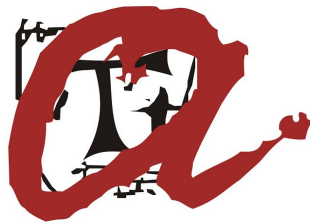


Planner exercise

Planning and Approximate Reasoning



UNIVERSITAT
ROVIRA I VIRGILI

Miguel Martínez
Suren Oganessian
MAI - PAR
2016

The coffee server

1. Introduction to the problem

For this assignment we had to create a linear planner by implementing the Stanford Research Institute Problem Solver (STRIPS) algorithm for a coffee server robot problem (this algorithm should work well enough to solve the problem we are given). Our environment will consist of a 6 by 6 matrix, which represents an office. The environment will have a robot, and coffee machines in different fields on the grid. In this environment the robot is allowed to move only horizontally and vertically to the adjacent offices. We will receive a file with information on the world we are working in, which will have the initial state: robot position, the coffee machines position and the coffee petitions, as well as the final goal state: offices served and the final robot position. Our objective will be to have our robot deliver the coffee to the offices, in the most efficient way.

2. Analysis of the Problem

1. Search Space

We are given a world represented by a 6 x 6 matrix. Each cell, of the matrix represents an office, which may establish a petition for coffee, and/or contain a coffee machine, and/or the robot that will be serving the coffee.

2. Operators

The table of operators, their preconditions and results of the linear planner are:

Operators	Preconditions	Results
Make(o,n)	robot-location(o) robot-free machine(o,n)	Adds: robot-loaded(n) Deletes: robot-free
Move(o1,o2)	robot-location(o1) steps(x)	Adds: robot-location(o2), steps(x+distance(o1,o2)) Deletes: robot-location(o1) steps(x)
Serve(o,n)	robot-location(o) robot-loaded(n)	Adds: served(o)

	petition(o,n)	robot-free Deletes: petition(o,n) robot-loaded(n)
--	---------------	--

3. Special situations

We decided to test how would the planner, in its current configuration, work if all the machines had the capacity of 1 cup of coffee. As expected the planner failed to find a solution for this problem, as it is not configured to use 1 cup machines 3 or 2 times to make petitions of 3 and 2 cups.

We tested what would happen if an office would have both coffee machines and requests on the same grid, which to no surprise failed, as our system is setup for only the robot having the capability of being in the same grid cell with a machine or petition.

The robot's initial position might be in any cell, including those with requests, and coffee machines.

The discrimination on which coffee machine to choose is based on minimum distance from request to coffee machine.

3. Planning Algorithm

The STRIPS algorithm pseudo code can be represented in the following way:

STRIPS (initial-state, goals)

- State = initial-state; plan = []; stack = []
- Push goals on stack
- Repeat until stack is empty
 - If top of stack is goal that matches state, then pop stack
 - Else if top of stack is a simple goal sg, then
 - Choose an operator o whose add-list matches goal sg
 - Replace goal sg with operator o
 - Push the preconditions of o on the stack
 - Else if top of stack is an operator o, then
 - state = apply(o, state)
 - plan = [plan; o]

4. Implementation design

We decided on using Python 3, and doing a sequential implementation, which how the STRIPS algorithm works. We first created an empty stack, that would be filled with the goals and

operators. Then we define each operator and set all their preconditions and how the add and delete will work in the stack of goals. At the end of the day what we are looking for is a list of operators that will tell us how to go about doing things.

We decided to add a parser into the planner, to help us identify the initial and goal states, that also removed whitespace, even though the assignment said that there will be no errors in the input code, we did come across white space in the given default input source. We decided to also add an “op:” tag to the operations, so we could distinguish them from goals.

5. Testing cases and results

We tested our planner program with 10 different scenarios, but included only 3 tests in this report. The first test, was the default scenario that we received during the assignment:













InitialState=Robot-location(o1);Machine(o4,3);Machine(o8,1);Machine(o21,2);Machine(o23,1);Machine(o31,2);Petition(o3,1); Petition(o11,3);Petition(o12,1);Petition(o13,2);Petition(o25,1);

GoalState=Robot-location(o7);Served(o3);Served(o11);Served(o12); Served(o13);Served(o25);







Our planner managed to complete this serving in 31 steps (human calculation 25 steps). Below we can see the plan the program gives for this task:

'op:Move(1,8)', 'op:Make(8,1)', 'op:Move(8,25)', 'op:Serve(25,1)', 'op:Move(25,31)', 'op:Make(31,2)',
'op:Move(31,13)', 'op:Serve(13,2)', 'op:Move(13,8)', 'op:Make(8,1)', 'op:Move(8,12)',
'op:Serve(12,1)', 'op:Move(12,4)', 'op:Make(4,3)', 'op:Move(4,11)', 'op:Serve(11,3)',
'op:Move(11,23)', 'op:Make(23,1)', 'op:Move(23,3)', 'op:Serve(3,1)', 'op:Move(3,7)'

Initial state:

O(1) 	O(2)	O(3) Petition (1 Cup) 	O(4) Machine (3 Cups) 	O(5)	O(6)
O(7)	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)	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)

Final state:

O(1)	O(2)	O(3)	O(4) Machine (3 Cups) 	O(5)	O(6)
O(7) 	O(8) Machine (1 Cup) 	O(9)	O(10)	O(11)	O(12)
O(13)	O(14)	O(15)	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)	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 second test was performed on the following scenario:

InitialState=Robot-location(o22);Machine(o1,1);Machine(o6,2); Machine(o36,3);Petition(o3,3);
 Petition(o8,1);Petition(o11,1);Petition(o15,3);Petition(o18,2);Petition(o19,2);Petition(o29,1);Petition(
 o31,2);Petition(o33,3);
 GoalState=Robot-location(o21);Served(o3);Served(o8);Served(o11);
 Served(o15);Served(o18);Served(o19);Served(o29);Served(o31);Served(o33)





Our planner managed to complete this serving in 105 steps. Below we can see the plan the program gives for this task:

'op:Move(22,36)', 'op:Make(36,3)', 'op:Move(36,33)', 'op:Serve(33,3)', 'op:Move(33,6)',
 'op:Make(6,2)', 'op:Move(6,31)', 'op:Serve(31,2)', 'op:Move(31,1)', 'op:Make(1,1)', 'op:Move(1,29)',
 'op:Serve(29,1)', 'op:Move(29,6)', 'op:Make(6,2)', 'op:Move(6,19)', 'op:Serve(19,2)', 'op:Move(19,6)',
 'op:Make(6,2)', 'op:Move(6,18)', 'op:Serve(18,2)', 'op:Move(18,36)', 'op:Make(36,3)',
 'op:Move(36,15)', 'op:Serve(15,3)', 'op:Move(15,1)', 'op:Make(1,1)', 'op:Move(1,11)',
 'op:Serve(11,1)', 'op:Move(11,1)', 'op:Make(1,1)', 'op:Move(1,8)', 'op:Serve(8,1)', 'op:Move(8,36)',
 'op:Make(36,3)', 'op:Move(36,3)', 'op:Serve(3,3)', 'op:Move(3,21)'

Initial state:

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

Final state:

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








The last test was performed on the following scenario:

InitialState=Robot-location(o1);Machine(o36,1);Machine(o35,3); Machine(o34,2);Petition(o2,1);
 Petition(o3,3);Petition(o4,2);
 GoalState=Robot-location(o7);Served(o2);Served(o3);Served(o4);




Our planner managed to complete this serving in 45 steps. Below we can see the plan the program gives for this task:

'op:Move(1,34)', 'op:Make(34,2)', 'op:Move(34,4)', 'op:Serve(4,2)', 'op:Move(4,35)', 'op:Make(35,3)',
 'op:Move(35,3)', 'op:Serve(3,3)', 'op:Move(3,36)', 'op:Make(36,1)', 'op:Move(36,2)', 'op:Serve(2,1)',
 'op:Move(2,7)'

Initial state:

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

Final state:

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

We can see that no matter how complex the environment is, our planner still manages to find a solution to the given problem. Our algorithm minimizes distances based on distance between requests and coffee machines, which simplifies the goal of our assignment. This might not necessarily be the best heuristic, as that would be one which minimizes the distance the robot has to traverse. Even though the planner does not find the most efficient solution (least steps necessary) it does complete the job with close to the best results. If we wanted to try and find the algorithm for the planner to achieve the most efficient solution, we would have to look into other algorithms and test them out and compare the result. It is possible that for some

scenarios there might be a better solution, but that could impact the solution for other scenarios too.

6. Instructions to execute the program.

The program is written in Python 3. Run `planner.py` using `python 3`, the program will ask for the input file name (for example: `'input.txt'`). Then press enter and the program will execute the planner and calculate the amount of steps it takes, and show a detailed plan for the given world.