Universitat Rovira i Virgili

Planning and Approximate Reasoning

Planner Practicum

Linial Planner with Stack of Goals

Authors:
Julià Camps Sereix

Supervisor:
Dr. Aida Valls

December 24, 2015

Contents

Introduction to the problem

In this practical exercise we will solve a planning problem using the well known lineal-planer with stack of goals.

This planner integrates a stack where on each iteration of the solution search tries to take out objectives from the stack by accomplishing them.

This simple behaviour will give us some advantages, such as being able to improve allot it's performance by adding heuristics and intelligence on it.

On the other hand it will restrict initially our planner to be inefficient, and perhaps, unable to solve some complex situations by it's standard behaviour.

Problem description

There is a square building composed by 9 offices, which are located in a matrix of 3 rows and 3 columns. From each office it is possible to move (horizontally or vertically) to the adjacent offices. Each of the offices may be clean or dirty. There is an automated cleaning robot that can move from one office to another. Each office may be empty or it may contain one box (there can't be more than one box in the same office). The number of boxes in the building may be between 1 and 8. The robot can push a box from one office to an adjacent office. An office can be cleaned only if it is empty. The robot will start with a given initial configuration (different for each test), with some clean and dirty offices, and with a box in some rooms. In the goal state all rooms will be clean, and the final position of the boxes will be explicitly indicated.

Operators to consider

The operators to be considered are the following:

- Clean-office(o): the robot cleans office o
  - Preconditions: robot-location(o), dirty(o), empty(o)
  - Add: clean(o)
  - Delete: dirty(o)
- Move(o1,o2): the robot moves from o1 to o2
  - Preconditions: robot-location(o1), adjacent(o1,o2)
  - Add: robot-location(o2)
  - Delete: robot-location(o1)
- Push(b,o1,o2): the robot pushes box b from o1 to o2
  - Preconditions: robot-location(o1), box-location(b,o1), adjacent(o1,o2), empty(o2)
  - Add: box-location(b,o2), robot-location(o2), empty(o1)
  - Delete: empty(o2), box-location(b,o1), robot-location(o1)

Predicates to consider

The predicates to be considered are the following:

- Robot-location(o): the robot is in office o.
- Box-location(b,o): box b is located in office o.
- Dirty(o): office o is dirty.
- Clean(o): office o is clean.
- Empty(o): there isn't any box in office o.
- Adjacent(o1,o2): offices o1 and o2 are horizontally or vertically adjacent (fixed information).

Glossary

Here I will explain some concepts used in my practicum than have different names than the concepts of the lectures, but actually mean the same.

- Condition: predicate.
- Partial-condition: partly instantiated predicate.
- Goal: objective of the goals stack.
- Settings: initial and final state together, is the description of a particular case for solving the generic planning problem.

Analysis of the problem

For this section of the practicum first I have analyzed the concepts involved in the problem presented. For doing it I build the following 'conceptual' diagram.

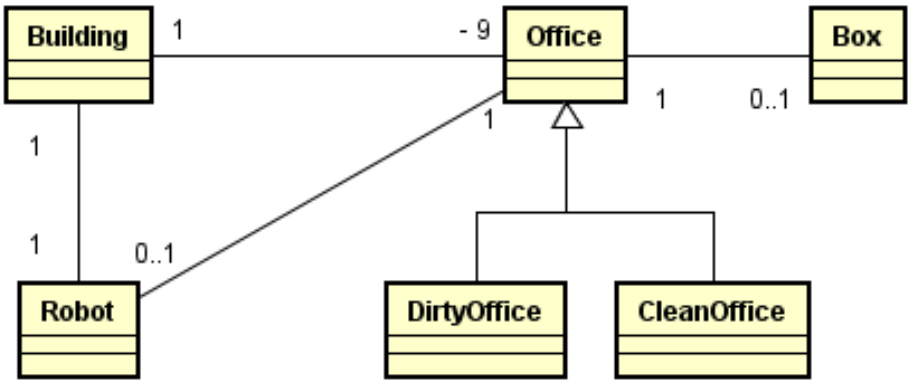

Figure 1: Conceptual diagram of the practicum problem

In figure 1 it can be observed than we have few elements involved in the initial problem presented. Even we have some numbers of the relations closed to a fixes values, so we don't have any ambiguity in this scenario.

Building

o1	o2	o3
o4	o5	o6
o7	o8	o9

Table 1: Scenario visual description

Special situations

After analysing the problem, I noticed than the complexity of the settings where given by the number of boxes, rather than the number of dirty offices.

From the problem description we know that as much we will have 8 boxes distributed on the building, so I could eventually have have to solve an 8-puzzle problem with cleaning office meanwhile.

I also considered some other complex situations:

- When for example the 'Adjacent' condition is needed, due than this condition can't be generated by any operator defined on the operators list.
- Having more than one operator than retrieves the condition 'Robot-location'. This could lead the robot to start dragging a box around the building without never being able to clean anything.
- Having to accumulate many objectives for accomplishing a previous one, this could difficult the understanding on what the algorithm is doing at some point, or lead to uncontrolled cases.

From controlling this cases I deduced than it should be able to solve much more difficult settings than the standard lineal-planner with stack of goals.

Planning algorithm

On this section I'll explain the strategies that I followed for building my lineal-planner with stack of goals.

Sorting objectives by office

After thinking about what I wanted the robot to do, I saw than he could do it in really messy orders, and go moving around without taking profit from his position.

So I build a distance matrix, where I mapped the distances from all offices.

Then the robot was trying to move to the closes offices, but this was not a good approach either, due than the closest offices where changing each time the robot was moving.

Finally I came up with the currently implemented idea, I presented an order among the offices of the building, so the robot will move from one goal to the other in a distance coherent order without complicating allot the next step decision.

Building

1rst	2nd	3rd
6th	5th	4th
7th	8th	9th

Table 2: Scenario visual moving order notion

Sorting objectives by type

From the previous approach I still could appreciate very undesirable orders, like for instance move the robot to the final position as the first action. Or pushing a box to it's final position without cleaning it.

For fixing this I proposed the following order:

1 Clean

2 Dirty

3 Box-location

4 Adjacent

5 Empty

6 Robot-location

In order to handle this order of objectives problem I decided to do a generic sort on the types of objectives, and with the matching objectives on the same type, sort them with the previous presented criteria.

So withing the 'Clean' conditions they will be sorted as shown on the table 2, but will always be treated the first ones when many conditions have to be considered at the same time (initially for example).

From this improvement I was able to avoid having the robot starting going to his final position, and also moving around on a crazy way due an undesirable order in spatial means.

Impossible operators

From having the 'Adjacent' condition, at first everything was easy and fancy, but when facing a simple example I noticed than the robot would not know how to cross offices just for moving to an far office, so this simple approximation was able to many problems, but only if all objectives where found from adjacent offices.

For solving this problem first I tried to use a recursive strategy, by in a movement operation performing small movement operations, but this notion was giving to much freedom to the order execution, and was destroying the notion of the stack how I was thinking about it.

My second approach was even worse, I tried to work from the side of the 'Adjacent' conditions, so when I found an 'Adjacent' condition on the stack I would try to manage to go to the desired location in order to achieve it. But, I had many problems, because I was allowing ambiguous situations. For example having the robot in one position, a box on another and willing to move the box from it's position to an non-adjacent position without moving the robot it self. It was trying to first go to the position, with another box and other similar useless behaviours like this one.

Finlay, I derived than complex conditions would be treated partially, so even than the objective was to drag a box to a non-adjacent position, the robot will think initially than the objective will be accomplished by just dragging it to an adjacent position of it's current location, so at each step he will just be moving the box one position towards the objective.

Operators ambiguity

As you can notice we have an undesirable operator than is the most complicate to handle, the 'Push' operator. This operator is not only leading difficult situations such as concatenating many objectives, but also has another handicap. The problem is than is moving the Robot with the dragged box.

So for moving the robot we have two possible operators:

- Move(o1,o2)
- Push(b,o1,o2)

This could lead the robot to try to approach an empty dirty office with carrying a box, and not being able to clean the mentioned office.

For solving this problem I redefined the priority of the operators such as when the objective is to move, the robot would not consider dragging a box with it.

1 Clean

2 Move

3 Push

Conflicts heuristic

The last intelligence than I added to the robot was to be able to compare all possible operators for each objective, and chose the best one according to the mentioned strategies, and also to try to not undo current work.

So I decided to give to the robot some memory of what he is trying to accomplish. By using checking if an operator is undoing work he should be able to avoid undoing the current work done if there are other possibilities, even if the undoing solution is shorter to execute.

From this approach I could observe than the robot was not considering difficulty of an operator. And I could not make the robot plan the full process of an operator, because otherwise I was deriving another type of planner.

So I decided just to check if one of the possible actions was performable in the current state. By this criteria the robot would prefer immediate solving operators rather than trying to follow very complicated plans.

Implementation design

Once we are sure of understanding completely the problem presented, I decided to proceed with the classes design and specification.

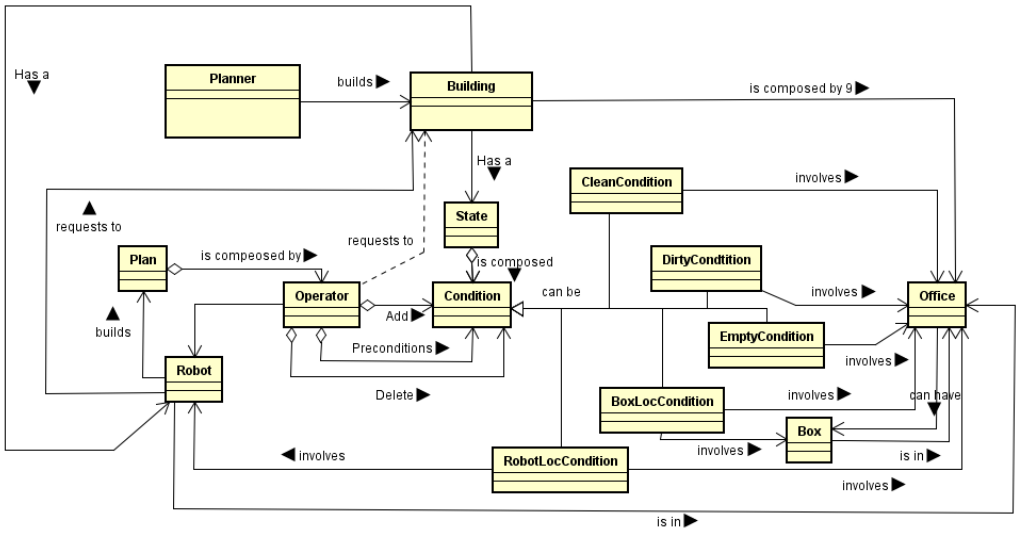

Figure 2: Components diagram with the class analysis

When implementing I tried to follow a complete solution design for the problem, considering further improvements to the presented solution.

Giving 'memory' to the Robot

In order to consider this improvements, I wanted to have a higher control on the current objectives of the robot, so I decided to give the robot the awareness of what is an condition or an operator for. So the robot will know than some of the predicates in the current state are there for some 'reason' (because he did an action to set them), but once this 'reason' is accomplished, he will forget that he did that before.

For implementing this strategy I wrapped the 'goal' concept in the problem, and gave to it a reference to his 'reason' of being stacked on the stack of goals. So even when an objective is accomplished it the robot won't forget it until the reason for accomplishing it is accomplished as well.

This temporal notion of dependencies will help the robot to avoid undoing his own work, I named this objectives as 'active objectives'. To this approach I decided to also integrate the notion of the undesirably lost properties, I named this lost objectives as 'pending objectives'.

This notion of the lost properties is not being used for taking more intelligent decisions, but is there for further improvements in considering solving more than one objective at the same time and better ordering of the objectives.

Taking away intelligence

After thinking about it I realised than the strategies that I could define for the robot could fail in some situations in which the best option always undoes some previous action, and fall in cycles.

As having a complete history of the past states would increase to much the complexity of the problem. I thought than there should be a better solution rather than building a full states tree.

I've been reading a paper in evolutionary algorithms, I've got inspired for trying a different strategy when my algorithm is not able to scape from a local minimum. I tried to simulate the mutation advantages, so for an unsolvable state for my algorithm, by mutation I will try to lead him to a solvable state.

I decided to set up an alarm to change the operators selection criteria. So when it shows up than the robot is only considering options without really performing them, I switch the last checks of which action is better to a random selection. The point is than if didn't do it it would always chose the first one found from the ones with equal heuristic value. And with this approach it seamed to solve many much more difficult problems than before.

So when the robot realises than his strategies are not working he changes them slightly and tries during some iterations with the new ones, which leads him to an unexplored state, and from there he tries again with the default strategies.

Giving the robot some coherence

From this I also added another special strategy, than it's, than if the robot crosses a dirty empty room, he will clean it before continuing.

When doing this last action, the stack of goals could have goals on it, but we could be on the final state. So the robot checks if he is on the final state at each round in order to avoid useless work.

Algorithm steps

First I implemented all the mentioned strategies of the previous section. Which ended being mainly an priority order of operators and conditions to be treated.

1 Take the final conditions and consider them a unique condition

2 Stack the condition block to the stack of goals

3 Sort conditions in type order

   1. Clean
      - Internally sort them in Office based order, such as shown in table 2.
   2. Dirty
      - Internally sort them in Office based order, such as shown in table 2.
   3. Box-location
      - ...
   4. Adjacent
   5. Empty

6. Robot-location

4 Stack all the sorted conditions to the stack of goals

5 Unstack a goal from the stack

1. If it's already accomplished or can be immediately accomplished
    1. Forget the related dependencies to this accomplished goal
    2. If the goal is wrapping an operator, execute it and apply it's consequences to the model and to the active objectives (passing them to the pending objectives set if necessary).
    3. Finally go to step 5
2. Otherwise do the following:

1. Choose an operator to solve it
    1. Select all possible operators
    2. The choosing criteria in importance order are the following:
        1. Retrieve an operator if exist
        2. Non conflicting with the 'current objectives'
        3. With a higher priority
            - Clean
            - Move
            - Push
        4. Performable in the current state
2. Stack this operator to the stack of goals
3. Take the preconditions of this operator and repeat the process from step 3

Testing cases and results

For a more detailed view of the results that will be commented on this section, I have an already generated 'logfile_old.txt'. In this file you can find the accurate description and visual interpretation of each step when solving the testing cases presented with the practicum code.

First test 'test1.txt'

This settings are the ones presented on practicum description as example. It's a very easy scenario in which the the most complicate part is to sort correctly the initial goals, in order to not have the robot going first to 'o9' because he has to finish there, then to clean 'o1', then to clean 'o9', etc.

For avoiding this crazy orders, I have set an priority order on the the conditions to be considered. But not only on the types of conditions, but also on the positions of each condition of each type. All this strategies have been deeply explained on the previous section in detail.

Initial State

o1	o2	o3
o4 Robot	o5 Box A	o6 Box B
o7	o8	o9

Table 3: Initial state setting in the test presented in the 'test1.txt' file.

Final State

o1	o2	o3
o4	o5	o6 Box B
o7	o8 Box A	o9 Robot

Table 4: Final state setting in the test presented in the 'test1.txt' file.

Evaluation on the Results

The retrieved solution was the following 17 operators Plan:

1 O1: Move(o4,o7)
2 O2: Move(o7,o8)
3 O3: Move(o8,o9)
4 O4: Clean(o9)
5 O5: Move(o9,o8)
6 O6: Clean(o8)
7 O7: Move(o8,o5)
8 O8: Push(A,o5,o2)
9 O9: Move(o2,o5)
10 O10: Clean(o5)
11 O11: Move(o5,o4)
12 O12: Move(o4,o1)
13 O13: Clean(o1)
14 O14: Move(o1,o2)
15 O15: Push(A,o2,o5)
16 O16: Push(A,o5,o8)
17 O17: Move(o8,o9)

Even than the solution retrieved is not the optimal plan, I think is a very good solution considering than is not very far from the optimal actually.

Second test 'test2.txt'

For the second test I decided to start exploring the limits of my planner, so I proposed a much more complex setting than the ones appearing on the practicum description.

Initial State

o1 Box A	o2 Box B	o3 Robot
o4 Box C	o5 Box D	o6
o7	o8	o9 Box E

Table 5: Initial state setting in the test presented in the 'test2.txt' file.

Final State

o1 Box A	o2	o3 Robot
o4 Box D	o5 Box E	o6
o7	o8 Box B	o9 Box C

Table 6: Final state setting in the test presented in the 'test2.txt' file.

Evaluation on the Results

The retrieved solution was the following 53 operators Plan:

1 The retreived plan is:
2 O1: Move(o3,o6)
3 O2: Move(o6,o9)
4 O3: Push(E,o9,o6)
5 O4: Move(o6,o9)
6 O5: Clean(o9)
7 O6: Move(o9,o8)
8 O7: Move(o8,o5)
9 O8: Push(D,o5,o8)
10 O9: Move(o8,o5)
11 O10: Clean(o5)
12 O11: Move(o5,o4)
13 O12: Push(C,o4,o5)
14 O13: Move(o5,o4)
15 O14: Clean(o4)
16 O15: Move(o4,o1)
17 O16: Move(o1,o2)
18 O17: Push(B,o2,o3)
19 O18: Move(o3,o2)
20 O19: Clean(o2)
21 O20: Move(o2,o1)
22 O21: Push(A,o1,o2)

23 O22: Move(o2,o1)
24 O23: Clean(o1)
25 O24: Move(o1,o2)
26 O25: Push(A,o2,o1)
27 O26: Move(o1,o2)
28 O27: Move(o2,o5)
29 O28: Push(C,o5,o2)
30 O29: Move(o2,o5)
31 O30: Move(o5,o6)
32 O31: Push(E,o6,o5)
33 O32: Move(o5,o8)
34 O33: Push(D,o8,o7)
35 O34: Push(D,o7,o4)
36 O35: Move(o4,o1)
37 O36: Move(o1,o2)
38 O37: Move(o2,o3)
39 O38: Push(B,o3,o6)
40 O39: Move(o6,o3)
41 O40: Move(o3,o2)
42 O41: Push(C,o2,o3)
43 O42: Move(o3,o6)
44 O43: Push(B,o6,o9)
45 O44: Move(o9,o6)
46 O45: Move(o6,o3)
47 O46: Push(C,o3,o6)
48 O47: Move(o6,o9)
49 O48: Push(B,o9,o8)
50 O49: Move(o8,o9)
51 O50: Move(o9,o6)
52 O51: Push(C,o6,o9)
53 O52: Move(o9,o6)
54 O53: Move(o6,o3)

As it can be observed the solution retrieved is very similar to one of the optimal solutions of this problem, just containing 53 operators, which is the same order of magnitude obtained for the first problem tested.

One important point is than the initial approach than I developed was not able to solve correctly this problem without the improvement of changing the behaviour of the algorithm for some iterations in order to continue advancing when it gets stuck.

Third test 'test3.txt'

This test as we can observe has a much higher complexity than the previous tests, the optimum solution is not so clear as we could think. And in fact the solution retrieved is very similar to the optimum one, both are from the same order of magnitude.

Initial State

o1 Box A	o2 Box B	o3 Box C
o4 Box D	o5 Box E	o6 Box F
o7	o8	Robot o9

Table 7: Initial state setting in the test presented in the 'test3.txt' file.

Final State

o1 Box A	o2	o3 Box B
o4 Box E	o5 Box F	o6 Box C
o7 Box D	o8	Robot o9

Table 8: Final state setting in the test presented in the 'test3.txt' file.

The solution retrieved was a 12731 operators plan. From the problem settings described in tables 7 and 8, this is not the optimal solution to the problem. But as we can see, the algorithm is able to solve it without getting stuck. Which, at least, is something already impressing knowing than the algorithm has been derived from a very simple planner.

Fourth test 'test4.txt'

For this test I set the most difficult situation than my planner had handled in the testing I've done.

It's a classical 8-puzzle problem with all offices dirty, so it's much more difficult than the previous tests done. But, I wasn't able to make the robot to solve the 8-puzzle in all the tests than I've done, so I decided to just request for having all offices clean, so the robot will have to clean all offices switching the boxes among the building.

In this example presented the final goal was:

- Robot-location(o2)
- Box-location(A,o1)
- Empty(o2)
- Box-location(B,o3)

This simplification is presented as one of the tests, to show than the robot is not blocking himself by having the maximum amount of blocks in the field.

The algorithm has solved the problem with 5660 operators, which is absolutely not the optimum solution.

Initial State

o1 Box A	o2 Box B	o3 Box C
o4 Box D	o5 Box E	o6 Box F
o7 Box G	o8 Box H	Robot o9

Table 9: Initial state setting in the test presented in the 'test4.txt' file.

Final State

o1 Box A	o2 Robot	o3 Box B
o4 Box E	o5 Box F	o6 Box C
o7 Box D	o8 Box G	o9 Box H

Table 10: Final state setting in the test presented in the 'test4.txt' file.

In table 10 we can see the retrieved solution when executing the planner on the described problem.

I have performed many other tests, but if I increase the difficulty of the problem, the planner finds really bad solutions, in the order of 10000 operators per solution (or more), which does not seam a correct solution from my point of view, in performance terms.

Instructions to execute the program

For the program execution, you do not need to consider any parameters when calling it. Due than if they haven't been sent with the code call as augments, they will be requested as input when the program starts with some indications about how should be this parameters set.

With the delivery I added an already generated 'logfile.txt', on it you will find a legend of explaining the information than you may find if you execute the code.

Settings file

This file should contain the description of the planning problem, as shown on the practicum documentation. Using the same tags for changing the described sections (e.g. "Boxes=", "Offices=", etc.).

Notice than the final state searched does not need an complete accurate description, can even be partial, because we will just look for one of the many solutions than implement all the conditions present on this state. On the other hand, we can't miss any condition on the initial state description, because otherwise we would not be able to state if some of the preconditions for a given operator are being accomplished or not.