Planning, Execution, and Learning 15-887
Homework 1

Abhishek Bhatia (abhatia1)

1) Lunar Lockout Game

a) **Representation 1:**
Objects: Player, non-player, 5x5 board.
States: Vector composed of coordinates of each non-player and player where for a n-length vector 1 - (n-1) elements represent a non-player and nth element represents a player.
Actions: Move each component of the state vector (player/non-player) up, down, left, and right.
Preconditions: Player/non-player continues moving in a direction till it hits another player/non-player, hence for a valid move there should be someone blocking the way in that direction.
Domain axioms: Player/non-player cannot land on top of another player/non-player, and player/non-player cannot move outside the board or cannot cross the boundaries.

Representation 2:
Objects: Player, non-player, 5x5 board.
States: Matrix composed of 5x5 board, each node of the matrix is either empty, or has a player or a non-player.
Actions: Each node of the board corresponding to the current state is evaluated, if a node is empty it is ignored. But if a node has a player or a non-player, the player/non-player is moved up, down, left, and right.
Preconditions: Player/non-player continues moving in a direction till it hits another player/non-player, hence for a valid move there should be someone blocking the way in that direction.
Domain axioms: Player/non-player cannot land on top of another player/non-player, and player/non-player cannot move outside the board or cannot cross the boundaries.

b) The number of possible states and the number of feasible states will be exactly the same for each representation. However, the representation 1 is preferable over representation 2 because the number of actions for representation 1 is much lesser. In representation 1 we only compute valid moves corresponding to each element in the vector, whereas in representation 2 we first evaluate if the node has a player/non-player and then evaluate valid moves corresponding to each player/non-player. Thus, in representation 2, to compute valid next states, we end up comparing much more conditions than representation 1, making representation 1 computationally preferable. However, representation 2 is much better for demonstration, as you will see while running the planner with both the representations. The state output at every state for representation 2 is the board itself, hence makes the analysis and debugging much easier.

c) The code dumps the output on the standard output of the terminal. The output contains the transition from the input state to the output state through a bunch of actions. The output also shows the transition state after each action.

d) The number of states needed to solve each of the 4 puzzles mentioned using both the representations is same. As explained above, for the representations I picked, total number of states and the number of feasible states are the same. Like discussed above, representation 1 definitely finds the solution path in less computations but representation 2 is much better in overall demonstration. However, given that the board size is just 5x5, even though representation 1 does some less computations than representation 2, the overall time both the representations take is almost similar, mostly because our graph is really small.

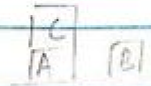
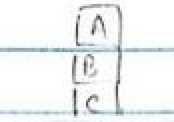
2) Mean Ends Analysis

a)

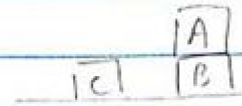
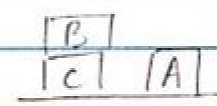
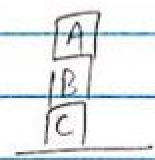
2 a)

Example where means-end analysis is not optimal:

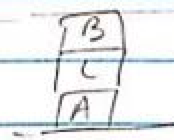
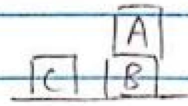
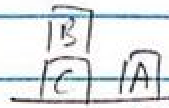
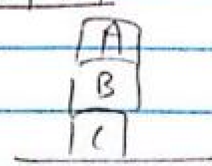
Note: This example was also discussed in class (The Sussman Anomaly)

Initial state:  Goal state: 

Linear Solution 1:

- (on A B)
    - unstack(C,A)
    - putdown(C) → 
    - stack(A,B)
- (on B C)
    - unstack(A,B)
    - putdown(A) → 
    - stack(B,C)
- (on A B)
    - stack(A,B) → 

Linear Solution 2:

- (on B C) → 
    - stack(B,C)
- (on A B)
    - unstack(B,C)
    - unstack(C,A) → 
    - stack(A,B)
- (on B C) → 
    - unstack(A,B)
    - stack(B,C) 
- (on A B) →
    - stack(A,B)

However, both the solutions discussed are not optimal
as the optimal solution is non-linear (discussed below)

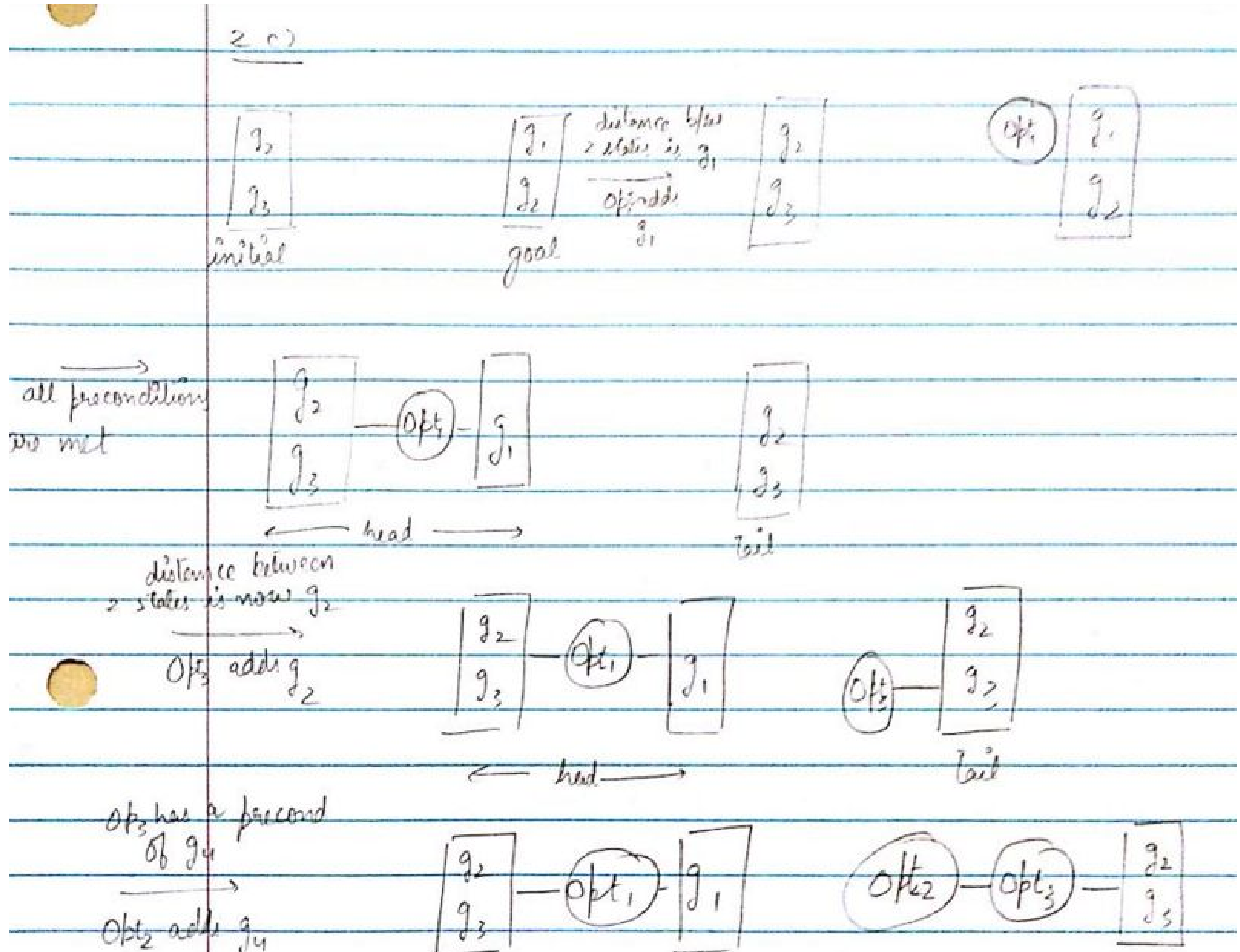
Non - Linear Solution:

- (On A, B)
  - Unstack (C, A)
  - Put down (C)
- (On B C) → Operation (On A,B) interrupted
  - Pickup (B)
  - Stack (B, C)
- (On A B) → Operation (On A B) continues after interruption and completes operation
  - Pickup (A)
  - Stack A, B

b) Solution plan for the given problem: Opt2 -> Opt1 -> Opt3.
Initial State: g2, g3.
After Opt2: g2, g3, g4.
After Opt1: g1, g4.
After Opt3: g1, g2, g4.

c)

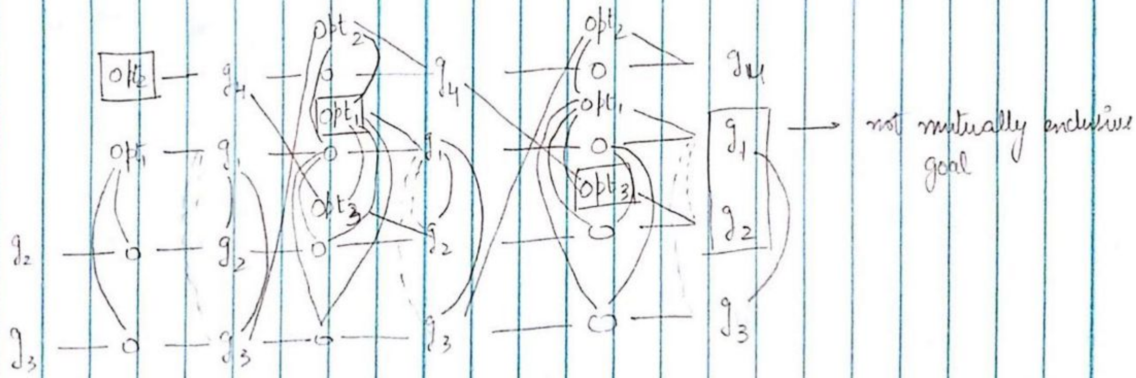
2 c)



As we can see, preconditions of Opt_2 are not met, hence we pop our last operations Opt_2, Opt_3, Opt_1 such that the list now becomes empty. At this point the prodigy algorithm with means end analysis fails to find a solution path and returns null

d)

2 d) Graph Plan to determine the solution path :

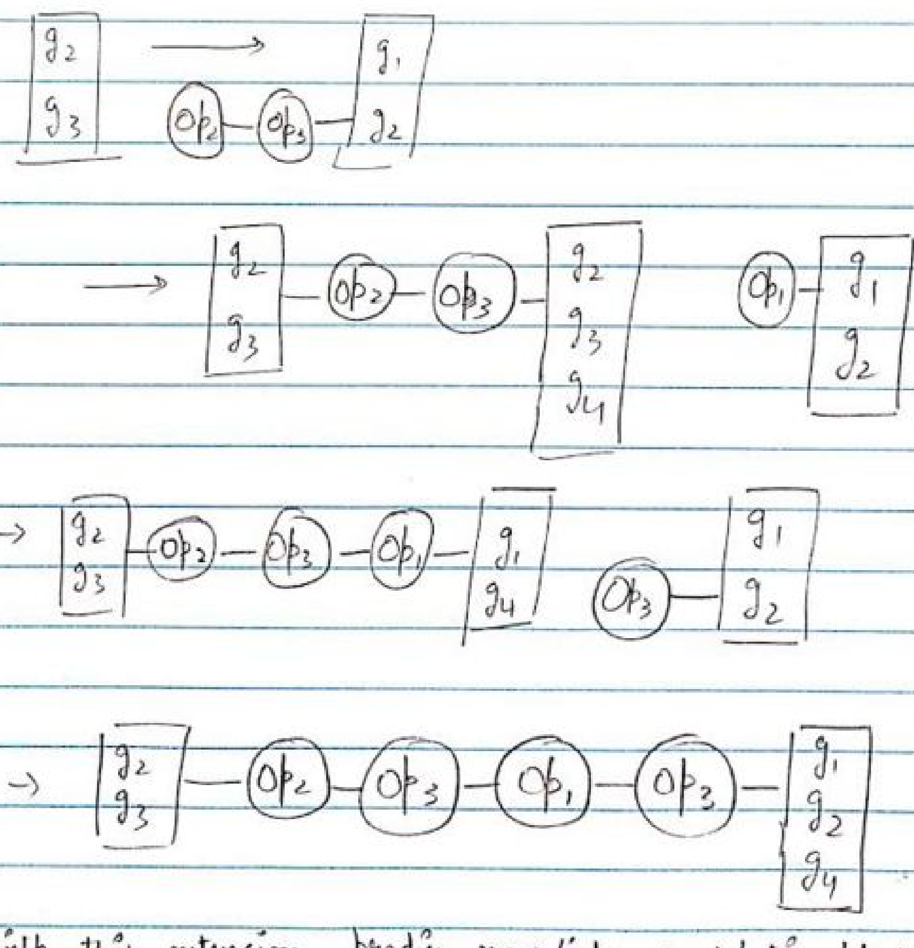


Solution plan for the given problem using graph plan : $Opt_2 \rightarrow Opt_1 \rightarrow Opt_3$

e)

2 e)

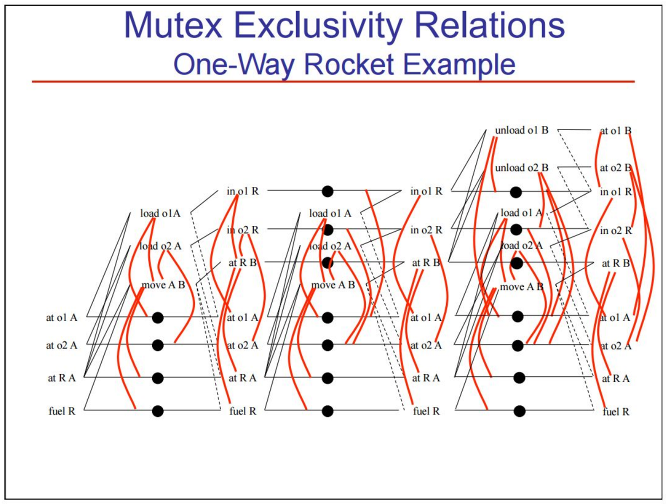
Prodigy always calculates the difference between the current and the goal state, initially and always expands the variable/conditions that are left after taking this distance. One possible extension to Prodigy is to expand all the variables/conditions in the goal state. For example, in 2c) we saw that prodigy with means end fails to find a solution as it only expands g_1 in goal state after taking the first difference. However, if we start with g_2 first:

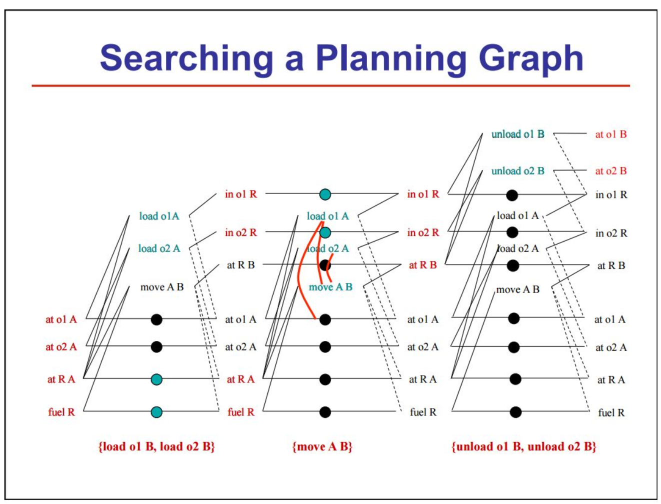


Hence, with this extension, prodigy now finds a solution plan

3) Graph Plan

a) As it is evident from the example below (covered in the class), while backwards search from the 2nd step to 1st step we see that the graph plan first selects no operations as actions to get to ‘in o1 R’, ‘in o2 R’, and ‘at R B’ states. But once it reaches the 1st step the planner realizes the states ‘in o1 R’, ‘in o2 R’, and ‘at R B’ are exclusive. Hence it backtracks and selects another set of operations which is no operations for ‘in o1 R’ and ‘in o2 R’ and selects ‘move AB’ for ‘at R B’. Hence, the graph-plan needs to backtrack during it’s backwards search to find the solution plan.





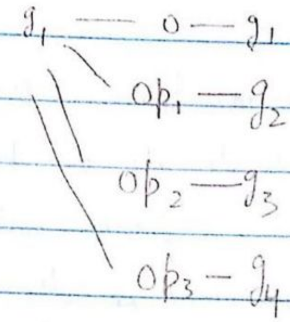
b) If a GraphPlan cannot find a solution during backwards search, that probably means that the forward search was not complete, and the forward search has not generated set of not exclusive goal states. In such a scenario, expanding the forward search to another time-stamp such that now the forward search generates non exclusive goal states will make the graph plan find a solution during backwards search. The problem is not solvable till the forward search finds a set of goal states that are non exclusive.

c)

3 c) Initial State : g_1

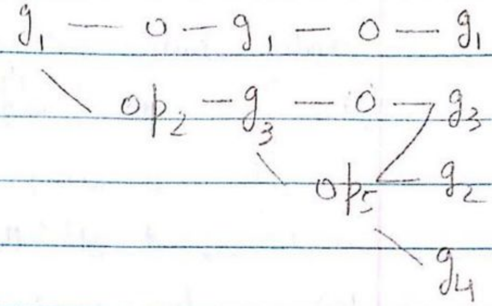
Goal State : g_1, g_2, g_3, g_4

(1)



1- time step, 3- operators

(2)



2 - time step, 2 - operators

In case (2) the precondition for Op_5 was met at time step 1. The graph plan will always generate solution plan as (1), but solution plan for (2) evidently has fewer operators.