

Coffee Delivery Robot Linear Planner

MAI-PAR Planning Exercise

Cole MacLean and Lucky Ekechi

November 6, 2016

Introduction

A discrete state planning exercise is presented to utilize and test the linear planning with a stack of goals (PRISM (<http://www.cs.cmu.edu/~mmv/planning/readings/strips.pdf>)) algorithm for agent planning.

A mobile robotic agent capable of making and serving coffee to offices within a 2D office building is tasked with efficiently serving the requested coffee petitions in the least amount of steps to reduce office worker disturbances. The office building is modelled as a 6x6 square grid of 36 offices, and the steps between 2 offices is defined as their manhattan distance. The agent is fed information about coffee machine and petition capacities and office locations, and is required to build a daily service plan to efficiently fulfill the petitions given the new office configuration.

The agent is a linear planner with 3 operators: Move, Make and Serve. The preconditions, additions and deletions of each operator in the planning algorithm are listed below.

Make(o,n): the robot makes n cups of coffee in the machine located at office o

- Preconditions: robot-location(o), robot-free, machine(o,n)
- Add: robot-loaded(n)
- Delete: robot-free

Move(o1,o2): the robot moves from o1 to o2

- Preconditions: robot-location(o1), steps(x)
- Add: robot-location(o2), steps(x+distance(o1,o2))
- Delete: robot-location(o1), steps(x)

Serve(o,n): the robot delivers n cups of coffee to office o

- Preconditions: robot-location(o), robot-loaded(n), petition(o,n)
- Add: served(o), robot-free
- Delete: petition(o,n), robot-loaded(n)

A linear planner using these operators and some simple search heuristics has been implemented and can be tested [here](http://ponderinghydrogen.pythonanywhere.com/) (<http://ponderinghydrogen.pythonanywhere.com/>)

Plan Step = 7 

initialize-----total steps = 0
 move,0,7-----total steps = 2
 make,7,1-----total steps = 2
 move,7,2-----total steps = 4
 serve,2,1-----total steps = 4
 move,2,3-----total steps = 5
 make,3,3-----total steps = 5
 move,3,10-----total steps = 7

Algorithm Implementation

The linear planner is implemented as a class object with 3 main methods that perform: Action search space traversal, Action filtering heuristics and Action execution.

Initialization

The Planner class is initialized with the initial state of the form

```
{'robot-location': 0, 'steps': 0,
'petitions': {'office':petition_count},
'robot-free': True, 'robot-loaded': 0,
'machines': {'office':capacity},
'served': []}
```

In [2]: **import** Planner

In [3]: `my_planner = Planner.Planner({'robot-location': 3, 'steps': 0,
'petitions': {2: 1, 10: 3, 11: 1, 12: 2, 24: 1},
'robot-free': True, 'robot-loaded': 0,
'machines': {3: 3, 7: 1, 20: 2, 22: 1,30:2},
'served': []},36)`

Search Space Traversal

The first method, `possible_actions`, checks the current state compared to operator preconditions and builds a list of all legal actions.

```
In [6]: acts = my_planner.possible_actions()  
acts
```

```
Out[6]: [['move', [3, 0]],  
         ['move', [3, 1]],  
         ['move', [3, 2]],  
         ['move', [3, 4]],  
         ['move', [3, 5]],  
         ['move', [3, 6]],  
         ['move', [3, 7]],  
         ['move', [3, 8]],  
         ['move', [3, 9]],  
         ['move', [3, 10]],  
         ['move', [3, 11]],  
         ['move', [3, 12]],  
         ['move', [3, 13]],  
         ['move', [3, 14]],  
         ['move', [3, 15]],  
         ['move', [3, 16]],  
         ['move', [3, 17]],  
         ['move', [3, 18]],  
         ['move', [3, 19]],  
         ['move', [3, 20]],  
         ['move', [3, 21]],  
         ['move', [3, 22]],  
         ['move', [3, 23]],  
         ['move', [3, 24]],  
         ['move', [3, 25]],  
         ['move', [3, 26]],  
         ['move', [3, 27]],  
         ['move', [3, 28]],  
         ['move', [3, 29]],  
         ['move', [3, 30]],  
         ['move', [3, 31]],  
         ['move', [3, 32]],  
         ['move', [3, 33]],  
         ['move', [3, 34]],  
         ['move', [3, 35]],  
         ['make', [3, 1]],  
         ['make', [3, 2]],  
         ['make', [3, 3]]]
```

Search Space Filtering Heuristics

The method, `select_action`, uses 4 simple rules to select the optimal action from the list of all possible actions.

#1.) If a serve action exists in the `action_list`, then perform the serve action

#2.) If robot is loaded with a petitioned amount of cups, perform move action to the minimum manhattan distance to the petitioning offices

#3.) If a make action is in the list, and #ofcups equals an existing petitioned #ofcups, then perform the make action that can serve the closest petition

#4.) If no other actions than move to a machine exist, move to the closest machine capable of making #ofcups > or = an existing petitioned #ofcups

```
In [7]: my_planner.select_action(acts)
```

```
Out[7]: ['make', [3, 1]]
```

Action Execution

The method, `perform_step`, executes the selected action and stores the resulting new state in the `planner.plan` property. A helper function, `build_plan`, iteratively performs the `perform_step` method until the `goal_state` is satisfied, and returns the full plan with intermediate states and actions for the agent.

```
In [8]: my_planner.build_plan()
```

```
Out[8]: [{ 'action': 'initialize',
  'state': { 'machines': {3: 3, 7: 1, 20: 2, 22: 1, 30: 2},
    'petitions': {2: 1, 10: 3, 11: 1, 12: 2, 24: 1},
    'robot-free': True,
    'robot-loaded': 0,
    'robot-location': 3,
    'served': [],
    'steps': 0}},
  { 'action': ['make', [3, 1]],
    'state': { 'machines': {3: 3, 7: 1, 20: 2, 22: 1, 30: 2},
      'petitions': {2: 1, 10: 3, 11: 1, 12: 2, 24: 1},
      'robot-free': False,
      'robot-loaded': 1,
      'robot-location': 3,
      'served': [],
      'steps': 0}},
  { 'action': ['move', [3, 2]],
    'state': { 'machines': {3: 3, 7: 1, 20: 2, 22: 1, 30: 2},
      'petitions': {2: 1, 10: 3, 11: 1, 12: 2, 24: 1},
      'robot-free': False,
```

Final Results

Using a simple linear planner with stack of goals and simple search space heuristics, an efficient agent for coffee service planning can be implemented. Through many different office configurations, the implemented agent performs logically and efficiently in collecting and serving petitions.

Limitations

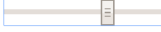
The described linear planner implementation does not guarantee maximally optimal service plans. Although the 4 rules for search space filtering capture nearly all situations for optimal action selection, there does exist situations where the agent will perform sub-optimally. These cases arise when a coffee machine that can serve a petition is closer to that petition, to another capable machine closer to the agent, causing the agent to unnecessarily. An example configuration of this is presented here.

Plan Step = 2 



initialize-----total steps = 0
 move,0,2-----total steps = 2
 make,2,1-----total steps = 2

In this example, the optimal route is for the agent to use the 2 cup capacity machine to serve the single cup petition, and carry-on towards the 3 cup capacity machine to serve the 2 cup petition. The above implementation incorrectly utilize the 2 cup capacity machine to serve both petitions, due to search heuristic #4 move to closest machine capable of serving an existing petition.

Plan Step = 6 

initialize-----total steps = 0
 move,0,2-----total steps = 2
 make,2,1-----total steps = 2
 move,2,3-----total steps = 3
 serve,3,1-----total steps = 3
 move,3,2-----total steps = 4
 make,2,2-----total steps = 4

Future work to improve the above implementation and correct this suboptimal behaviour would be to implement full search tree traversal and selection of the globally optimal plan. However, the advantages of the above implementation is its superior ability to scale to larger input configuration, compared to the computationally expensive method of a full search tree traversal implementation.

Execution Instructions

A graphical interface for the described agent can be found [here](http://ponderinghydrogen.pythonanywhere.com/)

(<http://ponderinghydrogen.pythonanywhere.com/>) To initialize the agent, fill in the desired coffee machine capacities and petitions in the correct offices and input the initial office of the robot in the 'robot_cell' input box. For example, if office 16 has a machine with capacity of 3 cups, and office 23 has a petition for 2 cups, input 3 in the 'machines' input box of office 16, and 2 in the 'petitions' input box of office 23. If the agent starts in office 2, enter 2 in the 'robot_cell' input box. Once configured, select the 'Submit' button.

You will be redirected to a graphical simulation of the agents plan, that can be stepped through using the 'Plan Step' slider to visualize plan execution.