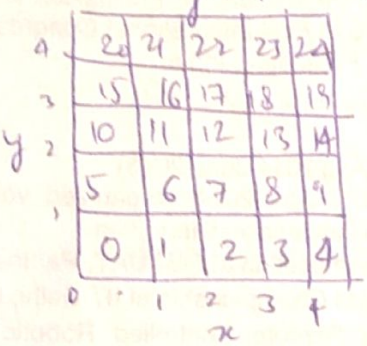
15-887 Homework 1

1. Lunar Lockout Game

a) Representation 1

Objects : 'r', 'y', 'o', 'b', 'g', 'B'
(Letter corresponding to color of spaceships)
Corresponding numerical ids have been used for calc.

States : ~~Data~~ Vector of node id corresponding to the grid (x, y) location for each object



(eg for a 5x5 grid)
eg Node id for loc (1,1) = 6

Action : Move (Object, Direction, State)
This action takes the object id and the direction to move for executing the movement.
~~Action pre cond~~ Pre : Can_Move (Object, Direction, State)
Add : At (Object, new_loc)
Can_Move (?Object, ?direction, new state)
Del : At (Object, prev_loc)
~~Can_Move (?Object, ?direction, new state)~~

~~Representation 2~~

Axioms : At (object, location, State)
Can_Move (Object, direction, state)
In_Board (~~Object~~ location, grid_size)
Is_Empty (location, state)

a) Representation 2

Objects : 1, 2, 3, 4, 5
(Number corresponding to spaceship)

States : 2D grid with 0 for empty cells and number corresponding to objects for filled cells.

Actions : Move (Object, Direction, State)
Description similar to Representation 1.

Axioms : Similar to Representation 1

b) Representation 1

Total possible states for a $n \times n$ grid with s objects

$= n^{2s}$

as each s can take n^2 values corresponding to locations

Representation 2

Total possible states for a $n \times n$ grid with s objects

$= (s+1)^{2n}$

as each of the n^2 cells can take s values and 1 empty value each.

Feasible states would be same as the actions and axioms are same for both. ~~Representation~~

Rep1 is favourable for ~~small~~ large state space as it is easier to use the stated axioms in this rep for large number of states.

Rep2 is favourable for ~~large~~ small state spaces.

d) Number of states for each problem:

Problem	Representation1	Representation2
9	36	36
18	53	53
27	1822	1822
36	1544	1544

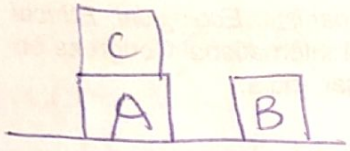
Since the number of feasible states are same for both reps due to similar axioms, it will take similar number of states to solve the problem.

15-887 Homework 1

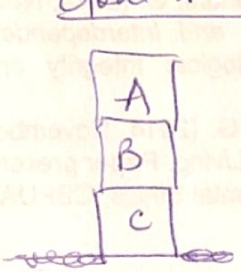
2. Means End Analysis

a) Sussman's Anomaly is a perfect example when MEA is not optimal (for GPS algorithm)

Initial state



Goal state



On(A, B)
On(B, C)

Actions -

a) Pickup(?b)
Pre: (handempty)
(clear ?b)
(on-table ?b)
Add: (holding ?b)
Delete: (handempty)
(on-table ?b)
(clear ?b)

b) Unstack(?a, ?b)
Pre: (handempty)
(clear ?a)(on ?a ?b)
Add: (holding ?a)(clear ?b)
Delete: (handempty)
(on ?a ?b) clear ?a)

c) Putdown(?b)
Pre: (holding ?b)
Add: (handempty)(on-table ?b)
Delete: (holding ?b)

d) Stack(?a, ?b)
Pre: (holding ?a)
(clear ?b)
Add: (handempty)
(on ?a ?b)
Delete: (holding ?a)
(clear ?b)

The GPS (linear) solution for this problem will vary with the order in which we chose the goals. This means that the optimality of GPS varies with the order of goals

Linear Solution

Chosing On(B,C) first

- on(B C)
  - Pickup(B)
  - Stack(B, C)
- on(A, B)
  - Unstack(B, C)
  - Putdown(B)
  - Unstack(C, A)
  - Putdown(C)
  - stack(A, B)
- on(B, C)
  - Unstack(A, B)
  - Putdown(A)
  - Pickup(B)
  - Stack(B, C)
- on(A, B)
  - Pickup(A)
  - Stack(A, B)

Chosing On(A,B) first

- on(A, B)
  - Unstack(C, A)
  - Putdown(C)
  - Stack(A, B)
- on(B, C)
  - Unstack(A, B)
  - Putdown(A)
  - Pickup(B)
  - Stack(B, C)
- on(A, B)
  - Pickup(A)
  - Stack(A, B)

It can be clearly seen that chosing On(A, B) first achieves the goal faster.

Non-linear Solution

- on(A, B)
  - Unstack(C, A)
  - Putdown(C)
- on(B, C)
  - Pickup(B)
  - Stack(B, C)
- on(A, B)
  - Pickup(A)
  - Stack(A, B)

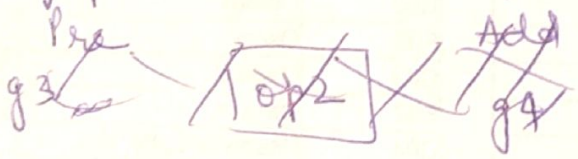
As can be seen, non-linear solution is optimal for the same problem.

(b)

	op1	op2	op3
pre	–	g3	g4
add	g1	g4	g2
del	g2, g3	–	–

Initial state : g2, g3 goal state : g1, g2

Following plan solves the problem :



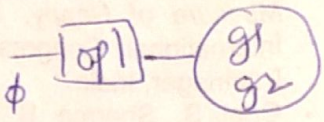
	State	Plan	Goal
t=0	g2 g3		g1 g2
t=1	g2 g3 g4	pre → op2 add	g1 g2
t=2	~~g2~~ del ~~g3~~ del g4 g1	op1 add	g1 g2
t=3	(g1) g4 (g2)	pre → op3 add	g1 g2

found solution

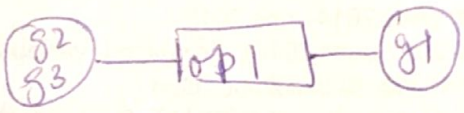
(c) Head Plan

Tail Plan

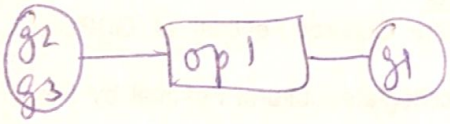
Since, g2 is already in the initial state, we start by adding actions to the tail plan that fulfill g1.

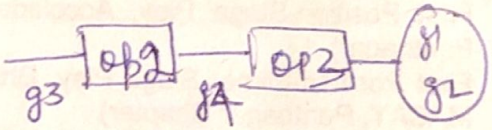


Since, op1 has no preconditions, we add it to head plan.



Now the state has only g1 & The goal state has now g1 achieved. So we will add operators to achieve g2.

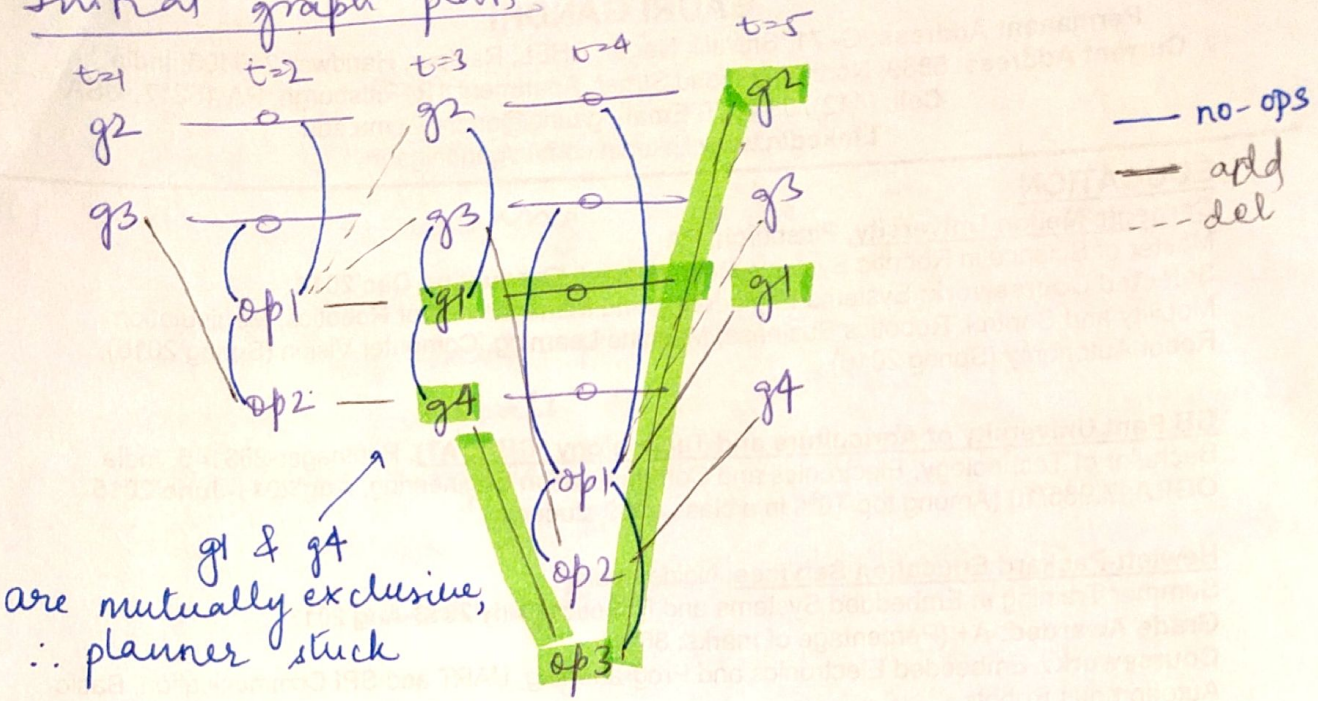




Since op3 has precondition g4 which is fullfilled by op2, we add both op2 & op3 to tail plan. But the precondition for op2 is g3 which is not achievable in the current state or by any operator. Hence we are unable to achieve g2 as the plan is incomplete with this order of actions.

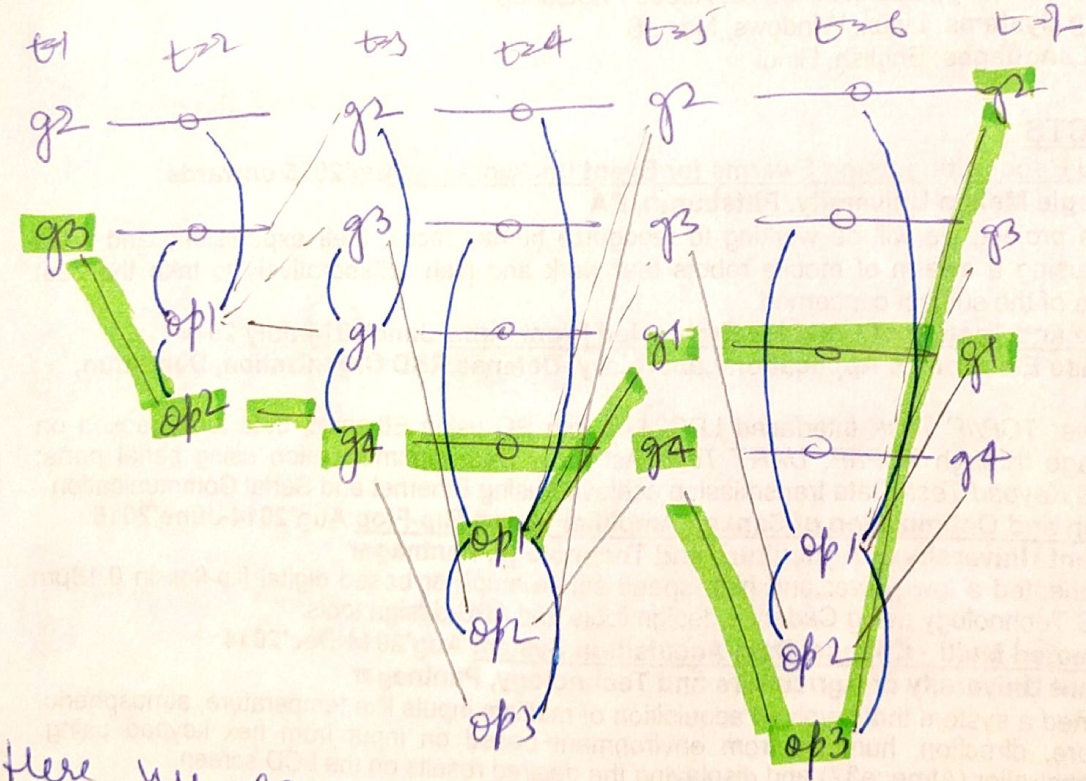
Prodigy is incomplete in such situations as it starts from goal set and does not consider alternate paths that were possible with forward tracking. ~~Prodigy does not explore the complete state space~~. It does not look at the preconditions of goals already in initial state.

(d) Initial graph plan



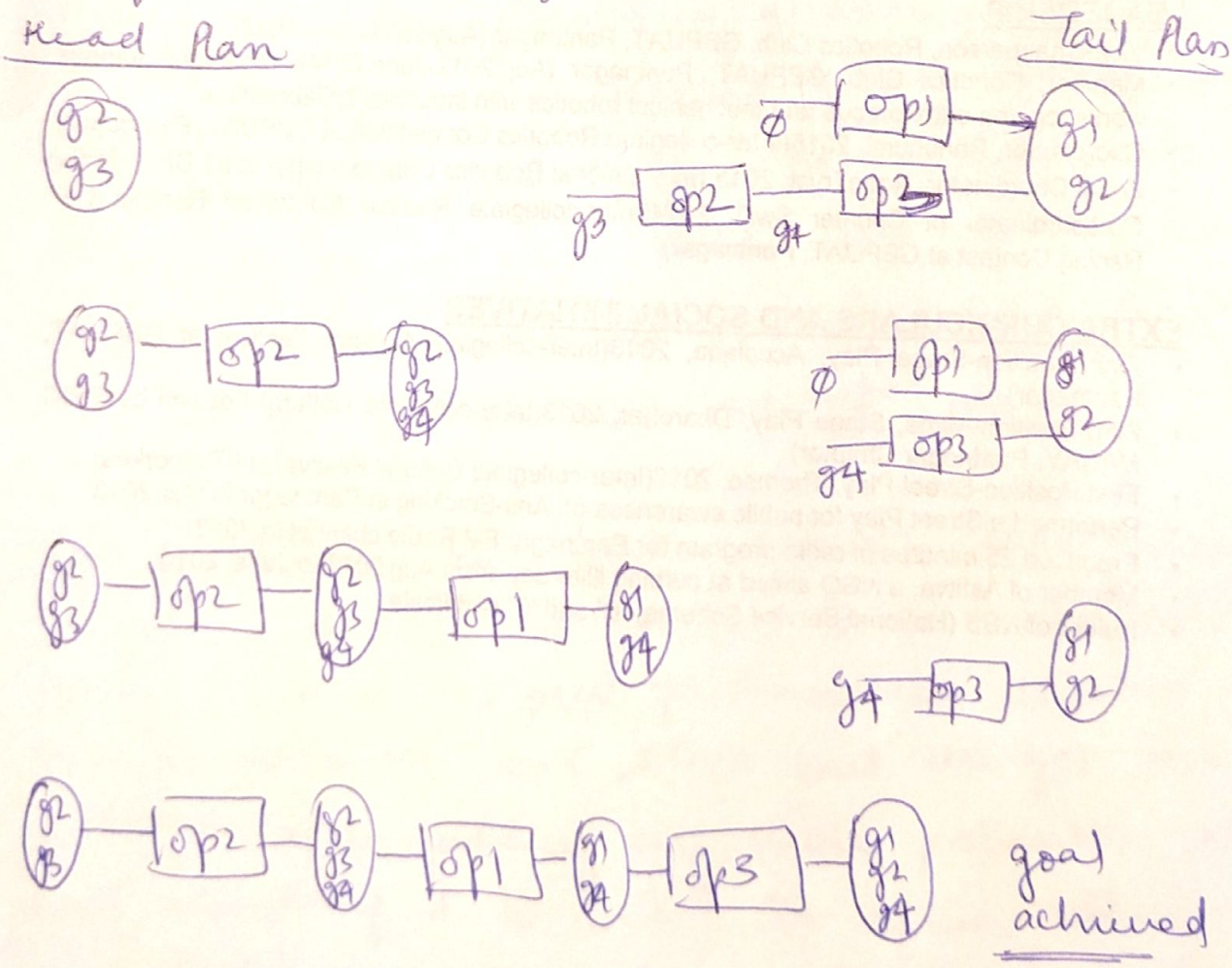
Extended Graph plan

Now we another action & predicate level to find solution.



Here we can see, we perform op2 with g3 as pre-condition, then perform op1, and then op3 with g4 as precondition.

(e) The prodigy planner got trapped in the given example and thus became incomplete. To avoid such traps, prodigy can be extended by adding operators for achieving literals that are already true in the current state and have not been linked to any tail-plan operators. Such literals are anycase subgoals that can be identified as a precondition or a top level goal literal that are negated by an application at some point in the search.
(References: "Prodigy Bidirectional Planning", Fink E., Blythe J.)



3. Graphplan

a) Graphplan needs to backtrack during the backward search, so that if at some point in the search, the planner selects 2 non-mutex actions whose pre-conditions are mutex, it can go back to select another set of non-mutex actions that result in feasible (non-mutex) goals.

for eg, in the previous question's graphplan, if at t=4, we would have chosen no-op action for g1 and op2, the result would be g1 and g3 at t=3 that were mutex. Hence we backtrack at t=4 and select op1 and no-op for g4 so that we get g4 at t=3.

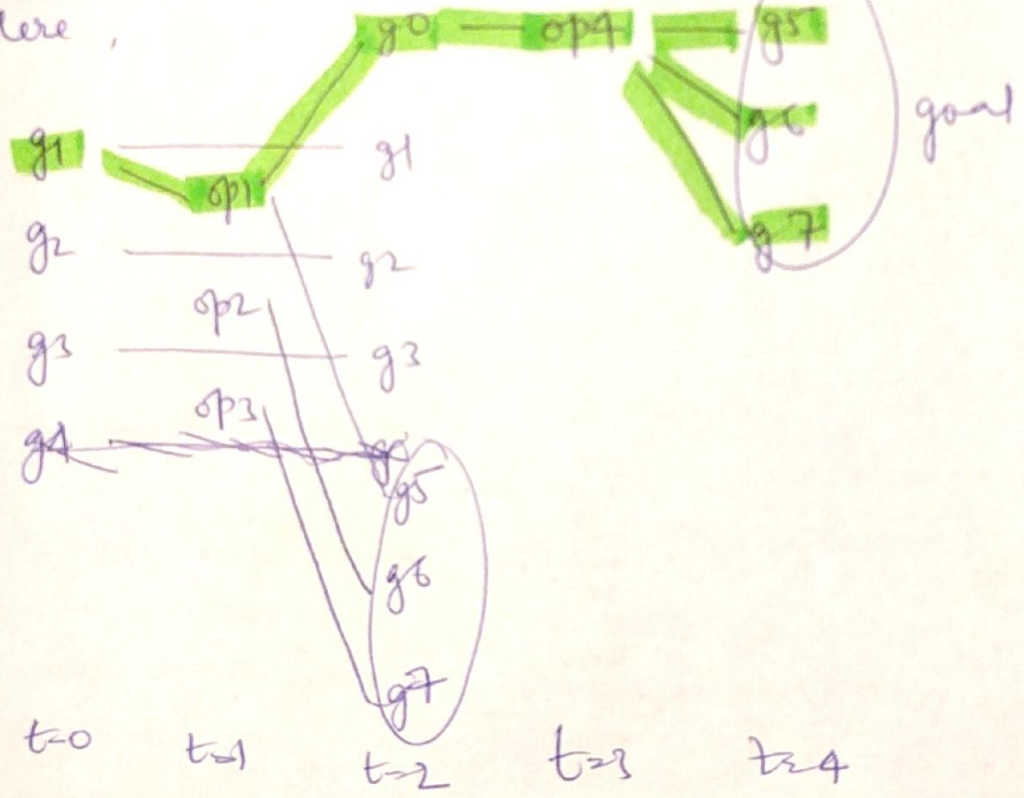
(b) If Graphplan cannot find a solution during backwards search, it does not mean it is unsolvable. For eg, the in the prev question, the solution was unachievable in t=5 steps but we got the solution after adding one more proposition level and extending it to t=7. If after adding the new level, we are still unable to find the solution, the problem is unsolvable.

(c) Since, graph plan optimizes in terms of number of time steps and not in terms of actions, it is possible to find a plan with k operators in n time steps and ~~k~~ <k operator in n+1 time steps -

For eg - Initial → g_1, g_2, g_3 Goal → g_5, g_6, g_7

	op1	op2	op3	op4
pre	g_1	g_2	g_3	g_0
add	g_5, g_0	g_6	g_7	g_5, g_6, g_7
del	-	-	-	-

Here,



Here we see, in t=2 time steps, total operators used are 3 to achieve goal.
in t=4 timesteps, total operators used are 2 to achieve goal.