CS 4649/7649 Fall 2014

Group 5 Homework 1

Trevor May, Valerie Reiss, Zachary Suffern, Abishek Tondehal, Eric Summins

Readme

Our answers are in this document. Our answers are also in our git repository in the answers directory. They are split up by which problem they are from. When the code or results for a specific problem is located in the git repository, it is marked as such with where it is located. For example, 2.a signifies where in our git repository the successful plans are located.

Who Did What

Trevor May

- Grand Master Wizard, Defender of the Faith, and Protector of the Realm
- Solved and typed out problems 2 and 5
- Ran most of the planners for both PDDL and HTN.

Valerie Reiss

- Wrote problem 2 PDDL domain
- Wrote the first problem 2 PDDL problem definition
- Helped to answer 2.b, 2.c, 2.d
- Wrote parser for problem 3 Sokoban planner
- Created state definitions for problem 3 Sokoban planner problems
- Helped to answer 4.b, 4.c
- Wrote problems 4.a, 5.d PDDL problem definitions
- Organized our write up

Zachary Suffern

- Answered Question 1
- Helped create PDDLs for Question 2
- Helped create HTN for Question 5
- Wrote and designed planner for Question 3
- Answered Questions 3.2 and 3.3

Abishek Tondehal

- Helped with Question 5
- Helped with Question 1
- Helped with Question 4

Eric Summins

- Implemented heuristic in planner
- Wrote improvements for both planner and parser
- Wrote/rewrote most of the answer to problem 3
- Helped work through number 5

Everyone

- Worked together to reason out how to develop planners, domains, problem definitions, and solve problems

1 Warming up: Towers of Hanoi

1.0: Work

We ran the Towers of Hanoi PDDL using Blackbox and VHPOP. We found that Blackbox ran quite quickly, while VHPOP ran significantly slower.

1.1: Questions

a. Explain the method by which each of the two planners finds a solution.

plan 1: Blackbox

Blackbox takes in a STRIPS file and then converts it to a boolean satisfiability problem. From there it attempts to find the theoretical amount of steps it would take to solve the boolean satisfiability problem and restricts the actual STRIPS search to a max amount of steps. If STRIPS fails, it checks to see if adding an extra step can still satisfy the boolean satisfiability problem and run STRIPS again. It continues to do this until STRIPS finds a solution.

plan 2: VHPOP

VHPOP is a partial order causal link (POCL) based planner that uses multiple flaw selection strategies simultaneously to discover the solution to the problem it has been given. By using multiple strategies at once, it is more likely to find a strategy that is dominant for that particular problem. When the problem starts VHPOP sets up each POCL with it's own type of flaw selection. Then it will run all the POCL's using A* as its search algorithm. From there it will continue until a solution is found.

b. Which planner was fastest?

Blackbox was faster than VHPOP. Blackbox accomplished the task in 9 milliseconds while VHPOP 4.8 seconds.

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

VHPOP does a lot of work up front. This requires more time then Blackbox, which does most of its work as its goes along. What this means is that even though the boolean satisfiability problem is NP-Complete, with the small amount of states this Towers of Hanoi problem has, it doesn't take a lot of time to solve. This means it can afford to figure out the solution as it goes and not take a lot of time. While VHPOP has to run multiple POCL flaw selection strategies at once. Which takes both relative large amounts of time to set up and to run. So even if the problem is not all too complex, it will still take time to solve.

2 Sokoban PDDL

2.0: Work

We created a PDDL sokoban domain and a problem definition for problems 2.1-2.3. We first tried to run them in Blackbox, but were unable to develop full solutions for all three problems. Blackbox was able to run the first and second problems successfully, but took a very long time. It

was unable to find a solution to problem 2.3. We then ran IPP and MIPS using our definition language. Both IPP and MIPS were able to successfully develop plans for all three Sokoban problems.

2.1: Questions

a. Show successful plans from at least one planner on the three Sokoban problems.

See answers/problem2.txt in our Git repository for the successful plans for both IPP and MIPS.

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

The two planners chosen for comparison are IPP and MIPS. IPP is a Graphplan descendant parallel regression planner, and MIPS is a BDD-based planner that performs forward-chaining and breadth-first search. Both planners are optimal and this showed in the fact that both computed the minimum number of steps needed to solve the problems. However, it was clear in the total time taken that IPP computed the solution much faster than MIPS. IPP took 0.00, 0.04, and 0.58 to solve problems 1, 2, and 3, respectively. Conversely, MIPS took 0.57, 0.64, 11.29 on those problems 1-3. The versions of MIPS and IPP tested both were developed around the year 2000 and thus there is not expected to be a major advantage for one or the other because of more knowledge or better computers. Instead, taking a look at the internals of the algorithms taken by the planners will tell why IPP is faster. For example, IPP is a "parallel planner" which means that actions that do not interfere with each other are run simultaneously. Thus, with each time step, one or more actions may be completed. This is a huge advantage over the sequential BFS performed by MIPS. Also, IPP computes an admissible heuristic with each problem by developing a graph, level by level, in order to see what actions can be computed at each time step to make the desired goal states to be true.
It does not look like MIPS uses any sort of admissible heuristic in its BFS.

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

The difficult thing comes from the fact that these planners like to define everything in forms of states. This is great in terms of being very explicit and detailed in every part of solving the problem. The issue comes when those states grow exponentially with all the different actions that can be taken at any one state. In the case of the sokoban problem, states must be made for each of the possible moves that the robot can make. Each of those actions then have nested actions and states that must represent them. Thus, the memory space needed for these graphs as well as the time for constructing them can be quite large. Semantic planning excels at problems that are far more finite and fit well to every state and situation of the problem being pre-planned before simple path searching algorithms are employed.

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

Problems that have only one action are those that suit themselves to classical planning. Additionally, the possible variables that can be filled in as arguments for those actions needs to be limited. This is because a minimal number of states is the chief goal for a classical planner that wants to solve things quickly. Thus, the blocks world problem where one attempts to move a stack of blocks from one configuration to another is an example of something that achieves this goal. The only action is to move a block, and there are only a small handful of possible blocks that can be moved at any one time. Another problem that would be solved well with classical planning is the "Number Slider Puzzles" that are 4x4 grids with 15 'sliders' in them that are numbered 1-15 with one open space. The goal is to get the numbers in order starting in the upper left hand corner and finishing in the bottom right hand corner. The only action would be to move a slider and at most you would have four options to choose from, thus achieving the goals of minimal actions and options to take for those actions. An admissible heuristic could also be constructed by desiring to have the numbers move to their end goals.

3 Sokoban Challenge

3.0: Work

We wrote a modified breadth first search algorithm for this problem. We input a map against which the robot can check the legality of its moves. Our states were defined as the set of locations of the robot and all boxes, and our transitions were any possible move the robot could make in any given state. As such, we did not have to define each location and its status as we would in PDDL. This reduction in state space was sufficient to allow a breadth first search algorithm to solve the given problems. From there we added heuristics to further reduce the state space, which reduced the time required for the breadth first search to run.

3.1: Questions

a. Give successful plans from your planner on the Sokoban problems

See git repository HW1_Team5 / planner_output / ourPlanner for the solutions.

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

Planner	**Problem 1 Time**	**Problem 2 Time**	**Problem 3 Time**	**Challenge Time**
Our Planner	0.00038 seconds	0.0012 seconds	0.0256 seconds	0.045 seconds
MIPS	0.57 seconds	0.64 seconds	11.29 seconds	n/a
IPP	0.00 seconds	0.04 seconds	0.58 seconds	n/a
Blackbox	0.07 seconds	16 minutes, 7 seconds	unable to find solution	n/a

Our planner was faster than the PDDL planners and found the solution while searching fewer nodes than any of the others. This is due to the fact that in our planner, we have limited the total amount of options the robot could take to 4: move left, right, up, or down and a transition function turns those actions into new states. This greatly diminishes the amount of possible states that are there unlike the PDDL's which have to compare each action to each other action multiple times, which means the state space grows much faster then the planner. So in the end our planner is faster since it has less comparisons to make and states to explore.

c. 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.

Our planner, at its essence is nothing more than a Breadth First Search algorithm. The algorithm itself searches through states to find a solution to whatever problem is given to it. It does this by defining the states as the locations of both the robot and all blocks present in the problem. The world of the sokoban problem is of finite size, with a finite number of permutations of block and robot positions, meaning there is a finite number of states that the algorithm can explore. Since we are using a Breadth First Search in a finite state space the algorithm is complete; it will return a solution if one exists. Since the algorithm will eventually search all possible state spaces, it will necessarily return no solution if no solution exists.

It's worth discussing the fact that different move sets may reach the same state space using a different number of moves, and our planner does not allow for the revisiting of specific state spaces. The important fact in this problem is that if a solution exists, the planner returns it, and if no solution exists the planner returns null. As such, it does not matter if there are multiple ways to reach a particular state space. We only care if there are zero solutions or greater than zero solutions. There is no meaningful difference between having one solution and thirty solutions. A side effect of using a breadth first search is this case is that it will always output one of the solutions with the smallest number of moves.

d. 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.

We implemented a single heuristic to eliminate certain decision chains from the breadth first search. We noticed that pushing a block into a corner (walls on two sides) which is not a goal state resulted in an impossible puzzle. The block could not be removed from the corner, so that decision chain is thus dead. This heuristic is most relevant in situations with lots of corners and lots of open space. As such, implementation resulted in decent (approximately 30 to 50%) improvements for problems 1, 2, and 3. They involved only a moderate number of corners which could result in dead chains. However, the challenge problem involved open space with lots of corners, which was particularly relevant to this heuristic. As such, this problem showed an improvement of approximately 94%, going from 55923 states checked to 3441 states.

4 Towers of Hanoi Revisited

4.0: Work

We first created a PDDL problem state for 10 and 12 discs, then ran them using IPP. From there we looked at the results. We saw that there were a few patterns, both at an individual movement level and at a larger level. First, Hanoi itself is recursive. For example, when computing Hanoi(N), it moves(calls Hanoi(N-1)) the first N-1 discs to the transition location, moves the Nth disc, then moves the first N-1 discs on top of the Nth disc to the goal state. Each movement can be recursively defined, down to the base state to move only one disc. Further, the discs always move in the same direction between P1, P2 and P3. For example, the smallest disc always moves from P1 to P2, from P2 to P3, and from P3 to P1.

4.1: Questions

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

See git repository HW1_Team5 / planner_output / IPP for output for Hanoi 10 and 12.

b. 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.

Hanoi works by recursively moving all but the bottommost disc from one stack to another. Each time it does this, it then moves the previous bottom disc to the target stack, and starts over on the new smaller stack. Therefore, it would be possible to increase the efficiency of planning for Towers of Hanoi by creating a move action that performs the move for an N-disc Towers of Hanoi problem on the first N-1 discs, moves the Nth disc, then calls move again on those N-1 discs until there are no discs left to process. This means that the goal state of Hanoi(N) does not need to be to move every single disc onto the new location, but simply to move the bottommost disc there and then call Hanoi(N-1) on the remaining discs.

In addition, for even number of discs in the problem space, the discs always travel in the same direction when moving: either clockwise or counterclockwise. In Hanoi-10 and Hanoi-12,

odd numbered discs always move from P1 to P2, P2 to P3, and P3 to P1. Even numbered discs, on the other hand, always move from P1 to P3, P2 to P1, and P3 to P2.

c. 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?

This type of problem is expanded by simply adding an additional portion to the initial problem, which allows for recursion to be easily used to solve it. Once the Nth level is solved, the remaining solution is just that of the N-1th level. In Hanoi, that means that once the first N-1 discs are moved and the Nth disc is put into place, it can then be solved by considering the state to now be Hanoi(N-1). This type of recursive problem solution can be used in other cases, such as solving for the Nth term in a sequence or sorting. A sorting algorithm that utilizes a recursive structure could sort N elements by first sorting the first N-1 elements, then adding the Nth element. Some other problems where we can use such a technique is the 8 queen puzzle and sudoku. Solving these problems recursively allows us to avoid unnecessary moves.

This is complete because it allows you to solve for any number of discs because the base case is defined.

5 Towers of Hanoi with HTN planning

5.0: Work

We started by looking at what we learned from problem 4. There is a definite pattern between how the discs move. When using an odd number of discs, discs moved clockwise. An even number of discs, however, discs moved counterclockwise. To find a HTN planning problem, we had to look at our goal task, and then figure out which methods we should define to get the goal task to work. We decided that our HTN goal task would be to move 1-N discs from the start peg to the goal peg using the other peg. Each compound task would be to move 1-X discs from one peg to another using a third peg, and the primitive tasks would be to simply move one peg to another. Another way to think of this is that because even discs move one way and odd discs move another and there is a definite pattern to the order that the discs move in because of the recursive design of the problem, we are able to dictate only one specific move for each state. We then worked to actually implement this logic into a working HTN planner. We created a move-disc method that calls move disc for N-1 discs until there is only one disc left.

5.1: Questions

a. Formulate a HTN planning problem for the Towers of Hanoi.

It is located in our git repository at HW1_Team5/hanoi.lisp

b. Describe the domain knowledge you encoded and resulting planning domain (i.e. primitive tasks, compound tasks, and methods) in detail.

We encoded which discs were on which tower. We did not actually encode the relative sizes of the discs. We did encode which disc was the smallest, since this was required to act as a terminating case for the recursive call.

Primitive Tasks: We implemented a single primitive task, the Move task. This task takes as inputs a disc, a source tower, and a destination tower. It places the disc on the target tower and removed the disc from its source tower.

Compound Tasks: We use no compound tasks in our HTN plan. Our planner uses a single recursive method that calls itself and our primitive task.

Methods: We implemented one recursive function called MoveMultipleDiscs. This method takes in a list of discs, the source tower, the free tower, and the destination tower. It then checks to see if the current disc is the smallest disc. If it is then then it calls the move task, else it will call moveMultipleDiscs for N-1 disc,
move for the current disc,
and finally calls moveMultipleDiscs for N-1 disc again.

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

See the results of our HTN in our git directory in answers/hanoiHTNoutput.txt

# of Discs	**Time**
3	0.0712 seconds
4	0.010 seconds
5	0.005 seconds
6	0.018 seconds
7	0.068 seconds
8	0.277 seconds
9	1.198 seconds
10	4.945 seconds
11	20.98 seconds
12	heap overload

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

See the results of the non-HTN planner in our git directory planner_output/IPP

# of Discs	**Time of HTN Planner**	**Time of IPP**
3	0.0712 seconds	0.00 seconds

4	0.010 seconds	0.00 seconds
5	0.005 seconds	0.00 seconds
6	0.018 seconds	0.02 seconds
7	0.068 seconds	0.08 seconds
8	0.277 seconds	0.3 seconds
9	1.198 seconds	1.32 seconds
10	4.945 seconds	4.64 seconds
11	20.98 seconds	21.70 seconds
12	heap overload	72.98 seconds

e. Describe your observations and discuss about them.

In general the HTN is faster than the IPP planner but not by much. We credit this to the fact that HTN has constraints in place unlike the IPP planner. This meant that the HTN planner had less options it had to consider. In fact we limited the HTN to one possible legal move at any space state. On the other hand, the pddl planner would use A* and attempt each and every permutation of actions available at any one step. The downside to this is that we have to hold all the search space in memory unlike IPP. This in fact lead to a heap overload for the HTN planner as it ran out of memory and couldn't complete problem 12. We are in essence trading memory space for some speed.

While the speed gain with HTN is nice, the fact that it already runs into memory space issues with twelve discs is disturbing. As the professor as stated, it is often more important for planners to be complete than optimal. Thus, there clearly appears to be an emphasis on the practicality and reliable nature of planners. Not blowing the heap would run right into that scenario and thus we would suggest the pddl planner where optimal times is not the prime goal.