Mini-Project1

Michael Anderson, Tyler McClung

February 4, 2011

CS533

Prof. Fern

1 Playing with Blocks

1.
```
Block World A:
    Put(A,B):
        Pre: {clear(B), holding(A)}
        Add: {on(A,B), clear(A), clawEmpty}
        Del: {clear(B), holding(A)}

    Pickup(A):
        Pre: {clear(A), onTable(A), clawEmpty}
        Add: {holding(A)}
        Del: {clawEmpty, onTable(A)}

    Unstack(A,B):
        Pre: {clear(A), on(A,B), clawEmpty}
        Add: {holding(A), clear(B)}
        Del: {on(A,B)}
    Drop(A):
        Pre: {holding(A)}
        Add: {clawEmpty, onTable(A)}
        Del: {holding(A)}

Block World B:
    Put(A,B):
        Pre: {clear(B), holding(A)}
        Add: {clawEmpty}
        Del: {clear(B), holding(A)}

    Insert(A,B,C):
        Pre: {on(C,B), holding(A)}
        Add: {on(C,A), on(A,B), clawEmpty}
        Del: {on(C,B), holding(A)}

    Pickup(A):
        Pre: {clear(A), clawEmpty}
        Add: {holding(A)}
        Del: {clawEmpty}

    Unstack(A,B):
        Pre: {clear(A), on(A,B), clawEmpty}
        Add: {holding(A), clear(B)}
        Del: {on(A,B)}

    Drop(A):
        Pre: {holding(A)}
        Add: {clear(A), clawEmpty}
        Del: {holding(A)}
```

The only difference between the two definitions is that Block World B has an insert action which allows held blocks to be directly inserted

between two objects stacked on top of one another.

Block World B's insert action will allow for shorter solutions, but BlackBox (our choice of planners) will have to consider possible insert actions in every state. This will create search trees that are shallower, but also broader, than Block World A's.

2. *Easy:* None of the blocks are stacked on top of one another, which seems easy. To solve this problem, all the blocks must simply be stacked into a single tower in the right order from the initial position.

Medium: There are two towers that must be merged. In Block World A, both towers must be destroyed in order to construct the new one. In BlockWorldB, the blocks of one tower can be placed directly into the other.

Hard: In this problem, the blocks are positioned in such a way that every block must be moved at least once in order to reach the goal state. Normally, the more steps a problem requires to solve, the harder it is for Blackbox to find the solution quickly.

3. Block World A
Easy: 1.33 seconds, 18 actions
Medium: 0.7 seconds, 24 actions
Hard: Failure. 10 minute 31 seconds, 28 steps

The problem that we considered Medium was solved faster by Blackbox. This happened because Blackbox had to explore fewer actions, since many of the potential actions did not meet their preconditions (blocks being in a tower limit the number of possible actions).

Block World B
Easy: Failure
Medium: 1.71 seconds, 8 actions
Hard: Failure

It is important to note that while the solution lengths for most puzzles are shorter for the the second block world, the number of actions per iteration is much greater. This added breadth drastically increases the search time for longer solutions.

2 Create your own planning domain

1. Our traffic jam domain is based on the board game Rush Hour, http://eslus.com/Gizmos/rushour/rushHour.html. The object of the game is to move a goal vehicle to a specified edge of a 6 by 6 grid to leave a jammed parking lot. Several other obstacle vehicles block the path. Vehicles may only move vertically or horizontally in a straight line, and may take up either 2 or 3 squares on the grid.

2.
```
MoveRight2H(v, s1, s2, d):
    PRE: {VEHICLE_2H(v), at_2(v, s1, s2), empty(d), next_to_right(s2, d)}
    ADD: {empty(s1), at_2(v, s2, d)}
    DEL: {empty(d), at_2(v, s1, s2)}

MoveLeft2H(v, s1, s2, d):
    PRE: {VEHICLE_2H(v), at_2(v, s1, s2), empty(d), next_to_right(d, s1)
    ADD: {empty(s2), at_2(v, d, s1)}
    DEL: {empty(d), at_2(v, s1, s2)}

MoveRight3H (v, s1, s2, s3, d):
    PRE: {VEHICLE_3H(v),at_3(v, s1, s2, s3),  empty(d), next_to_right(s3, d)}
    ADD: {empty(s1), at_3(v,s2, s3, d)}
    DEL: {empty(d), at_3(v, s1, s2, s3)}

MoveLeft3H(v, s1, s2, s3, d):
    PRE: {VEHICLE_3H(v), at_3(v, s1, s2, s3), empty(d), next_to_right(d, s1)}
    ADD: {empty(s3), at_3(v, d, s1, s2)}
    DEL: {empty(d), at_3(v, s1, s2, s3)}

MoveUp2V(v, s1, s2, d):
    PRE: {VEHICLE_2V(v), at_2(v, s1, s2), empty(d), next_to_up(s1, d)}
    ADD: {empty(s2), at_2(v, d, s1)}
    DEL: {empty(d), at_2(v, s1, s2)}

MoveDown2V(v, s1, s2, d):
    PRE: {VEHICLE_2V(v), at_2(v, s1, s2), empty(d), next_to_up(d, s2)}
    ADD: {empty(s1), at_2(v, s2, d)}
    DEL: {empty(d), at_2(v, s1, s2)}

MoveUp3V(v, s1, s2, s3, d):
    PRE: {VEHICLE_3V(v), at_3(v, s1, s2, s3), empty(d), next_to_up(s1, d)}
    ADD: {empty(s3), at_3(v, d, s1, s2)}
    DEL: {empty(d), at_3(v, s1, s2, s3)}

MoveDown3V(v, s1, s2, s3, d):
    PRE: {VEHICLE_3V(v), at_3(v, s1, s2, s3), empty(d), next_to_up(d, s3)}
    ADD: {empty(s1), at_3(v, s2, s3, d) }
    DEL: {empty(d), at_3(v, s1, s2, s3)}
```

Each action corresponds to the movement of a vehicle object in the grid. In order to code this domain into STRIPS, we defined up and down movement for vehicles placed vertically on the grid, and left and right actions for vehicles placed horizontally on the grid. Additionally, separate actions needed to be defined for vehicles that took up 2 squares and vehicles that took up 3 squares.

3. Our easiest test problem was the following:

```
1 1 - - - 2
3 - - 4 - 2
3 0 0 4 - 2
3 - - 4 - -
5 - - - 6 6
5 - 7 7 7 -
```

Where $vehicle_1$ is represented by the 1's in the grid, $vehicle_2$ is represented by the 2's, etc. The goal is to get $vehicle_0$ to the right side of the grid, where the "exit" is. In this particular initial position it is blocked by vehicles 4 and 2.

Included with our submission is the python script proj1.py, which converts such a grid into the corresponding set of PDDL propositions that can be used as the facts file for BlackCox.

The solution, as given by BlackBox, to this problem is:

```
--------------------------------------------------
Begin plan
1 (moveleft2h v6 sq-4-4 sq-4-5 sq-4-3)
1 (moveright2h v1 sq-0-0 sq-0-1 sq-0-2)
1 (moveleft3h v7 sq-5-2 sq-5-3 sq-5-4 sq-5-1)
1 (movedown3v v2 sq-0-5 sq-1-5 sq-2-5 sq-3-5)
2 (movedown3v v2 sq-1-5 sq-2-5 sq-3-5 sq-4-5)
2 (moveleft2h v6 sq-4-3 sq-4-4 sq-4-2)
2 (moveup3v v3 sq-1-0 sq-2-0 sq-3-0 sq-0-0)
3 (movedown3v v2 sq-2-5 sq-3-5 sq-4-5 sq-5-5)
3 (moveleft2h v6 sq-4-2 sq-4-3 sq-4-1)
3 (moveup2v v5 sq-4-0 sq-5-0 sq-3-0)
4 (moveleft3h v7 sq-5-1 sq-5-2 sq-5-3 sq-5-0)
4 (movedown3v v4 sq-1-3 sq-2-3 sq-3-3 sq-4-3)
5 (movedown3v v4 sq-2-3 sq-3-3 sq-4-3 sq-5-3)
6 (moveright2h v0 sq-2-1 sq-2-2 sq-2-3)
7 (moveright2h v0 sq-2-2 sq-2-3 sq-2-4)
8 (moveright2h v0 sq-2-3 sq-2-4 sq-2-5)
End plan
--------------------------------------------------

16 total actions in plan
0 entries in hash table,
7 total set-creation steps (entries + hits + plan length - 1)
16 actions tried
```

This solution can be checked to ensure its accuracy. For example, the first action is to move the horizontally-placed $vehicle_6$, currently occupying squares (4,4) and (4,5) left one square to (4,3). According to

the action schema, square (4,5) will be deleted from the at() predicate associated with v6, and will be replaced by square (4,3).

4. The website in section (1.) classified three of the below problems as easy, medium, and hard. We found the insane problem on a website claiming to have the hardest problems possible in this domain, http://cs.ulb.ac.be/ fservais/rushhour/index.php?. The site claims that it requires at least 93 moves to solve.

Easy: 0.08 seconds, 16 actions
Medium: 0.26 seconds, 23 actions
Hard: Failure
Insane: Failure

After running the planner on four sample problems of varying difficulties (from easy to the most challenging), it was clear that Blackbox performs well only in relatively shallow, narrow search spaces. Since Blackbox is essentially an iterative deepening descent search, it must iterate the entire search tree of depth n, where n is the number of steps of the optimal solution. Deep, wide trees, that grow exponentially per iteration, cause Blackbox to fail for problems with significant large search spaces.

The solutions that Blackbox provides are quite useful. As expected, every action in a layer must be executed before actions in the next layer, but are order-independent of actions in the same layer. To solve a puzzle, the actions in BlackBox's solution can be executed top-to-bottom to reach the goal state. All of the solutions have the added benefit of being optimal in the number of actions given.