# 15-381: Artificial Intelligence
# Assignment 4: Planning

Sample Solution

November 5, 2001

1. Consider a robot domain as shown in Figure 1. The domain consists a house that belongs to Pat, who has a robot-butler. Initially, Pat and the robot are in the living room, and there is an orange in the kitchen. The door between the family room and the kitchen is closed, while all other doors are open. The robot can perform the following actions with the associated costs:

```
go: Go through an open door to an adjacent room; cost: 2.
open-door: Open a door (when in the same room); cost: 3.
pick: Pick an orange (when in the same room); cost: 1.
```

   (a) Consider the task of sending the robot to the kitchen to pick the orange. An $A^*$ search was performed resulting in the search space shown in Figure 2. Choose an admissible heuristic, state it, and fill in the missing values for h and f, so that nodes 1 through 5 were expanded in that order. NOTE: Your heuristic cannot be 0 or some other constant value.

   Let $\delta$ be the shortest distance from the current position to the kitchen, assuming no doors are closed. Use following heuristic:

$$h = \begin{cases} 2\delta + 1 & \text{if orange has not been picked up} \\ 2\delta & \text{otherwise} \end{cases}$$

   The heuristic is admissible because it always underestimates the cost of moving from one room to another. If there is an open door between two rooms, the

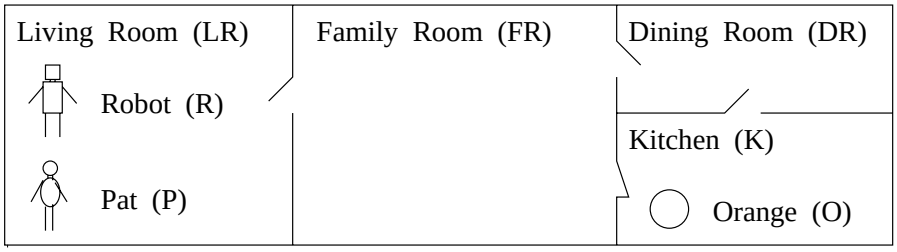



Figure 1: The robot-butler domain.

heuristic estimate is equal to the actual cost. If there is a closed door between two rooms, the cost of moving to the room on the other side of the door is at least 4 (by moving through another room first), and the heuristic estimate is only 2. Also, if the orange has not been picked up yet, it needs to be done eventually, so that will add a cost of 1.

(b) What is the solution found? Is this an optimal solution? Why or why not?

The solution is: go(LR,FR), go(FR,DR), go(DR,K), pick(O). It is optimal because the heuristic used with A* was admissible.

(c) Suppose that the robot has executed the plan of Part (b), and now it is in the kitchen with the orange. Consider the task of carrying the orange to the living room (in order to give it to Pat). Show the optimal plan for executing this task. (You do *not* need to show the search space this time, only the resulting plan.)

The solution is: go(K, DR), go(DR,FR), go(FR,LR).

(d) Now consider again the initial situation, shown in Figure 1, and suppose that the robot knows *both* goals, i.e. picking up the orange and bringing it to Pat, from the beginning. What is the optimal plan for achieving these two goals?

The solution is: go(LR,FR), open-door(FR,K), go(FR,K), pick(O), go(K,FR), go(FR,LR).

(e) How does the plan found in (d) differ from the plans found in Parts (b) and (c)? Can you suggest any search algorithm that would find this optimal plan? Is it possible to use $A^*$ (with an admissible heuristic) to perform this search?

The plan in (d) opens the door between the family room and the kitchen, and therefore avoids going through the dining room twice. The cost of the plan in (d) is 12, while the combined cost of the plans in (b) and (c) is 13. We can still use A* to find the optimal plan in (d), for example by using the heuristic $h = 0$. It is possible to construct more informed heuristics as well.

(f) Compare the efficiency of searching for the overall optimal plan given the two goals initially, as stated in part (d), with the efficiency of search in Parts (a) and (b). What conclusion can you make about the difficulty of finding the optimal plan for achieving multiple conjunctive goals?

While the branching factor $b$ in all cases are the same, the length of the overall optimal plan is likely to be longer than the length of the plans for each individual subgoal. The complexity is $O(b^d_{tot})$ for finding the overall optimal plan, but just $O(kb^d_{sub})$ for finding $k$ plans when treating subgoals independently. The former dominates the latter in most cases, so it is harder to find the overall optimal plan. By working on subgoals independently we can find a solution faster, but might have to sacrifice optimality.

2. Consider the following four operators (cf. page 346 in the textbook):

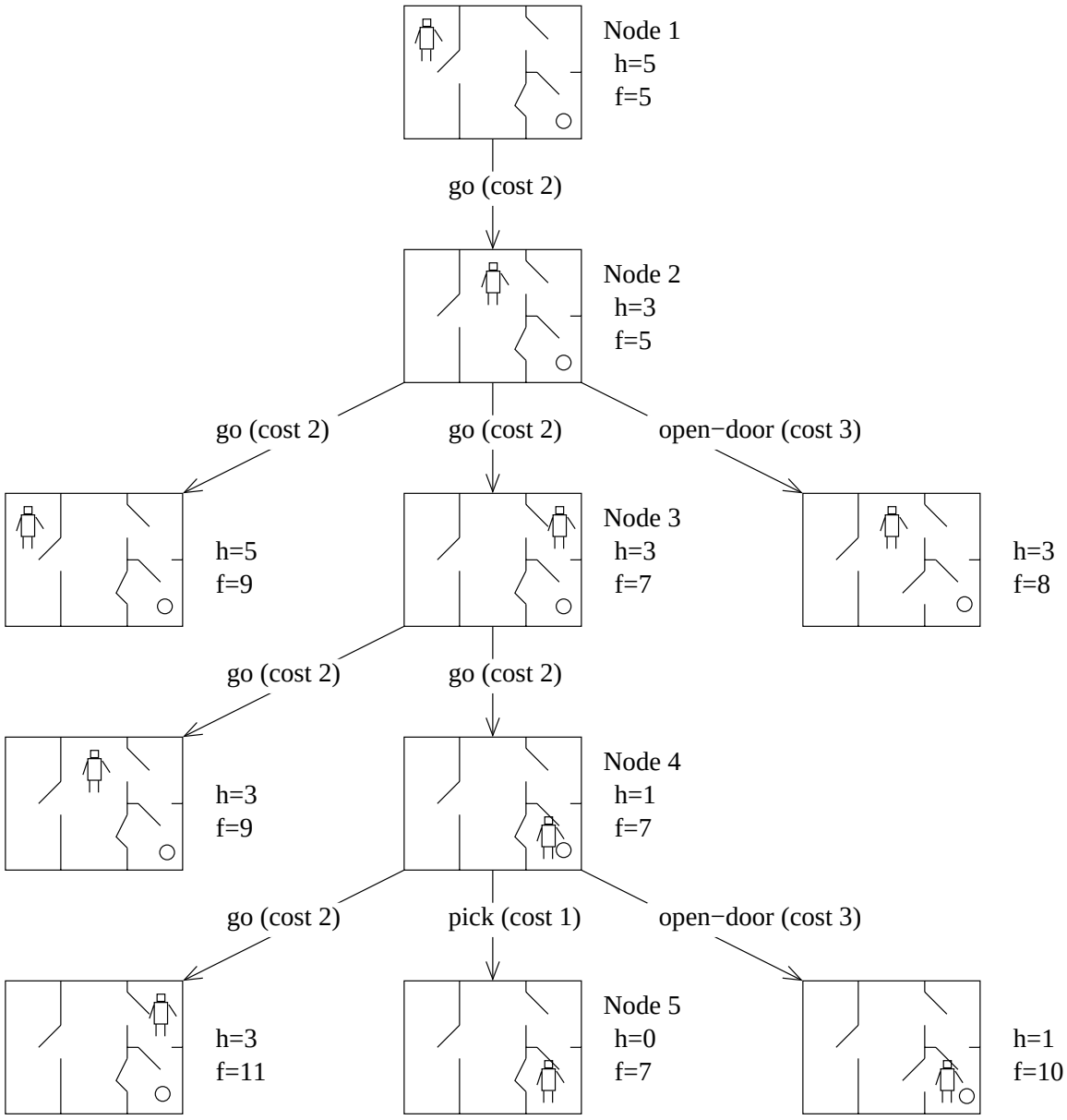


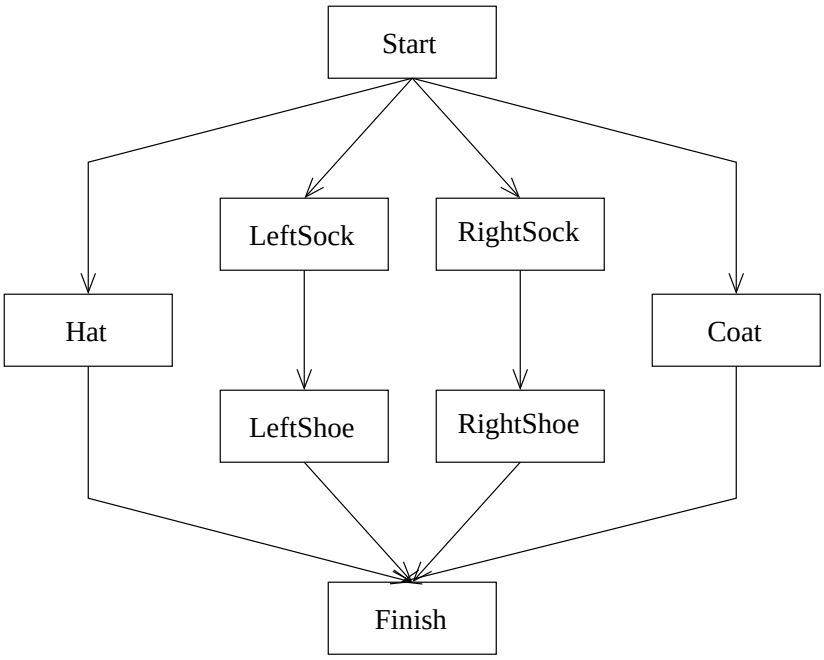

Figure 2: Search space for the robot-butler domain.

*Operators*

| | RightShoe | RightSock | LeftShoe | LeftSock | Hat | Coat |
|---|---|---|---|---|---|---|
| *preconds*: | RightSockOn | - | LeftSockOn | - | - | - |
| *adds*: | RightShoeOn | RightSockOn | LeftShoeOn | LeftSockOn | HatOn | CoatOn |
| *deletes*: | - | - | - | - | - | - |

Define additional operators for putting a hat and a coat on, respectively, assuming that there are no preconditions for putting on the hat and the coat. Give a partial-order plan that is a solution, and show that there are 180 different linearizations of this solution.

Partial order plan:




There are $6! = 720$ possible combinations of the six steps in the plan. Half of these will have the LeftShoe step before the LeftSock step, leaving $360$ steps not violating the ordering constraint between those two steps. Furthermore, half of the remaining plans will have the RightShoe step before the RightSock step, leaving $180$ steps not violating any ordering constraints.

3. The POP algorithm in the textbook is a regression planner, because it adds steps whose effects satisfy unsatisfied conditions in the plan. Progression planners add steps whose preconditions are satisfied by conditions known to be true in the plan. Modify POP so that it works as a progression planner.

Eliminate SELECT-SUBGOAL, and instead make CHOOSE-OPERATOR choose from the operators that have all their preconditions achieved by existing steps in the partial plan.

4. POP is a nondeterministic algorithm, and has a choice about which operator to add to the plan at each step and how to resolve each threat. Can you think of any domain-independent heuristics for ordering these choices that are likely to improve POP's efficiency?

   One simple heuristic is to minimize the number of unlinked preconditions in a plan at any time. We can do this by preferring steps that achieve preconditions without adding many new ones. A more sophisticated heuristic would try to estimate the number of steps that would have to be added to a plan in order to achieve all preconditions. A very effective such estimate is obtained by ignoring delete lists and positive interaction. Let the cost of an unlinked precondition be 0 if is is achieved by the start step. Otherwise, let the cost of the preconditions be the cost of the operator with the cheapest preconditions. The cost of a set of preconditions is defined to be the sum of the costs of the individual preconditions in the set. Other components of a heuristic could be the number of ordering constrains in a plan (the fewer ordering constraints the better) or the number of threats (the fewer threats the better).

5. Consider the following operators to load and unload objects into and from containers at and to some locations, to move containers between locations, and to get any new container at any location:

   Notice that any list of literals represents a conjunction (e.g., the list of preconditions and the statements of the initial state and the goal).

| | *Operators* | | | |
|---|---|---|---|---|
| | LOAD(o,c,l) | MOVE(c,l1,l2) | UNLOAD(o,c) | GET-NEW(c,l) |
| *preconds*: | At-Container(c,l)<br>Space-Available(c)<br>At-Obj(o,l) | At-Container(c,l1) | Inside (o,c)<br>At-Container(c,l) | - |
| *adds*: | Inside(o,c) | At-Container(c,l2) | At-Obj(o,l) | At-Container(c,l)<br>Space-Available(c) |
| *deletes*: | At-Obj(o,l) | At-Container(c,l1) | Inside(o,c) | - |

   (a) Consider the following problem:

   Initial State: At-Obj(o1,l1), At-Obj(o2,l1), At-Container(c1,l1), Space-Available(c1).

   Goal State: At-Obj(o1,l2), At-Obj(o2,l2).

   Pat claims:"There is more than one plan that can solve this problem." Is Pat correct? If yes, then show at least two plans to solve this problem. Otherwise, justify why Pat is incorrect.

   Pat is correct. Two different plans:

| | |
|---|---|
| LOAD(o1,c1,l1) | LOAD(o1,c1,l1) |
| LOAD(o2,c1,l1) | MOVE(c1,l1,l2) |
| MOVE(c1,l1,l2) | UNLOAD(o1,c1) |
| UNLOAD(o1,c1) | MOVE(c1,l2,l1) |
| UNLOAD(o2,c1) | LOAD(o2,c1,l1) |
| | MOVE(c1,l1,l2) |
| | UNLOAD(o2,c1) |

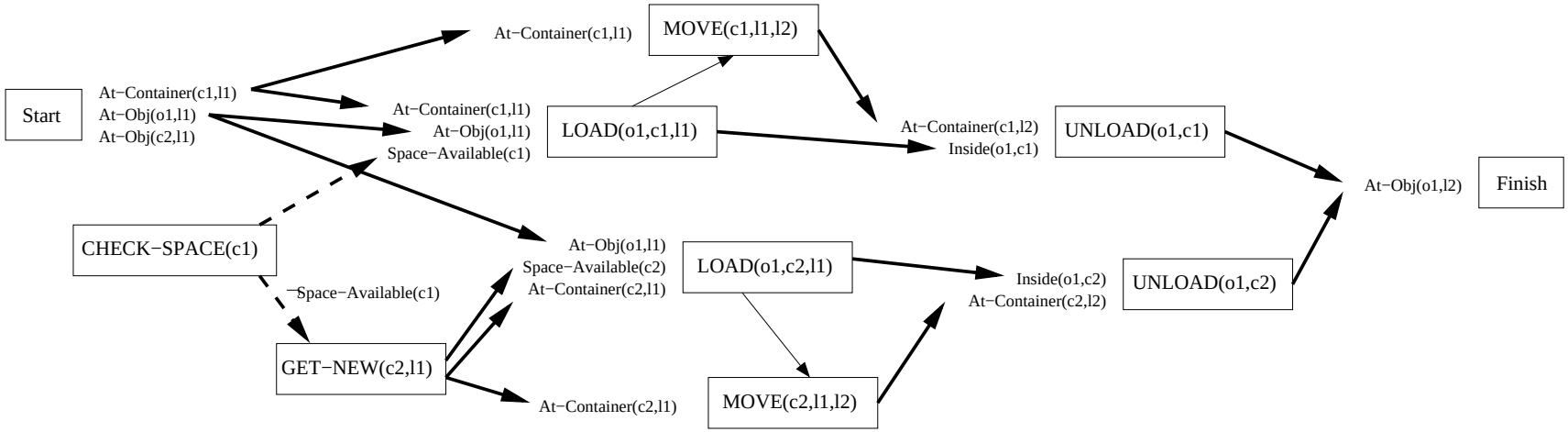
(b) Suppose now that Space-Available is not known in the initial state. Consider that you have available an additional operator, CHECK-SPACE, that allows you to check if there is Space-Available. That operator returns Space-Available(c) or ¬ Space-Available(c).

Show a conditional plan to solve the following simple problem:

Initial State: At-Obj(o1,l1), At-Obj(o2,l1), At-Container(c1,l1).

Goal State: At-Obj(o1,l2).

Conditional plan:


Start
At−Container(c1,l1)
At−Obj(o1,l1)
At−Obj(c2,l1)
At−Container(c1,l1)
At−Container(c1,l1)
At−Obj(o1,l1)
Space−Available(c1)
MOVE(c1,l1,l2)
LOAD(o1,c1,l1)
At−Container(c1,l2)
Inside(o1,c1)
UNLOAD(o1,c1)
At−Obj(o1,l2)
Finish
CHECK−SPACE(c1)
−Space−Available(c1)
At−Obj(o1,l1)
Space−Available(c2)
At−Container(c2,l1)
LOAD(o1,c2,l1)
Inside(o1,c2)
At−Container(c2,l2)
UNLOAD(o1,c2)
GET−NEW(c2,l1)
At−Container(c2,l1)
MOVE(c2,l1,l2)

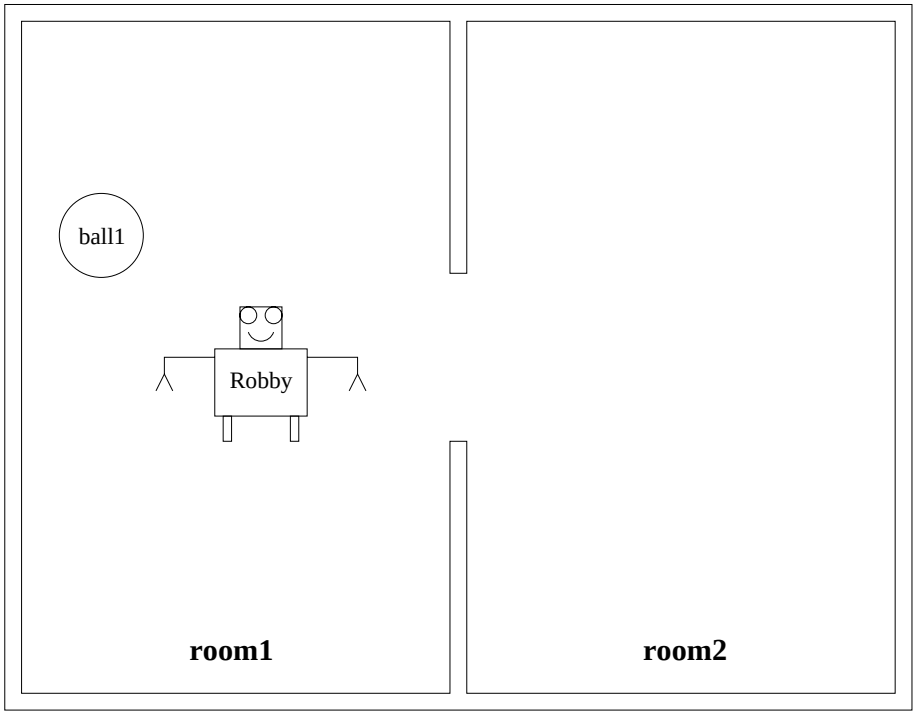



Figure 3: The gripper domain with one ball in room1.

6. Robby is a robot with two grippers—one left and one right. He can pick up and put down balls with the grippers, and each gripper can hold exactly one ball. The following operators represent these actions:

*Operators*

| | Pick(o,r,g) | Drop(o,r,g) | Move(r1, r2) | Start | Finish |
|---|---|---|---|---|---|
| *preconds*: | At(o,r)<br>At-Robby(r)<br>Free(g) | Carry(o,g)<br>At-Robby(r) | At-Robby(r1) | - | At(ball1,room2) |
| *adds*: | Carry(o,g) | At(o,r)<br>Free(g) | At-Robby(r2) | At(ball1,room1)<br>At-Robby(room1)<br>Free(left)<br>Free(right) | - |
| *deletes*: | At(o,r)<br>Free(g) | Carry(o,g) | At-Robby(r1) | - | - |

Robby can also move between rooms. This way he can pick up balls in one room and drop them off in another room.

(a) Write an operator `move`, representing the action of moving from one room to another.

Now consider the situation depicted in Figure 3. There are two rooms (`room1`, and `room2`). Robby is in `room1`, and both his grippers are empty. There is only one ball (`ball1`), and it is also in `room1`. The goal is to move `ball1` from `room1` to `room2`.

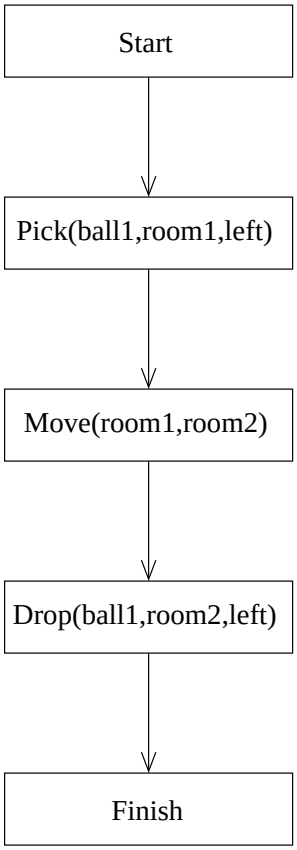
(b) Specify the Start and Finish operators representing the initial situation and the goal, respectively.

(c) Specify all *consistent* instantiations (with all parameters substituted for atoms) of each of the three operators.

Pick(`ball1`,`room1`,`left`)
Pick(`ball1`,`room2`,`left`)
Pick(`ball1`,`room1`,`right`)
Pick(`ball1`,`room2`,`right`)
Move(`room1`,`room2`)
Move(`room2`,`room1`)
Drop(`ball1`,`room1`,`left`)
Drop(`ball1`,`room2`,`left`)
Drop(`ball1`,`room1`,`right`)
Drop(`ball1`,`room2`,`right`)

(d) Give a partial-order plan that is a solution to the stated problem. How many linearizations exist for the plan?

Partial order plan:

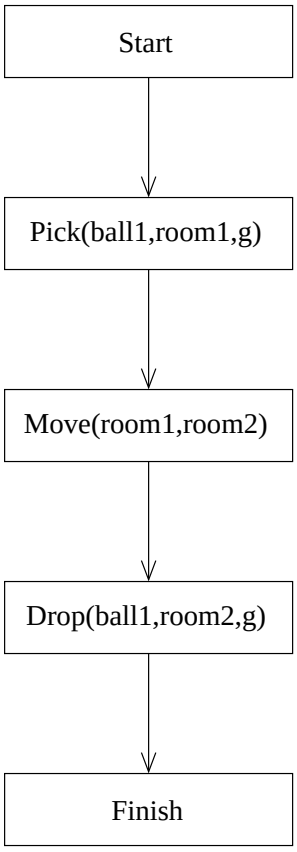



There is only one possible linearization (the plan steps are already totally ordered).

(e) The principle of least-commitment says that a planner should avoid making decisions until there is a good reason to make a choice. Give a least-commitment plan that is a

solution to the stated problem. How many fully instantiated plans can be constructed from this plan?

Least-commitment plan:




There are two possible instantiations (g = `left`, or g = `right`).