

## COMP3620/6320 – Planning Tutorial

Consider the following simple version of the Dock Worker Robot (DWR) domain. There is a map of *locations* that are connected pairwise. Each location can store any number of *boxes*. All boxes at a location lie on the floor instead of being stacked up on top of each other. A *robot* can carry at most one box at a time and can *move* between any two locations. In addition to robot movement, two more actions are defined. A *load* action places a box on a robot's platform, assuming that both the robot and the box are at the same location and that the robot's platform is empty. An *unload* action takes a box from a robot's platform and places it at the current location of the robot. In this simple application, loading and unloading boxes do not require the presence of a hoist.

Consider a problem instance with two locations LOC-0 and LOC-1, two boxes BOX-0 and BOX-1, and one robot ROBOT-0. Initially, both the robot and the two boxes are at location LOC-0. The goal is to have the two boxes at location LOC-1 and the robot at location LOC-0.

1. Write (part of) a STRIPS representation of the domain and the problem. Write the types, the predicates, and the operators.

*Answer:*

Domain:

```
(:types robot box location)

(:predicates
  (robot-at ?r - robot ?l - location)
  (box-at ?b - box ?l - location)
  (in ?b - box ?r- robot)
  (empty ?r - robot)
)

(:action Move
  :parameters (?r - robot ?l1 - location ?l2 - location)
  :precondition ((robot-at ?r ?l1))
  :effect (and (not (robot-at ?r ?l1))
               (robot-at ?r ?l2))
)
```

```

(:action Load
  :parameters (?r - robot ?b - box ?l - location)
  :precondition (and (robot-at ?r ?l)
                    (box-at ?b ?l)
                    (empty ?r))
  :effect (and (not (box-at ?b ?l))
              not (empty ?r)
              (in ?b ?r))
)

(:action Unload
  :parameters (?r - robot ?b - box ?l - location)
  :precondition (and (robot-at ?r ?l) (in ?b ?r))
  :effect (and (box-at ?b ?l)
              (empty ?r)
              not (in ?b ?r))
)

```

**Problem:**

```

([skipping the header]
(:objects
  robot0 - robot
  loc0 loc1 - location
  box0 box1 - box
)

(:init
  (robot-at robot0 loc0)
  (box-at box0 loc0)
  (box-at box1 loc0)
  (empty robot0)
)

(:goal
  (and (robot-at robot0 loc0)
       (box-at box0 loc1)
       (box-at box1 loc1))
)
)

```

2. Enumerate all ground actions that are generated for the problem at hand. For a few actions, show the preconditions and the effects.

*Answer:*

```

Move robot0 loc0 loc1
    :precondition ((robot-at robot0 loc0))
    :effect (and (not (robot-at robot0 loc0))
                 (robot-at robot0 loc1))
Move robot0 loc1 loc0
Load robot0 box0 loc0
Load robot0 box0 loc1
Load robot0 box1 loc0
Load robot0 box1 loc1
Unload robot0 box0 loc0
Unload robot0 box0 loc1
Unload robot0 box1 loc0
Unload robot0 box1 loc1

```

3. Starting from the initial state of the problem at hand, build a planning graph with three fact levels and two action levels. (Do not explicitly compute all mutexes.)

*Answer:* A picture would be the best answer, but it's hard to do it on the computer (it's much faster by hand). I try to explain it in words. The first level of facts is the initial state of the problem (the 4 atoms shown before). The first level of actions includes:

```

Move robot0 loc0 loc1
Load robot0 box0 loc0
Load robot0 box1 loc0

```

The second level of facts includes all facts in the previous level, plus:

```

(robot-at robot0 loc1)
(in box0 robot0)
(in box1 robot0)

```

The second level of moves includes all actions at the previous level, plus:

```

Move robot0 loc1 loc0
Unload robot0 box0 loc0
Unload robot0 box1 loc0
Unload robot0 box0 loc1
Unload robot0 box1 loc1

```

The third level of facts includes all facts in the previous level, plus:

```

(box-at box0 loc1)
(box-at box1 loc1)

```

In addition, each action level contains a set of NOOPs, one for each fact at the previous fact level.

4. Is this number of levels enough to extract a correct solution? *Answer:* No.
5. Is this number of levels enough to extract a relaxed solution? Here relaxation is obtained by ignoring all delete effects of actions. *Answer:* Yes.
6. Extract a relaxed solution from the planning graph.

*Answer:*

```
Load robot0 box0 loc0
Load robot0 box1 loc0
Move robot0 loc0 loc1
Unload robot0 box0 loc1
Unload robot0 box1 loc1
```

Indeed, if we ignore all the delete (negative) effects, then the sequence above is valid. The ignore-delete relaxation allows, among other things, to have the same box in two distinct places at a time, the robot to be in two places at a time, the robot to be both empty and loaded at a time, etc.

7. Write down a correct solution for this problem.

*Answer:*

```
Load robot0 box0 loc0
Move robot0 loc0 loc1
Unload robot0 box0 loc1
Move robot0 loc1 loc0
Load robot0 box1 loc0
Move robot0 loc0 loc1
Unload robot0 box1 loc1
Move robot0 loc1 loc0
```

8. Compare the relaxed solution with the correct solution. Look at the length and the actual actions included in each type of solution.

*Answer:* In this case, the relaxed plan is shorter than the real one. This is true in a majority of cases observed in practice, but situations where relaxed plans are longer are not very unusual either. Note that there are some common actions too. Again, there is no guarantee that some actions from the relaxed plan would be useful in a real plan as well, but this often happens in practice.

9. How can a relaxed plan be used in a search algorithm?

*Answer:* The most common usage is to take the length of the relaxed plan and use it as a (non-admissible) heuristic. Some recent work makes use of the actual contents of a relaxed plan too. The basic idea is to try to execute actions of the relaxed plan in the real world as long as this does not break the correctness of the search. For example, the following macro-action (i.e., sequence of actions) extracted from the relaxed plan can be applied to the initial state in the un-relaxed problem:

```
Load robot0 box0 loc0
Move robot0 loc0 loc1
Unload robot0 box0 loc1
```

The result is that we take longer steps in exploring the search space, and this can lead to finding a solution faster.

10. Show a few examples of fact mutexes that are permanent (i.e., they do not depend on the level of the planning graph).

*Answer:*

```
((robot-at robot0 loc0) (robot-at robot0 loc1))

((in box0 robot0) (empty robot0))
```