Project 1

Robot Intelligence: Planning 8803 **"RIP"**

Philip Rogers, Alex Cunningham, Oktay Arslan

September 21, 2009

2.) Pre-Project: Towers of Hanoi

1. *Explain the method by which each of the two planners finds a solution.*
   The Fast-Forward (FF) planner works by combining HSP with a heuristic using GraphPlan. Following the HSP method, the FF planner grows the state space while ignoring delete lists. A difference between HSP and FF is that the heuristic used in FF is based on a relaxed version of GraphPlan, which is a more informed heuristic than HSP because it takes into account positive interactions. An example of this is given in [2], where the naive heuristic does not consider a single precondition being shared between two actions, whereas GraphPlan considers it and produces a lower (more informed) heuristic value.

   The Blackbox planner also uses GraphPlan, but in a very different way than FF. Blackbox uses GraphPlan (especially Mutex constraint propagation) to generate a boolean satisfiability problem, which Blackbox then uses SAT solvers to solve. This is clear from the output of Blackbox, which shows how it works: first GraphPlan is used to generate a SAT problem, then a SAT solver is used to determine the feasibility.

2. *Which planner was fastest?*
   We used two planners:Fast-Forward (FF), and Blackbox.
   FF is clearly the fastest planner, with two of its variants (FFv2.3 and Contingent-FF) taking less than 0.01s to generate a plan for the 3-disk Hanoi problem. The Blackbox planner was considerably slower, taking 0.032s when using the Chaff solver, and taking 0.10s when using the SatPlan solver. All four generated the same plan, which was 7 steps.

3. *Explain why the winning planner might be more effective on this problem.*
   The heuristic used by FF appears to be particularly useful in this small Hanoi problem, because positive effects are frequent when dealing with only 3 plates. Dr. Hoffman mentions this in [2] when he notes that the heuristics of FF are good for many benchmark planning problems, such as Hanoi.

3.) Project Part I: Sokoban PDDL

1. *Show successful plans from at least one planner on the Sokoban problems.*
   We tried each of the Sokoban problems with the two planners, with each of the planner variations. It should be noted that a solution was not always reached, and for the harder problems some of the planners were killed manually to prevent the system from running out of memory. Due to size of the

plans generated for the later problems, the results in the form of raw logs are in a separate file. Here are sample plans for each of the problems:

(a) *Problem 1 - Plan Produced by BlackBox with Default Options*

```
Begin plan
1 (move-down player p3 p5)
2 (move-right player p5 p6)
3 (move-right player p6 p7)
4 (move-down player p7 p11)
5 (move-block-left player a p11 p10 p9)
6 (move-up player p10 p6)
7 (move-left player p6 p5)
8 (move-left player p5 p4)
9 (move-down player p4 p8)
10 (move-down player p8 p12)
11 (move-right player p12 p13)
12 (move-block-up player a p13 p9 p5)
13 (move-block-up player a p9 p5 p3)
14 (move-block-up player a p5 p3 p2)
End plan
```

(b) *Problem 2 - This problem is reported by all solvers (computer and human) as being impossible*

(c) *Problem 3 - Plan Produced by FF with Default Options - See SolverPlans.txt*

(d) *Problem 4 - Plan Produced by FF with Default Options - See SolverPlans.txt*

2. *Compare the performance of two planners on this domain. Which one works better? Does this make sense, why?*
The performance of the planners is numerically described in the table below:

Planner stats for Sokoban problems				
	FF (default)	Contingent FF	BlackBox (default)	BlackBox (chaff)
Problem 1 Time (ms)	0.00	0.00	381	309
Problem 1 Plan Length (steps)	14	14	14	14
Problem 2 Time (ms)	-	-	1077	1088
Problem 2 Plan Length (steps)	-	-	-	-
Problem 3 Time (ms)	40	-	-	-
Problem 3 Plan Length (steps)	105	-	-	-
Problem 4 Time (ms)	1230	-	-	-
Problem 5 Plan Length (steps)	79	-	-	-

Table 1: Statistics for the planners solving PDDL Sokoban.

It should be noted that because Problem 2 is not solvable, the FF variants did not report a time for completion. For Problems 3 and 4, there were no solutions generated by any planner other than FF, at least in the time given. These planners start using very large amounts of memory and will quickly stall a computer if left to run too long without finding a solution.

The fact that FF works better than BlackBox's SAT-based solver is probably due to the translation of this particular problem into satisfiability space increases the complexity and dimensionality of the problem. From the previous results of the simple Towers of Hanoi puzzle, it appears that the implementation of FF used is generally faster than BlackBox at solving problems. One observation to note is that an earlier version of the PDDL domain used implemented move commands with *move* and *move-block* actions with an OR statement in the preconditions. FF was able to both parse and execute using this version of the domain (though it was unable to solve problems 3 and 4), while BlackBox could not run. The final version of the domain used separate *move* commands for each direction, which produced much faster results in FF, and worked with BlackBox. This would seem to indicate that more complex preconditions for actions (and the move commands had a large number of preconditions) pose a larger challenge for BlackBox than for FF, likely due to the conversion to SAT format that BlackBox needs, rather than using a direct heuristic. This result also mimics the most recent results for the international planning competition, with the version of FF used placing second behind a derivative of FF and BlackBox not in the top spots.

3. *Clearly PDDL was not intended for this sort of application. Discuss the challenges in expressing geometric constraints in semantic planning.*
   The major difficulty in expressing geometry with PDDL is that all of the relationships between cells need to be explicitly described, which can be rather tedious as the size of the puzzle increases. The domain used manages to keep the links in rows and columns relatively concise, but certain intuitive concepts like adjacency that are easy to express in geometric spaces are more complicated to express in PDDL. The frame problem also presents an issue here, as each cell in the grid needs to have an explicit property of having a block or not so that obstacles can be detected. In order to move a block, the property of having a block needs to be updated from the block's start to the end pose, and the block object itself needs to be moved so that an *at* predicate can evaluate a specific block's location. While it is possible to create the necessary relationships between cells and evaluate those relationships to ensure that the player and the blocks can move in the environment, it seems overly complicated to express even a grid world by manually constructing what are geometrically intuitive constraints.

4. *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.*
   The basic split between when one planning representation is more useful is a rather straightforward one: domains with intuitive geometric constraints (such as the positions of objects in a grid or continuous space) are ones where geometric representations are most useful. In non-geometric cases, semantic representations are effective. One previously used example of a case where semantic planning is more reasonable is in the simple case of Towers of Hanoi. In this case, the rules governing the system are arbitrary constraints that could be described more as "rules of a game" than as intuitive representations of reality. The geometry of the disks on the pegs is largely irrelevant, (e.g., the distance between the pegs and the relative position of pegs has no bearing on how moves can be made), as the problem has been abstracted away from the geometric constraints. In this abstract space of largely arbitrary rules acting on abstract objects, semantic planning is really the only effective and robust representation method. This is in contrast to geometric space, where basic concepts of moving objects and obstacle avoidance are based out of a representation of a very physical system.

4.) Project Part II: Sokoban Planner [Description of our planner]

Planner stats for Sokoban problems				
	Problem 1	Problem 2	Problem 3	Problem 4
Computation time (ms)	1	2	3	1
# of steps	4	5	6	1
# of states explored	7	8	9	1

Table 2: Statistics for our Sokoban planner.

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

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

3. *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 relies on a straightforward implementation of the A* algorithm with an admissable heuristic. Our heuristic is admissable because we simply use a heuristic of 0, and prune "dead-ends" in the graph, which maintains the completeness properties of A*.

   Proof by contradiction: Let A* use an admissable heuristic, which underestimates the cost to the goal.
   Assume that A* does not reach the goal, though there is a path to the goal.
   If there is a path to the goal, at some point the robot decided to deviate from the route towards the goal, and instead took a different route.
   But with an admissable heuristic, this is impossible.
   Therefore, A*, using an admissable heuristic, will find a plan to the goal if it exists.

   The above simply proved that A* will find a solution if it exists. In the case that no solution exists, A* will expand every node, but will never re-visit nodes. This property allows A* to search the entire space, and return that no solution exists once the search space has been exhausted.

4. *What methods did you use to speed up the planning? Give a short description of each method and explain why it did or didnt help on each relevant problem.*
   We chose to use a heuristic that prunes "dead-ends" in the graph. To accomplish this, we located several situations specific to the Sokoban domain where an action would prevent the goal from being reached, and pruned those actions by using a large heuristic value. These dead-end actions included:

   (a) Pushing a block into a configuration where it cannot be removed.

   (b) Blocking the robot in a configuration where it cannot escape and cannot reach the goal.

5.) Post-Project: Towers of Hanoi Revisited

1. *Give successful plans from at least one planner with 6 and 10 disks.*
   See attached *hanoi_plans.txt*.

2. *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.*
   Yes, the plans have long spans where a large number of disks are "shuffled" from one peg to another, inverting all but one of the disks in a tower. To improve the efficiency of the planner, a move could be introduced to move all but one disk from one peg to another. This new move corresponds to one of the steps in the well-known Hanoi solving algorithm [1], which has been shown to be optimal (in terms of efficiency, and optimality.)

3. *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?*
   One strategy would be to locate sub-plans in an on-line method, and add the sub-plans as moves for the remainder of the plan. Consider the Hanoi problem: if the planner could recognize that portions of towers were frequently being inverted, a "meta-move" of *invert-partial-tower* could be added to the available moves, which would make future planning much more efficient by allowing one move to jump several steps ahead in the plan. In the case of an incorrect meta-move, the algorithm would simply determine that the meta-move was incorrect, and would continue searching using it's other moves. This strategy would still be complete, as the algorithm could always fall back onto it's previous strategy.

References

[1] Alexander Bogomolny. Tower of hanoi. http://www.cut-the-knot.org/recurrence/hanoi.shtml.

[2] J. Hoffmann. The fast-forward planning system. *AI magazine*, 22(3), 2001.