CS 4649/7649: Robot Intelligence Planning (Classical Planning)

Project 1: Classical Planning Assignment

Bharadwaj Tanikella, Claire Bergman, Jose Hernandez Anton, Kenneth Marino, Pujun Bhatnagar

Group Assignment 1: Classical Planning

Bharadwaj Tanikella

- Wrote PDDL files for Part 4, 10 and 12 discs
- Changed the FF planner to allow 12 disc plans.
- Write-up and Analysis for Part 1 and Part 4
- Compiled the document from all write-ups.

Claire Bergman

- Wrote A* planner for part 3
- Write-up for part 3.

Jose Hernandez Anton

- Wrote Domain file for sokoban problems
- Wrote PDDLs for all the sokoban problems, including the challenge
- Write-up for Part 2 of the assignment

Kenneth Marino

- Set up GIT repo and found 32-bit Linux FF and Blackbox planners
- Wrote HTN domain file
- Got HTN planner to work on Linux
- Write-up for part 5

Pujun Bhatnagar

- Wrote the backend for part 3 so that planner could work.
- Translated the 4 problems into a custom format.
- Helped Jose on part 2 and write-up.

Note

This report comprises of all the solutions and explanations to the problem set provided to us. The supporting files are provided in a zip file format with a collection of README files to navigate through the directories. The plans/results, which are generated through the planners, are provided in the respective directory. For instance the results for part 4 is provided in the /part4 directory of the zip.

Planners used: Blackbox, FF
Machine: 32-bit Linux
Github: https://github.com/KMarino/RIPFall2014P1

1. Towers of Hanoi

Explain the method by which each of the two planners find a solution:

The two classical planners, which are chosen, are the Blackbox planner and FF planner. Both the planners utilize a GRAPHPLAN system to iterate and traverse through the problem producing a planning graph. This graph will be further utilized to find action/operators across different stages of the problem in the propositional logic format with Pre-Condition ➔ Action.

BlackBox: The Blackbox planning algorithm essentially represents as a PDDL representation of a problem and a set of Boolean satisfiability problems (SAT), which are then solved using a variety of SAT solvers to produce a plan solving the problem. PDDL representation is structured in the STRIPS notation with an initial state, specification of the goal states and a set of pre and post conditions for the actions to be performed. Blackbox also contains state of the art satisfiability engines, which solves the converted PDDL representation problems, hence producing a set of plans. The front end of the Blackbox planner employs a GRAPHPLAN system to break the problem from the PDDL representation to a satisfiability format, further allowing SAT Engines to produce an optimal solution. The planning graph produced by GRAPHPLAN planner would allow Blackbox the capability of functioning fairly efficiently over a broad range of problems. The efficiency of the Blackbox planning algorithm is $O(n^3)$, which can be translated to O(GRAPHPLAN) + O(SAT Solver). Where n is the number of discs(problems).

FF Planner: Fast-Forward planning algorithm is a variation of a Heuristic as a special case of the graph-building phase in GRAPHPLAN. The difference being FF extends the heuristic to employ GRAPHPLANs plan extraction phase. FF planner uses the enforced hill-climbing algorithm to search for valid solutions, once a planning graph is generated through a GRAPHPLAN. The process of solving a problem (P) used by FF is as follows, when the problem is entered, the FF planner creates a collection of Sub Problems (P') to support the running iterations through the problem. Then the updated heuristic GRAPHPLAN is applied on the relaxed sub-problem (P'). Using the length of the relaxed GRAPHPLAN solution as a heuristic, the enforced hill-climbing algorithm to search for the valid solutions, which can relate to the problems. The FF planner keeps on executing the hill-climbing algorithm to find an optimal heuristic evaluation and finds a solution related to it.

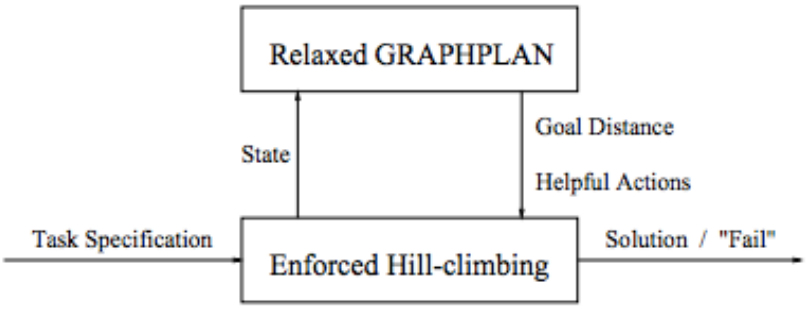
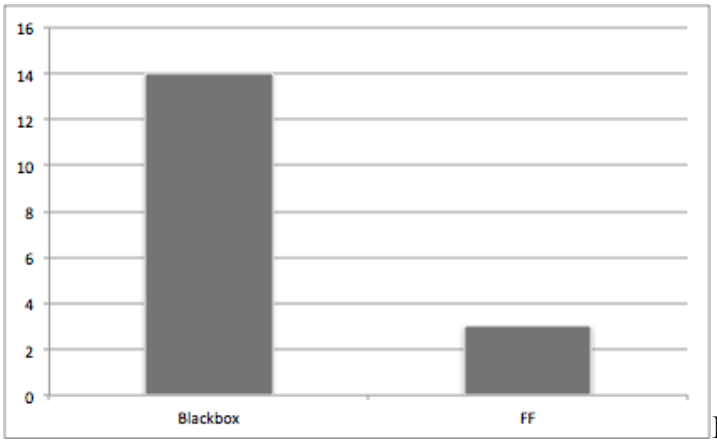


Figure: System Architecture of FF Planner (Hoffman. J & Nebel. B, 2001)

Which planner was fastest?

Both the planners were run in the same environment (Linux 32-bit) to produce a solution to the PDDL file of three discs. The graph represents the amount of time it took to solve for three disks by each planner in milliseconds. Clearly FF is the fastest compared to Blackbox.



Blackbox: 0.014 s
FF: 0.003 s

The reason FF is better than Blackbox is because of the fact that FF uses an Enforced Hill Climbing method which minimizes the use of SAT Solver. The Big O of SAT Solver is 2^n, n being the number of discs. Even though Enforced Hill Climbing algorithm is not optimal due to the usage of Best First Search and risk of finding a local maxima, it definitely is better than an SAT Solver the Blackbox planner utilizes. Also the use of a heuristic approach to update the GRAPHPLAN algorithm helps minimize the number of states, which are to be evaluated.

Explain why the winning planner might be more effective on this problem:

The number of states in the case of Towers of Hanoi can range from the following, 2^n-1 to 3^n, n being the number of discs in Towers of Hanoi. FF planner only uses the GRAPHPLAN as a heuristic for the relaxed subset of problems. FF planner however heavily depend on the enforced hill climbing algorithm, which utilizes best first search ideology and also contain the fact that not all the states are needed to be traversed if the optimal solution is found. However the other planner Blackbox requires to construct a planning graph for every level, which needs to traverse every state possible. Further usage of GRAPHPLAN algorithm over the multiple states can create very similar SAT problems and further making it very inefficient and unsatisifiable. Therefore making Blackbox very inefficient in respect of time and space for larger set of problems and states.

---

2 Sokoban PDDL

Show successful plans from at least one planner on the three Sokoban problems in Figure 2(1-3). The challenge problem is optional.

Below are the successful results from the **Black Box planner**:

```
pujuncool@pujuncool-VirtualBox:~/Desktop/project 1/Part2$ ../planners/blackbox -o sokoban-domain.pddl -f sokobanP2-1.pddl
blackbox version 43
command line:  ../planners/blackbox -o sokoban-domain.pddl -f sokobanP2-1.pddl

Begin solver specification
    -maxint        0   -maxsec 10.000000  graphplan
    -maxint        0   -maxsec 0.000000  chaff
End solver specification
Loading domain file: sokoban-domain.pddl
Loading fact file: sokobanP2-1.pddl
Problem name: sokobanp2-1
Facts loaded.
time: 1, 47 facts and 4 exclusive pairs.
time: 2, 51 facts and 21 exclusive pairs.
time: 3, 57 facts and 81 exclusive pairs.
time: 4, 59 facts and 104 exclusive pairs.
time: 5, 60 facts and 109 exclusive pairs.
time: 6, 61 facts and 116 exclusive pairs.
time: 7, 61 facts and 109 exclusive pairs.
time: 8, 62 facts and 117 exclusive pairs.
time: 9, 64 facts and 147 exclusive pairs.
time: 10, 64 facts and 135 exclusive pairs.
time: 11, 64 facts and 126 exclusive pairs.
time: 12, 65 facts and 139 exclusive pairs.
time: 13, 66 facts and 153 exclusive pairs.
time: 14, 67 facts and 166 exclusive pairs.
Goals first reachable in 14 steps.
2175 nodes created.

##################################################
goals at time 15:
  at_thebox_c2


--------------------------------------------------
Invoking solver graphplan
Result is Sat
Iteration was 356
Performing plan justification:
   0 actions were pruned in 0.00 seconds

--------------------------------------------------
Begin plan
1 (move-bot-down thebot c3 c5)
2 (move-bot-right thebot c5 c6)
3 (move-bot-right thebot c6 c7)
4 (move-bot-down thebot c7 c11)
5 (move-box-left thebot thebox c11 c10 c9)
6 (move-bot-up thebot c10 c6)
7 (move-bot-left thebot c6 c5)
8 (move-bot-left thebot c5 c4)
9 (move-bot-down thebot c4 c8)
10 (move-bot-down thebot c8 c12)
11 (move-bot-right thebot c12 c13)
12 (move-box-up thebot thebox c13 c9 c5)
13 (move-box-up thebot thebox c9 c5 c3)
14 (move-box-up thebot thebox c5 c3 c2)
End plan
--------------------------------------------------

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

##################################################
Total elapsed time:   0.13 seconds
Time in milliseconds: 132

##################################################
pujuncool@pujuncool-VirtualBox:~/Desktop/project 1/Part2$ $
```

```
pujuncool@pujuncool-VirtualBox:~/Desktop/project 1/Part2$ ../planners/blackbox -o sokoban-domain.pddl -f sokobanP2-2.pddl
blackbox version 43
command line:  ../planners/blackbox -o sokoban-domain.pddl -f sokobanP2-2.pddl

Begin solver specification
    -maxint        0   -maxsec 10.000000  graphplan
    -maxint        0   -maxsec 0.000000  chaff
End solver specification
Loading domain file: sokoban-domain.pddl
Loading fact file: sokobanP2-2.pddl
Problem name: sokobanp2-2
Facts loaded.
time: 1, 48 facts and 0 exclusive pairs.
time: 2, 48 facts and 0 exclusive pairs.
240 nodes created.

------------------------------------------------
NO SOLUTION
Problem not solvable: can't even reach non-mutex goals
    Search halted after 0 steps
    0 entries in hash table and 0 hits
    avg set size 0
------------------------------------------------

################################################
Total elapsed time:   0.02 seconds
Time in milliseconds: 21

################################################
pujuncool@pujuncool-VirtualBox:~/Desktop/project 1/Part2$
```

Successful Results from **ff planner**:

```
pujuncool@pujuncool-VirtualBox:~/Desktop/project 1/Part2$ ../planners/ff -o sokoban-domain.pddl -f sokobanP2-3.pddl

ff: parsing domain file
domain 'SOKOBAN-DOMAIN' defined
 ... done.
ff: parsing problem file
problem 'SOKOBANP2-3' defined
 ... done.



Cueing down from goal distance:   10 into depth [1]
                                   9           [1][2][3][4][5][6][7][8][9]
                                   8           [1][2]

Enforced Hill-climbing failed !
switching to Best-first Search now.

advancing to distance :   10
                           9
                           7
                           6
                           5
                           4
                           3
                           2
                           1
                           0

ff: found legal plan as follows

step    0: MOVE-BOT-RIGHT THEBOT C9 C10
        1: MOVE-BOT-DOWN THEBOT C10 C18
        2: MOVE-BOT-DOWN THEBOT C18 C23
        3: MOVE-BOT-RIGHT THEBOT C23 C24
        4: MOVE-BOT-RIGHT THEBOT C24 C25
        5: MOVE-BOT-UP THEBOT C25 C19
        6: MOVE-BOT-UP THEBOT C19 C12
        7: MOVE-BOX-LEFT THEBOT BOXC C12 C11 C10
        8: MOVE-BOX-LEFT THEBOT BOXC C11 C10 C9
        9: MOVE-BOT-RIGHT THEBOT C10 C11
       10: MOVE-BOT-RIGHT THEBOT C11 C12
       11: MOVE-BOT-DOWN THEBOT C12 C19
       12: MOVE-BOT-DOWN THEBOT C19 C25
       13: MOVE-BOT-RIGHT THEBOT C25 C26
       14: MOVE-BOT-RIGHT THEBOT C26 C27
       15: MOVE-BOT-UP THEBOT C27 C20
       16: MOVE-BOT-UP THEBOT C20 C14
       17: MOVE-BOX-LEFT THEBOT BOXB C14 C13 C12
       18: MOVE-BOX-LEFT THEBOT BOXB C13 C12 C11
       19: MOVE-BOT-RIGHT THEBOT C12 C13
       20: MOVE-BOT-RIGHT THEBOT C13 C14
       21: MOVE-BOT-UP THEBOT C14 C6
       22: MOVE-BOT-UP THEBOT C6 C4
       23: MOVE-BOT-UP THEBOT C4 C1
       24: MOVE-BOT-RIGHT THEBOT C1 C2
       25: MOVE-BOT-RIGHT THEBOT C2 C3
       26: MOVE-BOT-DOWN THEBOT C3 C5
       27: MOVE-BOT-DOWN THEBOT C5 C7
       28: MOVE-BOT-DOWN THEBOT C7 C16
       29: MOVE-BOX-LEFT THEBOT BOXA C16 C15 C14
       30: MOVE-BOX-LEFT THEBOT BOXA C15 C14 C13
       31: MOVE-BOT-DOWN THEBOT C14 C20
       32: MOVE-BOT-DOWN THEBOT C20 C27
       33: MOVE-BOT-LEFT THEBOT C27 C26
       34: MOVE-BOT-LEFT THEBOT C26 C25
       35: MOVE-BOT-UP THEBOT C25 C19
       36: MOVE-BOT-UP THEBOT C19 C12
       37: MOVE-BOX-RIGHT THEBOT BOXA C12 C13 C14
       38: MOVE-BOT-LEFT THEBOT C13 C12
       39: MOVE-BOT-DOWN THEBOT C12 C19
```

```
 40: MOVE-BOT-DOWN THEBOT C19 C25
 41: MOVE-BOT-RIGHT THEBOT C25 C26
 42: MOVE-BOT-RIGHT THEBOT C26 C27
 43: MOVE-BOT-UP THEBOT C27 C20
 44: MOVE-BOX-UP THEBOT BOXA C20 C14 C6
 45: MOVE-BOT-LEFT THEBOT C14 C13
 46: MOVE-BOT-LEFT THEBOT C13 C12
 47: MOVE-BOT-DOWN THEBOT C12 C19
 48: MOVE-BOT-DOWN THEBOT C19 C25
 49: MOVE-BOT-LEFT THEBOT C25 C24
 50: MOVE-BOT-LEFT THEBOT C24 C23
 51: MOVE-BOT-UP THEBOT C23 C18
 52: MOVE-BOT-UP THEBOT C18 C10
 53: MOVE-BOX-RIGHT THEBOT BOXB C10 C11 C12
 54: MOVE-BOX-RIGHT THEBOT BOXB C11 C12 C13
 55: MOVE-BOT-DOWN THEBOT C12 C19
 56: MOVE-BOT-DOWN THEBOT C19 C25
 57: MOVE-BOT-RIGHT THEBOT C25 C26
 58: MOVE-BOT-RIGHT THEBOT C26 C27
 59: MOVE-BOT-UP THEBOT C27 C20
 60: MOVE-BOT-UP THEBOT C20 C14
 61: MOVE-BOX-UP THEBOT BOXA C14 C6 C4
 62: MOVE-BOX-UP THEBOT BOXA C6 C4 C1
 63: MOVE-BOT-DOWN THEBOT C4 C6
 64: MOVE-BOT-DOWN THEBOT C6 C14
 65: MOVE-BOT-DOWN THEBOT C14 C20
 66: MOVE-BOT-DOWN THEBOT C20 C27
 67: MOVE-BOT-LEFT THEBOT C27 C26
 68: MOVE-BOT-LEFT THEBOT C26 C25
 69: MOVE-BOT-UP THEBOT C25 C19
 70: MOVE-BOT-UP THEBOT C19 C12
 71: MOVE-BOX-RIGHT THEBOT BOXB C12 C13 C14
 72: MOVE-BOT-LEFT THEBOT C13 C12
 73: MOVE-BOT-DOWN THEBOT C12 C19
 74: MOVE-BOT-DOWN THEBOT C19 C25
 75: MOVE-BOT-RIGHT THEBOT C25 C26
 76: MOVE-BOT-RIGHT THEBOT C26 C27
 77: MOVE-BOT-UP THEBOT C27 C20
 78: MOVE-BOX-UP THEBOT BOXB C20 C14 C6
 79: MOVE-BOT-LEFT THEBOT C14 C13
 80: MOVE-BOT-LEFT THEBOT C13 C12
 81: MOVE-BOT-LEFT THEBOT C12 C11
 82: MOVE-BOT-LEFT THEBOT C11 C10
 83: MOVE-BOT-DOWN THEBOT C10 C18
 84: MOVE-BOT-DOWN THEBOT C18 C23
 85: MOVE-BOT-LEFT THEBOT C23 C22
 86: MOVE-BOT-LEFT THEBOT C22 C21
 87: MOVE-BOT-UP THEBOT C21 C17
 88: MOVE-BOT-UP THEBOT C17 C8
 89: MOVE-BOX-RIGHT THEBOT BOXC C8 C9 C10
 90: MOVE-BOX-RIGHT THEBOT BOXC C9 C10 C11
 91: MOVE-BOX-RIGHT THEBOT BOXC C10 C11 C12
 92: MOVE-BOX-RIGHT THEBOT BOXC C11 C12 C13
 93: MOVE-BOT-DOWN THEBOT C12 C19
 94: MOVE-BOT-DOWN THEBOT C19 C25
 95: MOVE-BOT-RIGHT THEBOT C25 C26
 96: MOVE-BOT-RIGHT THEBOT C26 C27
 97: MOVE-BOT-UP THEBOT C27 C20
 98: MOVE-BOT-UP THEBOT C20 C14
 99: MOVE-BOX-UP THEBOT BOXB C14 C6 C4
100: MOVE-BOT-DOWN THEBOT C6 C14
101: MOVE-BOT-DOWN THEBOT C14 C20
102: MOVE-BOT-DOWN THEBOT C20 C27
103: MOVE-BOT-LEFT THEBOT C27 C26
104: MOVE-BOT-LEFT THEBOT C26 C25
105: MOVE-BOT-UP THEBOT C25 C19
106: MOVE-BOT-UP THEBOT C19 C12
107: MOVE-BOX-RIGHT THEBOT BOXC C12 C13 C14
108: MOVE-BOT-LEFT THEBOT C13 C12
```

```
      54: MOVE-BOX-RIGHT THEBOT BOXB C11 C12 C13
      55: MOVE-BOT-DOWN THEBOT C12 C19
      56: MOVE-BOT-DOWN THEBOT C19 C25
      57: MOVE-BOT-RIGHT THEBOT C25 C26
      58: MOVE-BOT-RIGHT THEBOT C26 C27
      59: MOVE-BOT-UP THEBOT C27 C20
      60: MOVE-BOT-UP THEBOT C20 C14
      61: MOVE-BOX-UP THEBOT BOXA C14 C6 C4
      62: MOVE-BOX-UP THEBOT BOXA C6 C4 C1
      63: MOVE-BOT-DOWN THEBOT C4 C6
      64: MOVE-BOT-DOWN THEBOT C6 C14
      65: MOVE-BOT-DOWN THEBOT C14 C20
      66: MOVE-BOT-DOWN THEBOT C20 C27
      67: MOVE-BOT-LEFT THEBOT C27 C26
      68: MOVE-BOT-LEFT THEBOT C26 C25
      69: MOVE-BOT-UP THEBOT C25 C19
      70: MOVE-BOT-UP THEBOT C19 C12
      71: MOVE-BOX-RIGHT THEBOT BOXB C12 C13 C14
      72: MOVE-BOT-LEFT THEBOT C13 C12
      73: MOVE-BOT-DOWN THEBOT C12 C19
      74: MOVE-BOT-DOWN THEBOT C19 C25
      75: MOVE-BOT-RIGHT THEBOT C25 C26
      76: MOVE-BOT-RIGHT THEBOT C26 C27
      77: MOVE-BOT-UP THEBOT C27 C20
      78: MOVE-BOX-UP THEBOT BOXB C20 C14 C6
      79: MOVE-BOT-LEFT THEBOT C14 C13
      80: MOVE-BOT-LEFT THEBOT C13 C12
      81: MOVE-BOT-LEFT THEBOT C12 C11
      82: MOVE-BOT-LEFT THEBOT C11 C10
      83: MOVE-BOT-DOWN THEBOT C10 C18
      84: MOVE-BOT-DOWN THEBOT C18 C23
      85: MOVE-BOT-LEFT THEBOT C23 C22
      86: MOVE-BOT-LEFT THEBOT C22 C21
      87: MOVE-BOT-UP THEBOT C21 C17
      88: MOVE-BOT-UP THEBOT C17 C8
      89: MOVE-BOX-RIGHT THEBOT BOXC C8 C9 C10
      90: MOVE-BOX-RIGHT THEBOT BOXC C9 C10 C11
      91: MOVE-BOX-RIGHT THEBOT BOXC C10 C11 C12
      92: MOVE-BOX-RIGHT THEBOT BOXC C11 C12 C13
      93: MOVE-BOT-DOWN THEBOT C12 C19
      94: MOVE-BOT-DOWN THEBOT C19 C25
      95: MOVE-BOT-RIGHT THEBOT C25 C26
      96: MOVE-BOT-RIGHT THEBOT C26 C27
      97: MOVE-BOT-UP THEBOT C27 C20
      98: MOVE-BOT-UP THEBOT C20 C14
      99: MOVE-BOX-UP THEBOT BOXB C14 C6 C4
     100: MOVE-BOT-DOWN THEBOT C6 C14
     101: MOVE-BOT-DOWN THEBOT C14 C20
     102: MOVE-BOT-DOWN THEBOT C20 C27
     103: MOVE-BOT-LEFT THEBOT C27 C26
     104: MOVE-BOT-LEFT THEBOT C26 C25
     105: MOVE-BOT-UP THEBOT C25 C19
     106: MOVE-BOT-UP THEBOT C19 C12
     107: MOVE-BOX-RIGHT THEBOT BOXC C12 C13 C14
     108: MOVE-BOT-LEFT THEBOT C13 C12
     109: MOVE-BOT-DOWN THEBOT C12 C19
     110: MOVE-BOT-DOWN THEBOT C19 C25
     111: MOVE-BOT-RIGHT THEBOT C25 C26
     112: MOVE-BOT-RIGHT THEBOT C26 C27
     113: MOVE-BOT-UP THEBOT C27 C20
     114: MOVE-BOX-UP THEBOT BOXC C20 C14 C6


time spent:    0.00 seconds instantiating 192 easy, 0 hard action templates
               0.00 seconds reachability analysis, yielding 114 facts and 126 actions
               0.00 seconds creating final representation with 97 relevant facts
               0.00 seconds building connectivity graph
               0.01 seconds searching, evaluating 2306 states, to a max depth of 9
               0.01 seconds total time

pujuncool@pujuncool-VirtualBox:~/Desktop/project 1/Part2$
```

Results for the *challenge problem*:

```
pujuncool@pujuncool-VirtualBox:~/Desktop/project 1/Part2$ ../planners/ff -o sokoban-domain.pddl -f sokobanChallenge.pddl

ff: parsing domain file
domain 'SOKOBAN-DOMAIN' defined
 ... done.
ff: parsing problem file
problem 'SOKOBANP2-3' defined
 ... done.



Cueing down from goal distance:   19 into depth [1]
                                  18            [1]
                                  15            [1]
                                  11            [1]
                                   8            [1]
                                   6            [1]
                                   5            [1]

Enforced Hill-climbing failed !
switching to Best-first Search now.

advancing to distance :   19
                          18
                          11
                          10
                           9
                           8
                           6
                           3
                           2
                           1
                           0

ff: found legal plan as follows

step    0: MOVE-BOT-LEFT THEBOT C9 C8
        1: MOVE-BOT-LEFT THEBOT C8 C7
        2: MOVE-BOT-DOWN THEBOT C7 C12
        3: MOVE-BOT-LEFT THEBOT C12 C11
        4: MOVE-BOT-LEFT THEBOT C11 C10
        5: MOVE-BOT-UP THEBOT C10 C5
        6: MOVE-BOX-RIGHT THEBOT BOXB C5 C6 C7
        7: MOVE-BOX-RIGHT THEBOT BOXB C6 C7 C8
        8: MOVE-BOT-DOWN THEBOT C7 C12
        9: MOVE-BOT-DOWN THEBOT C12 C16
       10: MOVE-BOT-DOWN THEBOT C16 C19
       11: MOVE-BOT-LEFT THEBOT C19 C18
       12: MOVE-BOX-UP THEBOT BOXC C18 C15 C11
       13: MOVE-BOT-RIGHT THEBOT C15 C16
       14: MOVE-BOT-UP THEBOT C16 C12
       15: MOVE-BOT-UP THEBOT C12 C7
       16: MOVE-BOT-LEFT THEBOT C7 C6
       17: MOVE-BOT-LEFT THEBOT C6 C5
       18: MOVE-BOT-DOWN THEBOT C5 C10
       19: MOVE-BOX-RIGHT THEBOT BOXC C10 C11 C12
       20: MOVE-BOT-DOWN THEBOT C11 C15
       21: MOVE-BOT-RIGHT THEBOT C15 C16
       22: MOVE-BOT-RIGHT THEBOT C16 C17
       23: MOVE-BOT-UP THEBOT C17 C13
       24: MOVE-BOT-RIGHT THEBOT C13 C14
       25: MOVE-BOT-UP THEBOT C14 C9
       26: MOVE-BOX-LEFT THEBOT BOXB C9 C8 C7
       27: MOVE-BOX-LEFT THEBOT BOXB C8 C7 C6
       28: MOVE-BOT-RIGHT THEBOT C7 C8
       29: MOVE-BOT-DOWN THEBOT C8 C13
       30: MOVE-BOT-DOWN THEBOT C13 C17
       31: MOVE-BOT-LEFT THEBOT C17 C16
       32: MOVE-BOT-LEFT THEBOT C16 C15
```

```
33: MOVE-BOT-UP THEBOT C15 C11
34: MOVE-BOX-RIGHT THEBOT BOXC C11 C12 C13
35: MOVE-BOT-UP THEBOT C12 C7
36: MOVE-BOT-RIGHT THEBOT C7 C8
37: MOVE-BOT-UP THEBOT C8 C4
38: MOVE-BOT-UP THEBOT C4 C2
39: MOVE-BOT-LEFT THEBOT C2 C1
40: MOVE-BOX-DOWN THEBOT BOXA C1 C3 C7
41: MOVE-BOX-DOWN THEBOT BOXA C3 C7 C12
42: MOVE-BOX-DOWN THEBOT BOXA C7 C12 C16
```

```
      43: MOVE-BOT-LEFT THEBOT C12 C11
      44: MOVE-BOT-DOWN THEBOT C11 C15
      45: MOVE-BOT-DOWN THEBOT C15 C18
      46: MOVE-BOT-RIGHT THEBOT C18 C19
      47: MOVE-BOX-UP THEBOT BOXA C19 C16 C12
      48: MOVE-BOT-RIGHT THEBOT C16 C17
      49: MOVE-BOX-UP THEBOT BOXC C17 C13 C8
      50: MOVE-BOT-RIGHT THEBOT C13 C14
      51: MOVE-BOT-UP THEBOT C14 C9
      52: MOVE-BOX-LEFT THEBOT BOXC C9 C8 C7
      53: MOVE-BOT-DOWN THEBOT C8 C13
      54: MOVE-BOT-DOWN THEBOT C13 C17
      55: MOVE-BOT-LEFT THEBOT C17 C16
      56: MOVE-BOT-LEFT THEBOT C16 C15
      57: MOVE-BOT-UP THEBOT C15 C11
      58: MOVE-BOX-RIGHT THEBOT BOXA C11 C12 C13
      59: MOVE-BOX-UP THEBOT BOXC C12 C7 C3
      60: MOVE-BOT-RIGHT THEBOT C7 C8
      61: MOVE-BOT-UP THEBOT C8 C4
      62: MOVE-BOT-UP THEBOT C4 C2
      63: MOVE-BOT-LEFT THEBOT C2 C1
      64: MOVE-BOX-DOWN THEBOT BOXC C1 C3 C7
      65: MOVE-BOX-DOWN THEBOT BOXC C3 C7 C12
      66: MOVE-BOX-DOWN THEBOT BOXC C7 C12 C16
      67: MOVE-BOT-LEFT THEBOT C12 C11
      68: MOVE-BOT-LEFT THEBOT C11 C10
      69: MOVE-BOT-UP THEBOT C10 C5
      70: MOVE-BOX-RIGHT THEBOT BOXB C5 C6 C7
      71: MOVE-BOT-DOWN THEBOT C6 C11
      72: MOVE-BOT-RIGHT THEBOT C11 C12
      73: MOVE-BOX-UP THEBOT BOXB C12 C7 C3
      74: MOVE-BOT-RIGHT THEBOT C7 C8
      75: MOVE-BOT-RIGHT THEBOT C8 C9
      76: MOVE-BOT-DOWN THEBOT C9 C14
      77: MOVE-BOX-LEFT THEBOT BOXA C14 C13 C12
      78: MOVE-BOT-DOWN THEBOT C13 C17
      79: MOVE-BOX-LEFT THEBOT BOXC C17 C16 C15
      80: MOVE-BOT-DOWN THEBOT C16 C19
      81: MOVE-BOT-LEFT THEBOT C19 C18
      82: MOVE-BOX-UP THEBOT BOXC C18 C15 C11
      83: MOVE-BOT-RIGHT THEBOT C15 C16
      84: MOVE-BOT-RIGHT THEBOT C16 C17
      85: MOVE-BOT-UP THEBOT C17 C13
      86: MOVE-BOT-UP THEBOT C13 C8
      87: MOVE-BOT-LEFT THEBOT C8 C7
      88: MOVE-BOT-LEFT THEBOT C7 C6
      89: MOVE-BOX-DOWN THEBOT BOXC C6 C11 C15
      90: MOVE-BOX-RIGHT THEBOT BOXA C11 C12 C13
      91: MOVE-BOT-DOWN THEBOT C12 C16
      92: MOVE-BOT-DOWN THEBOT C16 C19
      93: MOVE-BOT-LEFT THEBOT C19 C18
      94: MOVE-BOX-UP THEBOT BOXC C18 C15 C11
      95: MOVE-BOT-RIGHT THEBOT C15 C16
      96: MOVE-BOT-UP THEBOT C16 C12
      97: MOVE-BOT-UP THEBOT C12 C7
      98: MOVE-BOT-RIGHT THEBOT C7 C8
      99: MOVE-BOT-UP THEBOT C8 C4
     100: MOVE-BOT-UP THEBOT C4 C2
     101: MOVE-BOT-LEFT THEBOT C2 C1
     102: MOVE-BOX-DOWN THEBOT BOXB C1 C3 C7
     103: MOVE-BOX-DOWN THEBOT BOXB C3 C7 C12


time spent:    0.00 seconds instantiating 150 easy, 0 hard action templates
               0.00 seconds reachability analysis, yielding 118 facts and 150 actions
               0.00 seconds creating final representation with 114 relevant facts
               0.00 seconds building connectivity graph
               0.72 seconds searching, evaluating 42468 states, to a max depth of 1
               0.72 seconds total time
```

Compare the performance of two planners on this domain. Which one works better? Does this make sense, why?

FF planner performs considerably better than the blackbox planner. FF planner was able to find a solution for all the problems taking under 0.01 seconds for the first three problems and 0.68 seconds for the Sokoban Challenge. On the other hand, the blackbox planner takes 0.11 seconds to solve the first problem, 0.02 seconds for the second one, and is unable to solve the third problem and the challenge because the running time exceeds the allotted time. As discussed in part 1, this result is expected because FF has uses a heuristic function to approach the solution. FF first finds a solution to a simpler task using a graph-style algorithm, and uses that to approximate the number of actions needed to achieve the goal of the main task. In the other hand, Blackbox does not use a heuristic function, but first converts the problem into a boolean problem, and tries to reach a solution implementing a graphplan system, which results in a longer time of execution that the FF planner.

Results:

- **FF Planner**
  - Problem 1: Took 0.01 seconds to solve the problem with 13 steps.
  - Problem 2: Took 0.01 seconds to determine it could not be solved.
  - Problem 3: Took 0.01 seconds to solve the problem with 114 steps
  - Challenge: Took 0.71 seconds to solve the problem in 103 steps

- **BlackBox Planner**
  - Problem 1: Took 0.113 seconds to solve problem performing 14 actions.
  - Problem 2: Took 0.020 seconds to find there was no solution to the problem.
  - Problem 3: Times out. Runs out of sources.
  - Challenge: Times out. Runs out of sources.

Clearly PDDL was not intended for this sort of application. Discuss the challenges in expressing geometric constraints in semantic planning.

Several problems arise when trying to represent geometric constraints in classical/semantic planning. First, it is required that all geometric constraints be specified in the domain. In this case, a robot is to move in a 2-D grid. To do this in semantic planning, it is required to specify the location of each grid with respect to the others, i.e. onTop(x,y), isLeft(x,y), and which grids are occupied by another object. This will result in a large domain file. Not only that, but also there will be a huge amount of preconditions to be checked before any movement happens, which will result in a very slow running time.

The group experienced one such problem when creating PPDL files to solve the Sokoban challenges. At first, the domain file accounted for the walls in the puzzles, and one of the preconditions for robot and box movement was that the destination could not be a wall. When trying to execute this, the running time was so big that the blackbox planner was not able to solve any of the problems in the allotted time. The domain definition was changed to not include the walls, and just define spots where the robot could freely move. This is an example of how geometric constraints are not handled well in semantic planning. For these reasons, classical planning is not really suitable for problems with geometric constraints, because

creating a geometric constraint in classical planning usually leads to clutter in the domain file and over complexity of the problem, which consequently leads to slow execution.

In many cases, geometric and dynamic planning are insufficient to describe a domain. Give an example of a problem that is best suited for sematic (classical) planning. Explain why a semantic representation would be desirable.

Classical planning is better suited for problems that do not have geometric or motion constraints, but rather that deal with finding a solution to logical/logistics problems. For example, classical planning is really useful for coordination problems, such as scheduling: project scheduling, plane departure scheduling, and the so. These problems do not require over specification in the domain file, i.e. for these problems it is not necessary to specify a space because it really does not matter how the problems are solved in a kinetic space. In these kind of problems what really matters is event/action dependency, which can be specified in the domain without much clutter. A logistics solution would be easily found without the need of specifying, or even worrying about geometric constraints.

3 Sokoban Challenge

Give successful plans from your planner on the Sokoban problems in Figure 2 and any others.

Language used: Python
Machine vitals:

Processor:	Intel(R) Core(TM) i5 CPU	M 540 @ 2.53GHz 2.53 GHz
Installed memory (RAM):	8.00 GB	
System type:	64-bit Operating System	

Plans in form of player's location: (x,y):
Problem 2.1:

```
Found Goal State in Time:  0.00300002098083  Seconds
Map:
[2, 2, 2, 2, 2, 2]
[2, 1, 0, 2, 2, 2]
[2, 2, 0, 2, 2, 2]
[2, 1, 0, 0, 1, 2]
[2, 0, 0, 0, 1, 2]
[2, 1, 1, 2, 2, 2]
[2, 2, 2, 2, 2, 2]
PlayerLocation:  (2, 2)
Box Locations:  [(2, 1)]
Goal Location:  [(2, 1)]
Num States Visited:  28
Num Moves:  14
Player Moves:  [(2, 2), (2, 3), (3, 3), (4, 3), (4, 4), (3, 4), (3, 3), (2,
(2, 3)]
```

Problem 2.2:

```
Could not find Goal State in Time:  0.0160000324249  Seconds
Map:
[2, 2, 2, 2, 2, 2]
[2, 0, 0, 1, 2, 2]
[2, 1, 0, 0, 1, 2]
[2, 2, 1, 0, 0, 2]
[2, 2, 2, 2, 0, 2]
[2, 2, 2, 2, 0, 2]
[2, 2, 2, 2, 2, 2]
Num States Visited:  121
```

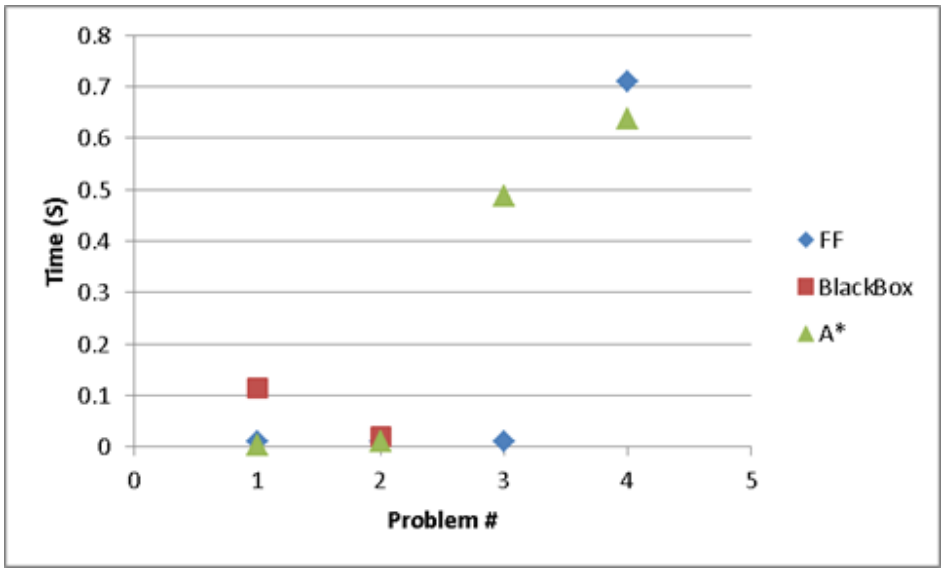
Problem 2.3:

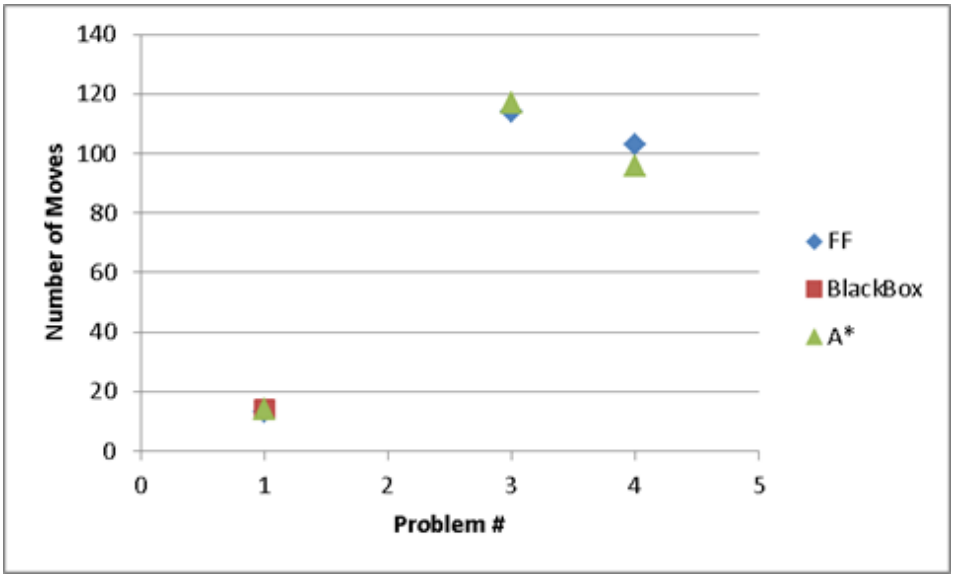
```
Found Goal State in Time:  0.493000030518  Seconds
Map:
[2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
[2, 2, 2, 2, 2, 2, 2, 0, 0, 1, 2]
[2, 2, 2, 2, 2, 2, 2, 0, 2, 0, 2]
[2, 2, 2, 2, 2, 2, 2, 0, 2, 0, 2]
[2, 1, 0, 0, 0, 0, 0, 0, 0, 1, 2]
[2, 0, 2, 0, 2, 0, 2, 0, 2, 2, 2]
[2, 1, 0, 0, 0, 0, 0, 1, 2, 2, 2]
[2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
PlayerLocation:  (7, 4)
Box Locations:  [(7, 3), (7, 2), (7, 1)]
Goal Location:  [(7, 1), (7, 2), (7, 3)]
Num States Visited:  2166
Num Moves:  117
Player Moves:  [(2, 4), (3, 4), (3, 5), (3, 6), (4, 6), (5, 6), (5, 5), (5, 4),
(5, 6), (6, 6), (7, 6), (7, 5), (7, 4), (7, 3), (7, 2), (7, 1), (8, 1), (9, 1),
(9, 3), (9, 2), (9, 1), (8, 1), (7, 1), (7, 2), (7, 3), (7, 4), (6, 4), (5, 4),
(3, 4), (4, 4), (5, 4), (4, 4), (3, 4), (3, 5), (3, 6), (2, 6), (1, 6), (1, 5),
(4, 6), (5, 6), (6, 6), (7, 6), (7, 5), (7, 4), (7, 3), (7, 2), (7, 3), (7, 4),
(5, 4), (6, 4), (5, 4), (5, 5), (5, 6), (4, 6), (3, 6), (2, 6), (1, 6), (1, 5),
(5, 5), (5, 6), (6, 6), (7, 6), (7, 5), (7, 4), (7, 3), (7, 4), (7, 5), (7, 6),
(1, 6), (1, 5), (1, 4), (2, 4), (3, 4), (4, 4), (5, 4), (6, 4), (5, 4), (5, 5),
```

Problem 2.4:

```
Found Goal State in Time:  0.713000059128  Seconds
Map:
[2, 2, 2, 2, 2, 2, 2]
[2, 2, 2, 1, 1, 2, 2]
[2, 2, 2, 0, 0, 2, 2]
[2, 1, 0, 0, 0, 1, 2]
[2, 1, 0, 0, 0, 1, 2]
[2, 2, 0, 0, 1, 2, 2]
[2, 2, 1, 1, 2, 2, 2]
[2, 2, 2, 2, 2, 2, 2]
PlayerLocation:  (3, 3)
Box Locations:  [(3, 4), (2, 4), (4, 4)]
Goal Location:  [(2, 4), (3, 4), (4, 4)]
Num States Visited:  2249
Num Moves:  96
Player Moves:  [(5, 3), (4, 3), (3, 3), (3, 4), (2, 4), (1, 4), (1, 3), (2, 3),
(2, 5), (3, 5), (3, 4), (3, 3), (2, 3), (1, 3), (1, 4), (2, 4), (2, 5), (3, 5),
(3, 3), (4, 3), (4, 2), (4, 1), (3, 1), (3, 2), (4, 2), (4, 3), (4, 4), (4, 5),
(4, 3), (4, 2), (4, 1), (3, 1), (3, 2), (3, 3), (3, 4), (2, 4), (1, 4), (1, 3),
(4, 2), (4, 1), (3, 1), (3, 2), (4, 2), (4, 3), (5, 3), (5, 4), (4, 4), (3, 4),
(4, 3), (4, 4), (4, 5), (3, 5), (3, 4), (3, 3), (2, 3), (1, 3), (1, 4), (2, 4),
(3, 5), (3, 4), (3, 3), (4, 3), (4, 2), (4, 1), (3, 1), (3, 2)]
```

Compare the performance of your planner to the PDDL planners you used in the previous problem. Which was faster? Why?





On average, FF is that fastest algorithm, although A* does perform faster than FF on Problem 4, the most complicated problem in the set. Blackbox tends to take the most time and could not even solve problems 3 or 4. In terms of optimality, A* tends to outperform FF because on average it solves the problems in fewer numbers of moves. This is expected because FF sacrifices some optimality in favor of faster performance.

Prove that your planner was complete. Your instructor has a math background: a proof is a convincing argument. Make sure you address each aspect of completeness and why your planner satisfies it. Pictures are always welcome.

Completeness means that a planner will either give a solution or report that there is none in a finite amount of time. The planner used was an A*, which is by definition complete. The only change made to A* was to remove the states that resulted in a cube being stuck in a corner. This would not affect the completeness of A* as the states removed are not goal states and their child states cannot become goal states either.

The heuristic used was the sum of the Manhattan distances from each block to the nearest goal state. This consistent satisfies the requirements for consistency because the action of moving a block, the only action that can get us closer to the goal state, will decrease the number of moves to the goal state by one and decrease the heuristic by one. Additionally, at the goal state, the heuristic is always 0.

What methods did you use to speed up the planning? Give a short description of each method and explain why it did or didn't help on each relevant problem.

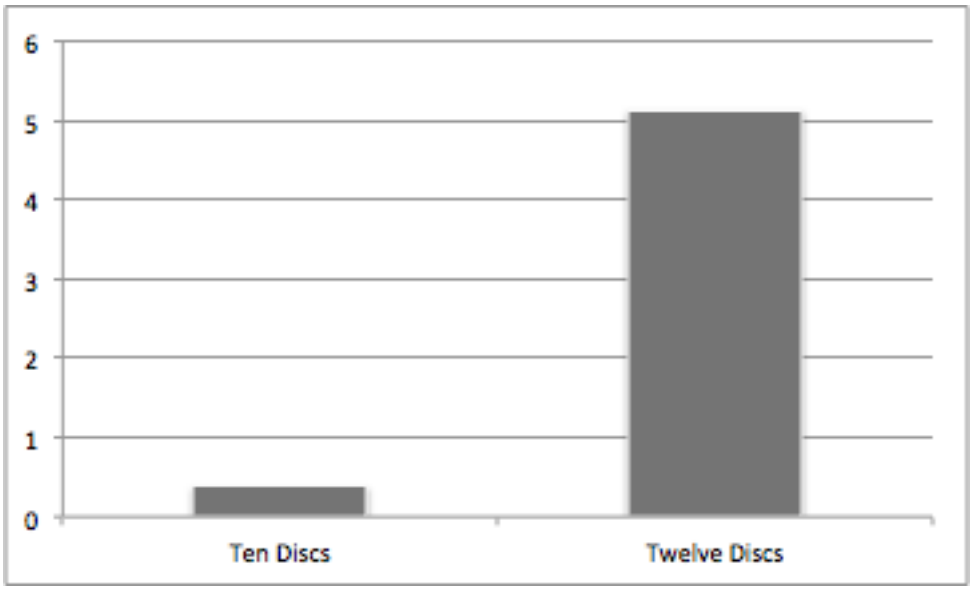
In order to speed up planning, A* was used with a heuristic that optimized matching boxes with goal states. Additionally, some pre-processing was done in order to avoid any states that would result in a box getting stuck in a corner from which it could not move (ie, a box could not move into a position where it was surrounded by walls on two non-opposing sides unless that spot was a goal state). This not only sped up the planner, but also decreased the number of states visited.

---

4 Towers of Hanoi Revisited

Give successful plans from at least one planner with 10 and 12 discs.

Due to the large number of (2^n-1) states the plans for 10 and 12 disks are provided in the part4 directory listed as FFHanoi10Results.txt and FFHanoi12Results.txt respectively. The solution documents are easy to interpret as they are structured in a very simplistic format. Black box planner was tried to execute the 12 and 10 disc problems, but due to the size of the problem the time it takes for Blackbox planner is immense and the space was not sufficient. However FF planner provides the solution and the plan for 10 and 12 discs. The following is the time it took for the FF planner to provide the solution for ten and twelve discs respectively.



Ten Discs: 0.38 seconds
Twelve Discs: 5.13 seconds

Do you notice anything about the structure of the plans? Can you use this to increase the efficiency of planning for Towers of Hanoi? Explain

An observable difference, which can be observed, can be evident in the result files. The pictures below are screenshots from resulting plans for 10 and 11 discs. For all the odd numbered discs the plans place the discs on the first/destination pole, P1, whereas the even numbered discs are placed in the second pole P2. This is a trait, which can be observed for all the odd and even numbered discs for the Towers of Hanoi problem (a list of solution files are created from 3-12 discs). Another noticeable improvement in the FF planner is that the way it solves the problem, it mostly uses the brute force method to produce a solution to the problem. The brute force method can be generic and can apply to multiple problems, but the usage of a recursive implementation for the evaluation can be optimal. When given the problem of Towers of Hanoi an optimal solution would be

the Frame-Stewart Algorithm. Application to this algorithm to the planner would be optimal for a problem such as the Towers of Hanoi.

```
ff: found legal plan as follows

step    0: MOVE-DISK D1 D2 P2
        1: MOVE-DISK D2 D3 P1
        2: MOVE-DISK D1 P2 D2
        3: MOVE-DISK D3 D4 P2
        4: MOVE-DISK D1 D2 D4
        5: MOVE-DISK D2 P1 D3
```

```
-
0

ff: found legal plan as follows

step    0: MOVE-DISK D1 D2 P1
        1: MOVE-DISK D2 D3 P2
        2: MOVE-DISK D1 P1 D2
        3: MOVE-DISK D3 D4 P1
        4: MOVE-DISK D1 D2 D4
        5: MOVE-DISK D2 P2 D3
```

In a paragraph or two, explain a general planning strategy that would take advantage of problem structure. Make sure your strategy applies to problems other than Towers of Hanoi. Would such a planner still be complete?

Since the possible improvements include a recursive cyclic algorithm to break down the problem into smaller problem sets to find the solution, a generic recursive algorithm can be used to find the solution to the original problem. The recursive algorithm planner can have a terminal condition which can check for a point to stop, in the case of Towers of Hanoi the terminal condition can keep track of all the possible discs and if the disc count is equal to zero it can be considered that all the possible solutions have been dissolved. But the problems, which can arise through this is that, the amount of storage, which can be required, might be very large if the problem set is fairly large. Also another drawback of having a recursive algorithm planner is that the terminating or terminal condition needs to be provided and set so that the algorithm can find a solution if a solution exists. Finally through the recursive approach a planner can provide a thorough solution making it complete. Also another planner strategy, which can be applied, is, if there is enough memory to allow a bi-directional search making it fairly efficient to go through the graph, which can be generated through GRAPHPLAN to provide a set of solutions.

5 Towers of Hanoi with HTN planning

Formulate a HTN planning problem for the Towers of Hanoi

The HTN file used by the actual planner can be found in the source directory at ./part5/derplanner_copy/examples/hanoi.domain. The specification is slightly different than what was shown in class for HTN, so for convenience, the HTN specification is shown below. Note for instance, that the original file anonymously defines compound tasks within methods. Here, we write them out separately.

predicates:
- disc(d)
- loc(l)
- startloc(l)
- endloc(l)
- clearloc(l)
- clear(l)
- top(d)
- on_table(d)
- on_other(d)
- at(d, l)
- on(d, d)

primitive tasks:

move_single_to_clear_op(move-disc, xo, xf)
- add: clear(xo), at(move-disc, xf)
- delete: clear (xf), at(move-disc, xo)

move_top_to_clear_op(move-disc, disc-below, xo, xf)
- add: top(disc-below), at(move-disc, xf), on_table(move-disc)
- delete: clear(xf), at(move-disc, xo) on_other(move-disc), on(move-disc, disc-below)

move_single_to_top_op(move-disc, disc-other, xo, xf)
- add: clear(xo), at(move-disc, xf) on_other(move-disc), on(move-disc, disc-other)
- delete: top(disc-other), at(move-disc, xo), on_table(move-disc)

move_top_to_top_op(move-disc, disc-below, disc-other, xo, xf)
- add: top(disc-below), at(move-disc, xf) on(move-disc, disc-other)
- delete: top(disc-other), at(move-disc, xo) on(move-disc, disc-below)

compound tasks:
- Move-Stack-Recursive(bttm-disc, start, end, spare)
  - task: Move-Stack(bttm-disc, start, end, spare)
  - precond: disc(bttm-disc), disc(second-bttm), loc(xo) loc(xf), loc(xs), at(bttm-disc, xo) on(second-bttm, bttm-disc)

subtasks: {Move_Stack(second-bttm, xo, xs, xf), Move_Stack(bttm-disc, xo, xf, xs), Move_Stack(second-bttm, xs, xf, xo)}

Move-Single-Clear(bttm-disc, start, end, spare)
task: Move-Stack(bttm-disc, start, end, spare)
precond: disc(bttm-disc), loc(xo), loc(xf), loc(xs), at(bttm-disc, xo), top(bttm-disc), on_table(bttm-disc), clear(xf)
subtasks: {move_single_to_clear_op(bttm-disc, xo, xf)}

Move-Top-Clear(bttm-disc, start, end, spare)
task: Move-Stack(bttm-disc, start, end, spare)
precond: disc(bttm-disc), loc(xo), loc(xf), loc(xs), at(bttm-disc, xo), top(bttm-disc), on(bttm-disc, below-bttm), clear(xf)
subtasks: {move_top_to_clear_op(bttm-disc, below-bttm, xo, xf)}

Move-Single-Top(bttm-disc, start, end, spare)
task: Move-Stack(bttm-disc, start, end, spare)
precond: disc(bttm-disc), disc(disc-other), loc(xo), loc(xf), loc(xs), at(bttm-disc, xo), at(disc-other, xf) top(disc-other), top(bttm-disc), on_table(bttm-disc)
subtasks: {move_single_to_top_op(bttm-disc, disc-other, xo, xf)}

Move-Top-Top(bttm-disc, start, end, spare)
task: Move-Stack(bttm-disc, start, end, spare)
precond: disc(bttm-disc), disc(disc-other), loc(xo), loc(xf), loc(xs), at(bttm-disc, xo), at(disc-other, xf), top(disc-other), top(bttm-disc), on_other(bttm-disc), on(bttm-disc, below-bttm)
subtasks: {move_top_to_top_op(bttm-disc, below-bttm, disc-other, xo, xf)}

Move-All()
task: Move-Root()
precond: startloc(A), endloc(C), clearloc(B), on_table(bttm-disc)
subtasks: {Move_Stack(bttm-disc, A, C, B)}

methods:
Move-Stack(bttm-disc, start, end, spare)
Move-Root()

Describe the domain knowledge you encoded and resulting planning domain in detail.

The properties that I chose to represent with predicates were the locations of different discs (at), which discs were on others (on), if a disc was on top (top), if a location was clear (clear) and whether a disc was on the table or on another disc (on_table, on_other). Other predicates are either object checking (disc, loc) or used to specify the root task (startloc, endloc, clearloc). Notice that information about which disc is bigger or smaller is not specified. This is where the domain knowledge comes in, as we'll see later.

For this problem, we have four primitive actions. On a high level, there is only one basic action -- move a disc from one space to another. However, for the HTN

description to have adds and deletes work, we need to have four different primitives representing the four cases we might encounter. For all of these four primitives, they all have the result that the disc is at the new location, is no longer at the old location. Other effects depend on which action it is. move_single_to_clear_op moves a disc that is directly on the table to a clear space. The result of the action is that the old location is now clear, and the new location is no longer clear. move_top_to_clear_op moves a disc that is on top of another disc to a clear location. The result of the action is that the disc below the one you're moving is now top, the disc is now on the table, the new location is no longer clear, the disc is no longer on another disc and the disc is no longer on the disc below. move_single_to_top_op moves a disc that is on the table to a location on top of another disc. The result of the action is that the original location is now clear, the disc is now on another disc, the disc is on the other disc, the other disc is no longer top and the disc is no longer on the table. move_top_to_top_op moves a disc that is on top of a disc below and moves it on top of another disc. The result of the action is that the disc below is now top, the disc is now on the other disc, the other disc is no longer top, and the disc is no longer on the disc below.

Our specification has two methods or "abstract tasks," that specify an action but are simplified by compound tasks. The first is Move-Stack(bttm-disc, start, end, spare). The method moves a stack starting at bttm-disc, and moves bttm-disc and all discs above it from start to end, using spare as an intermediate location if necessary. The other method is Move-Root(). This is basically just the abstract "top" task that the HTN planner is trying to solve. It's only available task to complete it is Move-All().

Our specification has six compound tasks, five for Move-Stack and one for Move-Root. The first compound task is Move-Stack-Recursive(bttm-disc, start, end, spare). This compound task requires that the bttm-disc to have another disc on top of it (second-bttm). The task breaks down into three tasks - moving second-bttm and all above it to the spare location, moving bttm-disc to the final location and moving second-bttm and all above it to the final location. All of these are specified with Move_Stack so that the planner can later figure out which compound task to use. The four other compound tasks are Move-Single-Clear, Move-Top-Clear, Move-Single-Top and Move-Top-Top. These are all basically wrappers for the primitive tasks - they check that bttm-disc is top and that it is the case that the primitive action is build for (on table, moving to clear, etc.) It decomposes simply into the primitive task. Finally, Move-All() is the only compound task for Move-Root(). It basically just uses the startloc, clearloc, endloc literals to figure out what the "basic move" of towers of hanoi is and perform that.

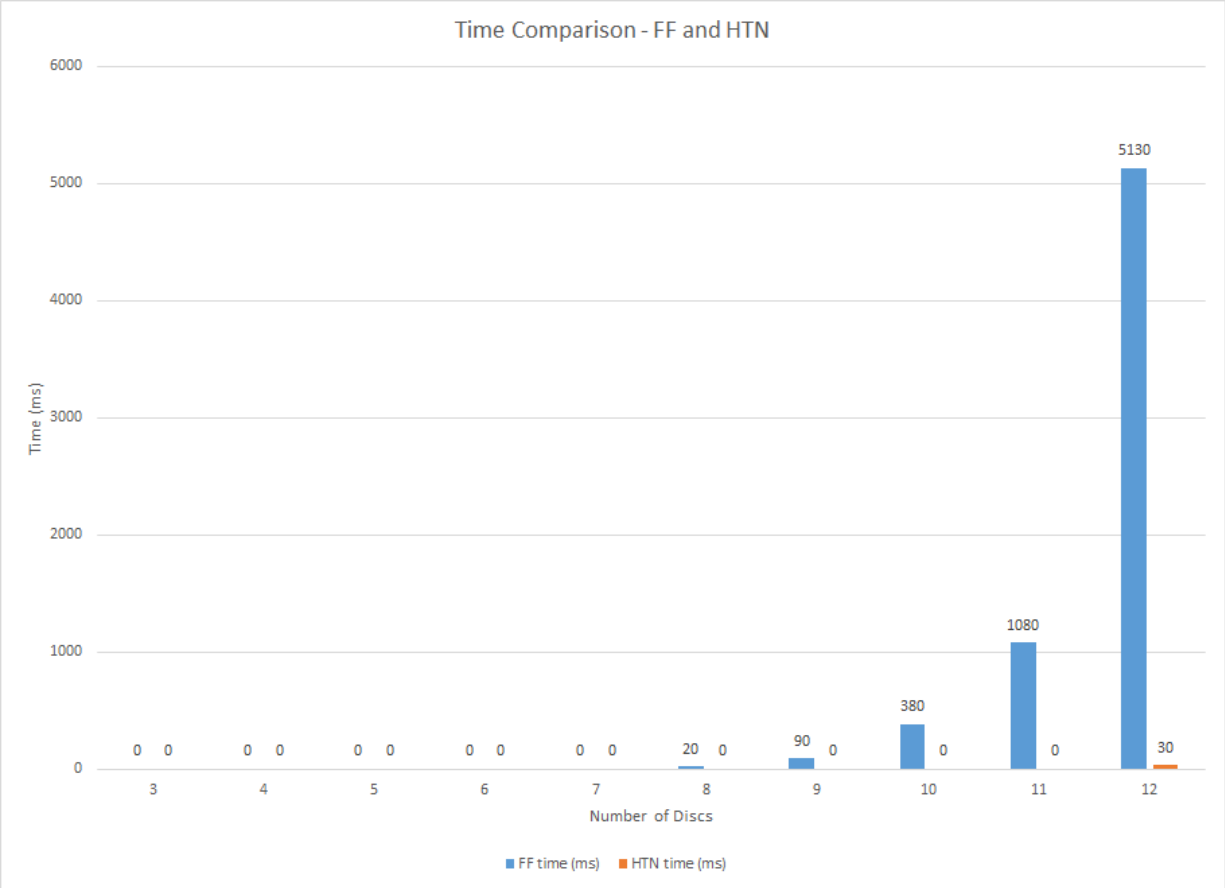
As we will discuss in greater detail we encode our knowledge that the best way to solve the problem is recursively. By encoding the recursive compound task, we are essentially encoding our knowledge about the solution, which is to move everything off the bottom disc, move the bottom disc and finally move the rest back on top of the bottom disc. In addition, because we put this breakdown in the HTN planner directly, no backtracking is required, so it can solve the problem very quickly.

Solve your HTN problem for the cases with 3 v 12 discs (use SHOP/SHOP2 or whatever HTN planner available to you).

See the results of the HTN planner for 3-12 towers of hanoi in the source directory at ./part5/Results.

Solve the same cases with a non-HTN planner of your choice, and compare the results with (c).

See the raw results of the non-HTN planner in the source directory at ./part5/Results



The HTN results were compared with the PDDL planner FF. In terms of finding the optimal solution, both found an optimal solution (a plan that took 2^N - 1 steps). This is somewhat surprising because FF doesn't guarantee optimality, since it's just a forward search with heuristic. The advantage of the Hanoi domain is that there aren't many suboptimal actions that are legal - they all basically boil down to moving a disc twice to get to the correct location. Because FF, unlike STRIPS, does have some backtracking and because it should be fairly obvious it's down a suboptimal path, it is able to find the optimal solution fairly easily.

The difference between the two then is in time. While both run fairly quickly for smaller numbers of discs, when you get to a large number of discs, the HTN planner starts to really do better. The HTN planner takes 0.03 seconds while FF takes 5.12 seconds. In addition, as you can see from the comparison graph, the FF planner starts to have worse end behavior as the number of discs increases as compared to HTN. Because FF does do some amount of searching through states, it is going to take longer as the

search graph gets larger. HTN, on the other hand should have linear end behavior (to the number of total moves). The main advantage of HTN for this domain then is speed.

Describe your observations and discuss about them.

As we discussed in part d, we saw significant speed performance improvements. This is because the subtask breakdown is unambiguous - i.e. there is no real choice as to what compound task you break down to, so the planner doesn't have to search. We also claim that we in fact get the optimal solution with the recursive solution. To see why, consider a stack of N discs to move from location A to C. We know that we need to move the N^{th} (bottom) disc of the stack to C. In order to do this, N must be top, so the discs 1 to N-1 must be on another peg. We also know that C must be clear. Therefore, All of the 1 to N-1 discs must be on peg B. Therefore, the quickest thing to do is to move the 1 to N-1 discs to B, move N from A to C and move the 1 to N-1 discs to C. This is in fact the optimal thing to do for all N. We can show that it's true trivially for N=2, and then we can prove this recursively using the logic from earlier. Other solutions are possible -- they end up being moving a "bottom" piece more than once to get to the final location -- but by restricting ourselves to these recursive moves, we greatly simplify the problem by not exploring other paths, which do not contain the optimal solution.
Another observation about this problem is that memory becomes a huge issue for many of these planners when doing this for 12 discs. You can show trivially that an N Hanoi problem requires 2^N - 1 moves. For 12 discs, this becomes 4095. Since a lot of planners have some kind of stack trace, this can mean that you need to store a ton of information as you plan. If you do something like depth-first search, you can end up storing an impossibly large amount of data. Even the HTN planner, we had to increase the allocated memory to get it to work correctly.

Works Cited

Hoffman, J., & Nebel, B. (2001). The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Articial Intelligence Research, 14*, 253-302. Retrieved September 28, 2014, from http://arxiv.org/pdf/1106.0675.pdf