1.

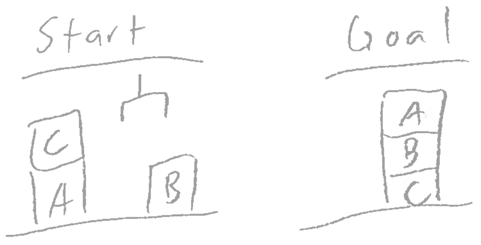
a) Unfortunately, the 2 representations are not very different. Since the game has a restricted set of actions, and coordinates are necessary to represent states, there are not a lot of options.
    i) Representation 1
        1) Objects: Ships with coordinates
        2) States: A combination of ships, basically a combination of coordinates
        3) Actions: Move (left, right, up, down)
        4) Domain Axioms: Actions are constrained such that the endpoint of a movement must ensure that a ship is next to another ship (their coordinates differ by 1 in either dimension)
    ii) Representation 2
        1) Objects: a 5x5 grid
        2) States: each cell in the grid is either 0, or 1-n (n is number of ships)
        3) Actions: Each non-zero cell can move left, right, up, or down
        4) Domain Axioms: During movement, we have to check if each cell is free. Also, we have the concept of blank cells.

b) Both representations will have the same number of feasible states since we enforce this during action execution. In terms of computational resources, the 1st representation would be preferable since it doesn't require maintaining an entire grid. Additionally, it can easily be extended to an nxn dimension puzzle.
    i) Representation 1: In the general case, there will be (n^2)*s states (n = dimension, s = number of ships)
    ii) Representation 2: (s+1)^(n^2) states

c) See README

d) Although there is a difference in the number of states explored before finding a solution, the representations would inherently perform the same because of the way we generate successor states. Actions are constrained by the rules of the game so both representations will have the same successors. The difference here is due to the fact that we are using Depth First Search and the order in which successors are added are not necessary the same for both representations.
    i) Representation 1
        1) Puzzle 9: 12
        2) Puzzle 18: 25
        3) Puzzle 27: 838
        4) Puzzle 36: 14
    ii) Representation 2
        1) Puzzle 9: 20
        2) Puzzle 18: 9
        3) Puzzle 27: 841
        4) Puzzle 36: 1389

2. a) Sussman's anomaly is one example



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)

Linear Solution

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

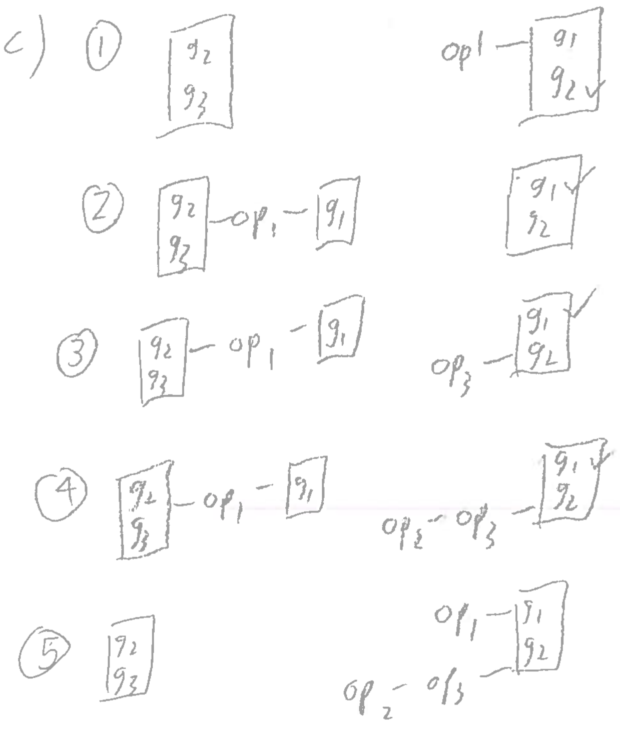
→ (on B C)
 - unstack (A,B)
 - putdown (A)
 - pickup (B)
 - stack (B,C)

→ (on A B)
 - pickup (A)
 - stack (A,B)

* with the same goal ordering, the linear solution performs worse

b) $\boxed{op2, op1, op3}$

c)



- choose g1 and op1 to satisfy g1

- apply op1, now g1 is satisfied but g2 needs to be worked on

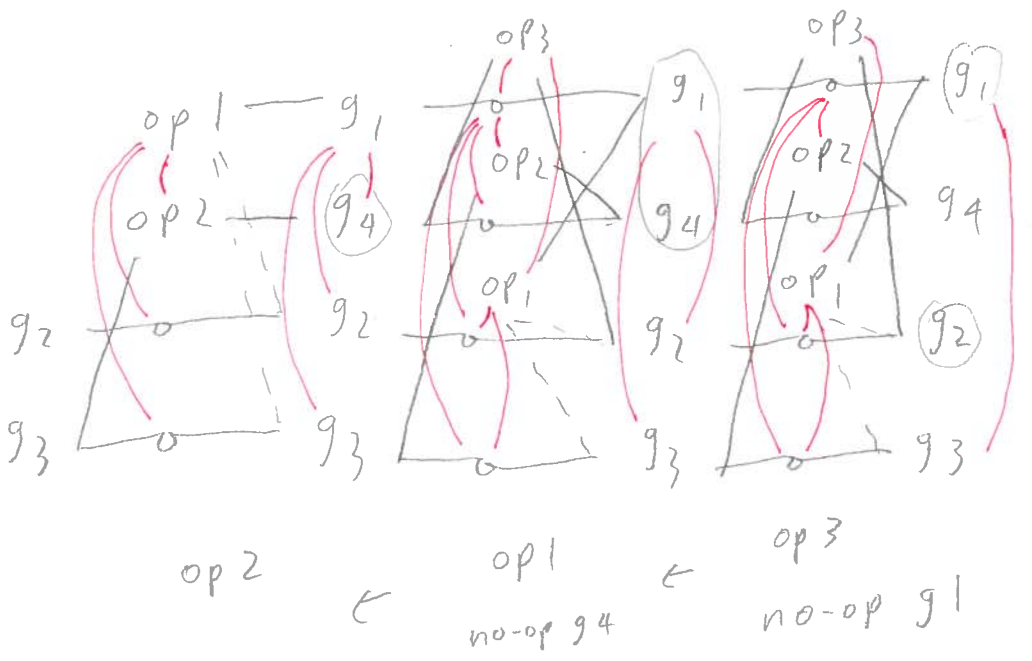
- select op3 to satisfy g_2

- select op2 to satisfy precondition for op3

- pop op1 back since nothing can satisfy op2

- at this point, we terminate with failure since we don't work on g2 (already satisfied), and we can't apply op1 since we just popped it back.

d)



⇒ op2, op1, op3

e) Prodigy could be modified so that we try operators on the tail plan which we haven't tried even if it doesn't agree with Means Ends Analysis.

3.

a) From the one-way rocket example from class, the backwards search actually encounters two possible sets of actions in the 2nd time step. It can choose either {Move (A,B), no-op in o1 R, no-op in o2 R} or {load (o1, A), load (o2, A), no-op at R B}. If we choose the 2nd set of actions, we realize that it's not possible to satisfy the preconditions at the 1st time step. Thus, GraphPlan must backtrack and select the 1st set.
b) Not necessarily. This is because he problem may have goal statements which are exclusive or may not be possible to be defined well in the STRIPS domain. Some actions may modify or create objects which will not be known statically at plan time. One solution is that we can try to keep track of these objects dynamically, but in general, it may be possible that infinitely many objects can be created.
c) In the one-way rocket example, the solution uses 5 operators in 3 time steps. We can make up some sequence of actions, such as a situation where we spend 4 time steps (with one action at each step) to find a wizard who can teleport both our objects to location B. This happens because GraphPlan will find the shortest solution in terms of time steps. Once the goal can be reached at some level, it will stop searching and return the first solution it can find, without further expansion.