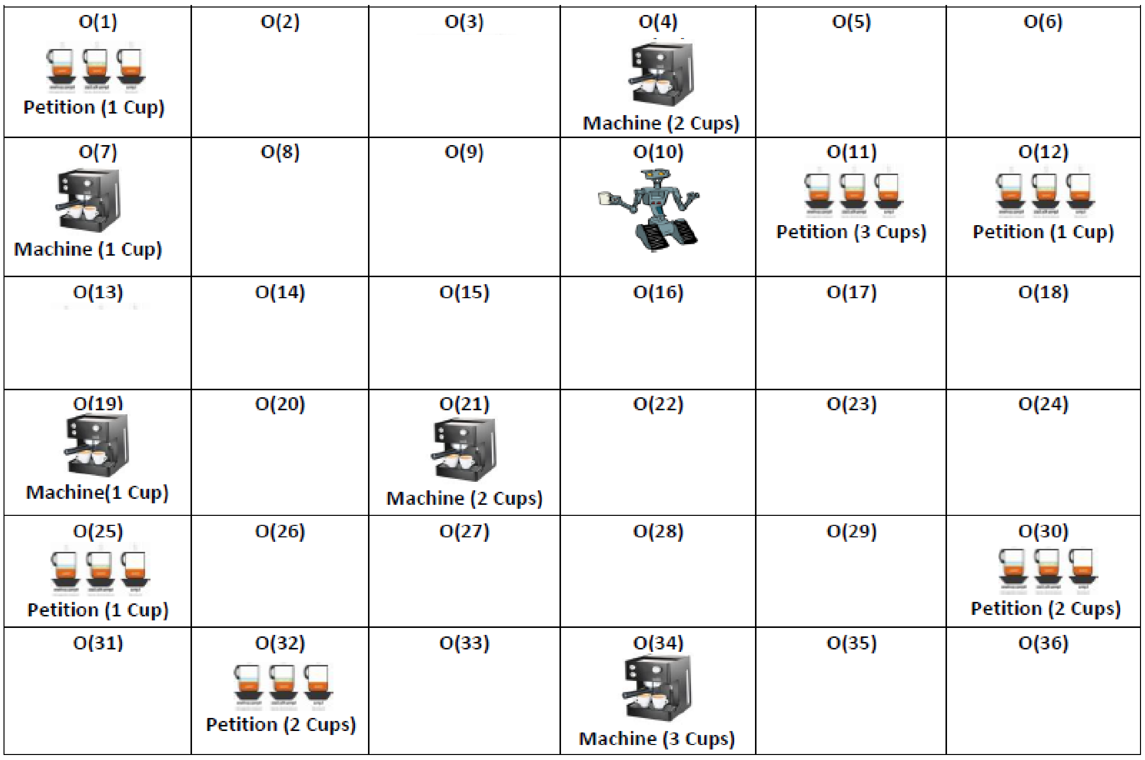
Move(22,23); Make(23,1); Move(23,35); Serve(35,1); Move(35,23); Make(23,1); Move(23,9);
Serve(9,1); Move(9,1); Make(1,2); Move(1,13); Serve(13,2); Move(13,32); Make(32,3);
Move(32,11); Serve(11,3); Move(11,23); Make(23,1); Move(23,3); Serve(3,1); Move(3,2);

The number of **steps** travelled by the robot is 33.

## 5.3 Case 3

The structure of the **initial state** of case 3 is the following:

Robot-free;Robot-location,10;Machine,4,2;Machine,7,1;Machine,19,1;Machine,21,2;
Machine,34,3;Petition,1,1;Petition,11,3;Petition,12,1;Petition,25,1;Petition,30,2;Petition,32,2;



The **final state** is defined as:

Robot-location,10;Served,1;Served,11;Served,12;Served,25;Served,30;Served,32;

For the execution of the algorithm we have a stack. In this stack we push three types of objects : list of predicates, operator and single predicates. For this case we show the stack in the appendix.

After applying the Strips algorithm we obtain the following **plan**:

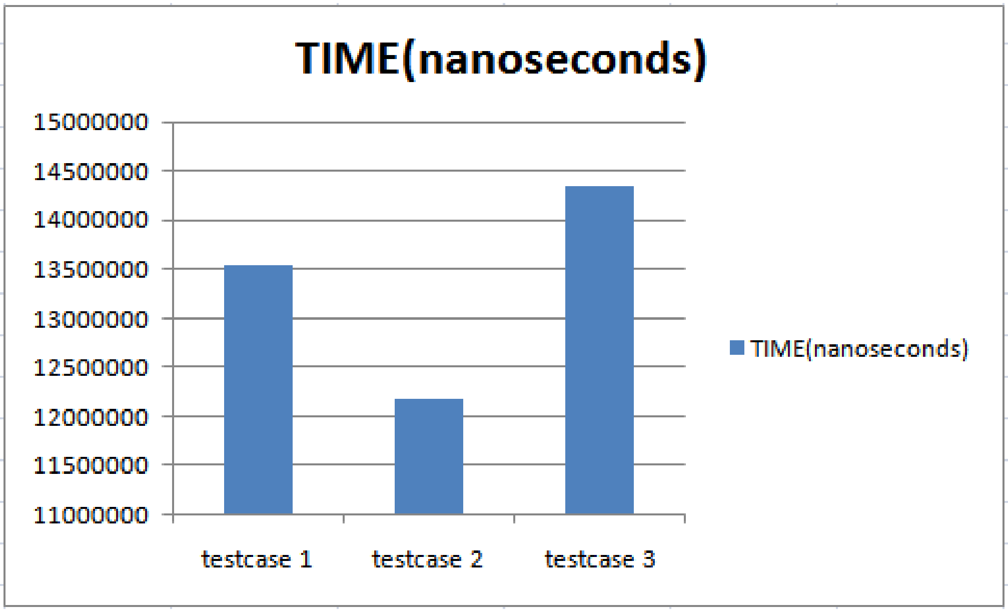
Move(10,4); Make(4,2); Move(4,30); Serve(30,2); Move(30,34); Make(34,3); Move(34,11); Serve(11,3); Move(11,4); Make(4,2); Move(4,32); Serve(32,2); Move(32,19); Make(19,1); Move(19,25); Serve(25,1); Move(25,19); Make(19,1); Move(19,1); Serve(1,1); Move(1,7); Make(7,1); Move(7,12); Serve(12,1); Move(12,10);

The number of **steps** travelled by the robot for this case is 40.

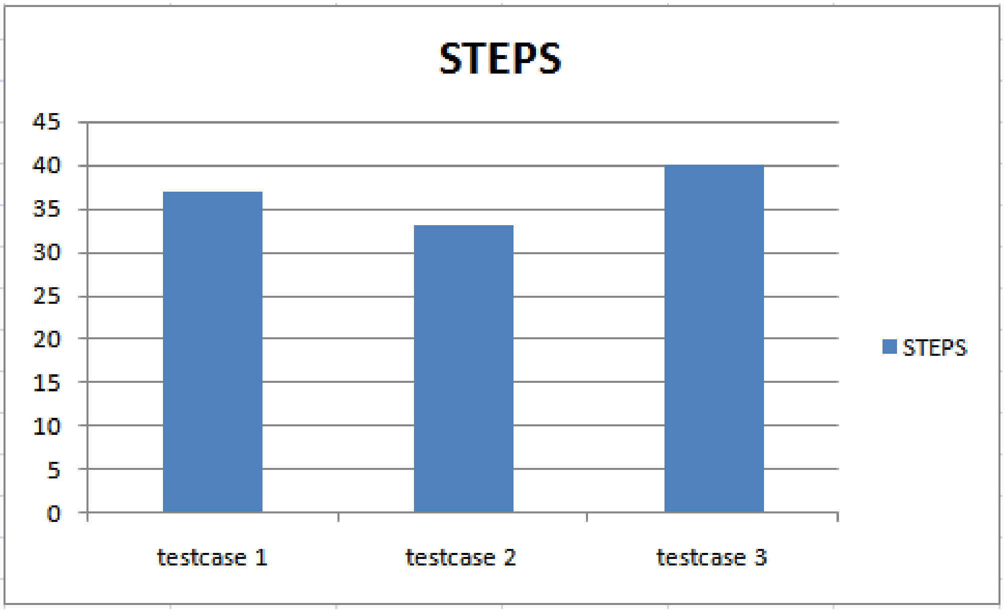
## 5.4 Comparison test cases

We make two types of comparison between the test cases, the first is the time and the second is the number of steps.

The following image shows the time of the test cases to execute the STRIPS algorithm. As we can see we measure the time in nanoseconds, which means that the execution is very fast. One of the things that can influence the time is the distance that the robot needs to travel to serve all the petitions.



We also compare the number of steps the robot travels to serve the requests of the offices. The image shows the table with the comparison.



## 6 Intructions to execute the program

The implementation is given as an Eclipse project. For testing a new sample is needed to place the file in the package named pack.

The file must have the structure defined in chapter 4 section 2. In the Strips class constructor we need to change the parameter to the read_file method and set the name of the file with the initial and final state.

# Appendices

Stack of case 3:

- PUSH List Robot-location(10) ; Served(12) ; Served(1) ; Served(25) ; Served(32) ; Served(11) ; Served(30) ;
- PUSH Predicate Robot-location(10)
- PUSH Predicate Served(12)
- PUSH Predicate Served(1)
- PUSH Predicate Served(25)
- PUSH Predicate Served(32)
- PUSH Predicate Served(11)
- PUSH Predicate Served(30)
- POP Predicate Served(30)
- PUSH Operator Serve(30,2)
- PUSH List Robot-location(30) ; Robot-loaded(2) ; Petition(30,2);
- PUSH Predicate Robot-location(30)
- PUSH Predicate Robot-loaded(2)
- PUSH Predicate Petition(30,2)
- POP Predicate Petition(30,2)
- POP Predicate Robot-loaded(2)
- PUSH Operator Make(4,2)
- PUSH List Robot-location(4) ; Robot-free) ; Machine(4,2);
- PUSH Predicate Robot-location(4)
- PUSH Predicate Robot-free
- PUSH Predicate Machine(4,2)
- POP Predicate Machine(4,2)
- POP Predicate Robot-free
- POP Predicate Robot-location(4)
- PUSH Operator Move(10,4)
- PUSH List Robot-location(10) ; Steps(0) ;
- PUSH Predicate Robot-location(10)

- PUSH Predicate Steps(0)
- POP Predicate Steps(0)
- POP Predicate Robot-location(10)
- POP List Robot-location(10) ; Steps(0) ;
- POP Operator Move(10,4)
- POP List Robot-location(4) ; Robot-free) ; Machine(4,2);
- POP Operator Make(4,2)
- POP Predicate Robot-location(30)
- PUSH Operator Move(4,30)
- PUSH List Robot-location(4) ; Steps(1) ;
- PUSH Predicate Robot-location(4)
- PUSH Predicate Steps(1)
- POP Predicate Steps(1)
- POP Predicate Robot-location(4)
- POP List Robot-location(4) ; Steps(1) ;
- POP Operator Move(4,30)
- POP List Robot-location(30) ; Robot-loaded(2) ; Petition(30,2);
- POP Operator Serve(30,2)
- POP Predicate Served(11)
- PUSH Operator Serve(11,3)
- PUSH List Robot-location(11) ; Robot-loaded(3) ; Petition(11,3);
- PUSH Predicate Robot-location(11)
- PUSH Predicate Robot-loaded(3)
- PUSH Predicate Petition(11,3)
- POP Predicate Petition(11,3)
- POP Predicate Robot-loaded(3)
- PUSH Operator Make(34,3)
- PUSH List Robot-location(34) ; Robot-free) ; Machine(34,3);
- PUSH Predicate Robot-location(34)
- PUSH Predicate Robot-free)

- PUSH Predicate Machine(34,3)
- POP Predicate Machine(34,3)
- POP Predicate Robot-free)
- POP Predicate Robot-location(34)
- PUSH Operator Move(30,34)
- PUSH List Robot-location(30) ; Steps(7) ;
- PUSH Predicate Robot-location(30)
- PUSH Predicate Steps(7)
- POP Predicate Steps(7)
- POP Predicate Robot-location(30)
- POP List Robot-location(30) ; Steps(7) ;
- POP Operator Move(30,34)
- POP List Robot-location(34) ; Robot-free) ; Machine(34,3);
- POP Operator Make(34,3)
- POP Predicate Robot-location(11)
- PUSH Operator Move(34,11)
- PUSH List Robot-location(34) ; Steps(10) ;
- PUSH Predicate Robot-location(34)
- PUSH Predicate Steps(10)
- POP Predicate Steps(10)
- POP Predicate Robot-location(34)
- POP List ¡ Robot-location(34) ; Steps(10) ;¿
- POP Operator Move(34,11)
- POP List Robot-location(11) ; Robot-loaded(3) ; Petition(11,3);
- POP Operator Serve(11,3)
- POP Predicate Served(32)
- PUSH Operator Serve(32,2)
- PUSH List Robot-location(32) ; Robot-loaded(2) ; Petition(32,2);
- PUSH Predicate Robot-location(32)
- PUSH Predicate Robot-loaded(2)

- PUSH Predicate Petition(32,2)
- POP Predicate Petition(32,2)
- POP Predicate Robot-loaded(2)
- PUSH Operator Make(4,2)
- PUSH List Robot-location(4) ; Robot-free) ; Machine(4,2);
- PUSH Predicate Robot-location(4)
- PUSH Predicate Robot-free)
- PUSH Predicate Machine(4,2)
- POP Predicate Machine(4,2)
- POP Predicate Robot-free)
- POP Predicate Robot-location(4)
- PUSH Operator Move(11,4)
- PUSH List Robot-location(11) ; Steps(15) ;
- PUSH Predicate Robot-location(11)
- PUSH Predicate Steps(15)
- POP Predicate Steps(15)
- POP Predicate Robot-location(11)
- POP List Robot-location(11) ; Steps(15) ;
- POP Operator Move(11,4)
- POP List Robot-location(4) ; Robot-free) ; Machine(4,2);
- POP Operator Make(4,2)
- POP Predicate Robot-location(32)
- PUSH Operator Move(4,32)
- PUSH List Robot-location(4) ; Steps(17) ;
- PUSH Predicate Robot-location(4)
- PUSH Predicate Steps(17)
- POP Predicate Steps(17)
- POP Predicate Robot-location(4)
- POP List Robot-location(4) ; Steps(17) ;
- POP Operator Move(4,32)

- POP List Robot-location(32) ; Robot-loaded(2) ; Petition(32,2);
- POP Operator Serve(32,2)
- POP Predicate Served(25)
- PUSH Operator Serve(25,1)
- PUSH List Robot-location(25) ; Robot-loaded(1) ; Petition(25,1);
- PUSH Predicate Robot-location(25)
- PUSH Predicate Robot-loaded(1)
- PUSH Predicate Petition(25,1)
- POP Predicate Petition(25,1)
- POP Predicate Robot-loaded(1)
- PUSH Operator Make(19,1)
- PUSH List Robot-location(19) ; Robot-free) ; Machine(19,1);
- PUSH Predicate Robot-location(19)
- PUSH Predicate Robot-free)
- PUSH Predicate Machine(19,1)
- POP Predicate Machine(19,1)
- POP Predicate Robot-free)
- POP Predicate Robot-location(19)
- PUSH Operator Move(32,19)
- PUSH List Robot-location(32) ; Steps(24) ;
- PUSH Predicate Robot-location(32)
- PUSH Predicate Steps(24)
- POP Predicate Steps(24)
- POP Predicate Robot-location(32)
- POP List Robot-location(32) ; Steps(24) ;
- POP Operator Move(32,19)
- POP List Robot-location(19) ; Robot-free) ; Machine(19,1);
- POP Operator Make(19,1)
- POP Predicate Robot-location(25)
- PUSH Operator Move(19,25)

- PUSH List Robot-location(19) ; Steps(27) ;
- PUSH Predicate Robot-location(19)
- PUSH Predicate Steps(27)
- POP Predicate Steps(27)
- POP Predicate Robot-location(19)
- POP List Robot-location(19) ; Steps(27) ;
- POP Operator Move(19,25)
- POP List Robot-location(25) ; Robot-loaded(1) ; Petition(25,1);
- POP Operator Serve(25,1)
- POP Predicate Served(1)
- PUSH Operator Serve(1,1)
- PUSH List Robot-location(1) ; Robot-loaded(1) ; Petition(1,1);
- PUSH Predicate Robot-location(1)
- PUSH Predicate Robot-loaded(1)
- PUSH Predicate Petition(1,1)
- POP Predicate Petition(1,1)
- POP Predicate Robot-loaded(1)
- PUSH Operator Make(19,1)
- PUSH List Robot-location(19) ; Robot-free) ; Machine(19,1);
- PUSH Predicate Robot-location(19)
- PUSH Predicate Robot-free)
- PUSH Predicate Machine(19,1)
- POP Predicate Machine(19,1)
- POP Predicate Robot-free)
- POP Predicate Robot-location(19)
- PUSH Operator Move(25,19)
- PUSH List Robot-location(25) ; Steps(28) ;
- PUSH Predicate Robot-location(25)
- PUSH Predicate Steps(28)
- POP Predicate Steps(28)

- POP Predicate Robot-location(25)
- POP List Robot-location(25) ; Steps(28) ;
- POP Operator Move(25,19)
- POP List Robot-location(19) ; Robot-free) ; Machine(19,1);
- POP Operator Make(19,1)
- POP Predicate Robot-location(1)
- PUSH Operator Move(19,1)
- PUSH List Robot-location(19) ; Steps(29) ;
- PUSH Predicate Robot-location(19)
- PUSH Predicate Steps(29)
- POP Predicate Steps(29)
- POP Predicate Robot-location(19)
- POP List Robot-location(19) ; Steps(29) ;
- POP Operator Move(19,1)
- POP List Robot-location(1) ; Robot-loaded(1) ; Petition(1,1);
- POP Operator Serve(1,1)
- POP Predicate Served(12)
- PUSH Operator Serve(12,1)
- PUSH List Robot-location(12) ; Robot-loaded(1) ; Petition(12,1);
- PUSH Predicate Robot-location(12)
- PUSH Predicate Robot-loaded(1)
- PUSH Predicate Petition(12,1)
- POP Predicate Petition(12,1)
- POP Predicate Robot-loaded(1)
- PUSH Operator Make(7,1)
- PUSH List Robot-location(7) ; Robot-free) ; Machine(7,1);
- PUSH Predicate Robot-location(7)
- PUSH Predicate Robot-free)
- PUSH Predicate Machine(7,1)
- POP Predicate Machine(7,1)

- POP Predicate Robot-free)
- POP Predicate Robot-location(7)
- PUSH Operator Move(1,7)
- PUSH List Robot-location(1) ; Steps(32) ;
- PUSH Predicate Robot-location(1)
- PUSH Predicate Steps(32)
- POP Predicate Steps(32)
- POP Predicate Robot-location(1)
- POP List Robot-location(1) ; Steps(32) ;
- POP Operator Move(1,7)
- POP List Robot-location(7) ; Robot-free) ; Machine(7,1);
- POP Operator Make(7,1)
- POP Predicate Robot-location(12)
- PUSH Operator Move(7,12)
- PUSH List Robot-location(7) ; Steps(33) ;
- PUSH Predicate Robot-location(7)
- PUSH Predicate Steps(33)
- POP Predicate Steps(33)
- POP Predicate Robot-location(7)
- POP List Robot-location(7) ; Steps(33) ;
- POP Operator Move(7,12)
- POP List Robot-location(12) ; Robot-loaded(1) ; Petition(12,1);
- POP Operator Serve(12,1)
- POP Predicate Robot-location(10)
- PUSH Operator Move(12,10)
- PUSH List Robot-location(12) ; Steps(38) ;
- PUSH Predicate Robot-location(12)
- PUSH Predicate Steps(38)
- POP Predicate Steps(38)
- POP Predicate Robot-location(12)

- POP List Robot-location(12) ; Steps(38) ;
- POP Operator Move(12,10)
- POP List Robot-location(10) ; Served(12) ; Served(1) ; Served(25) ; Served(32) ; Served(11) ; Served(30) ;

## References

[1] ”Stripslet: a Robocup-playing STRIPS Planner based on Krislet”, [Online]. Available: , `http:http://www.employees.org/ alokem/robocup-sw/stripslet.html` [Accesed: Nov. 01, 2016]

[2] ”Heuristic”, [Online]. Available: , `https://en.wikipedia.org/wiki/Heuristic` [Accesed: Nov. 01, 2016]

[3] Elif Aktolga. [*A Java planner for Blocksworld problems*]. `http://www.cogsys.wiai.uni-bamberg.de/schmid/uoshp/thesesuos/aktolga/ba.pdf`