CS 4649/7649: RIP - Robot Intelligence - Planning
PROJECT 1: CLASSICAL SOKOBAN PLANNER

Arvind Krishnaa Jagannathan, Zheng Yong, Luis Gustavo, Zhengyi Hu

1 Pre-Project: Towers of Hanoi

Planners Used

The two classical planners which we are using for the Towers of Hanoi problem are the Blackbox planner [1] (downloaded from `http://www.cs.rochester.edu/~kautz/satplan/blackbox/blackbox-download.html`) and the FF planner [2] (downloaded from `http://fai.cs.uni-saarland.de/hoffmann/ff/FF-v2.3.tgz`). The definition of the Towers of Hanoi domain, as well as the representation of the initial state of the problem (from Figure 1) are in the corresponding PDDL files, namely *hanoi-domain.pddl* and *hanoi-3.pddl*.

Figure 1: Towers of Hanoi with 3 disks

1.1 Questions

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

BlackBox- The Blackbox planning algorithm essentially represents the PDDL representation of a problem as a set of Boolean satisfiability problems (SAT), which are then solved using a variety of SAT solvers (such as satz, walksat and chaff) to produce a plan. In order to construct the SAT problem, Blackbox uses the GRAPHPLAN algorithm to construct a plan graph for the given problem representation. This plan graph is then "translated" to the SAT problem representation. This is outlined briefly below:

The GRAPHPLAN algorithm works by constructing a planning graph out of a STRIPS representation, which will propagate actions/operators across different "layers" along with their "mutex" pre-conditions. So at every stage there will be a list of actions, which have mutually exclusive pre-conditions. The plan generated by GRAPHPLAN will be a sub-graph of the plan graph such that, all the conditions of the initial and final state are incorporated without there being any "conflicting" actions. As described by Kautz et. al [3], each state in this solution graph can be encoded into propositional logic. This can be done by adding propositions of the form,

$$Precondition_i \implies Action_j$$

At each layer of the solution sub-graph, clauses/fluents can be resolved away to result in a compact propositional logic representation of the plan graph. This list of proposition corresponds to the "translation" of the plan graph into a SAT problem. Then any available SAT solver can be applied on this problem which (although NP-complete in theory) with reasonable assumptions be completed in exponential time. ($O(n^3)$)

FF Planner - The fast-forward planning algorithm utilizes GRAPHPLAN as an admissible (and informed) heuristic to solve the planning problem using a search algorithm across the set of permissible states in the state space. FF planner works as follows: it sets up a relaxed solvable sub-problem ($S\prime$) of the original problem (S).Then GraphPlan is applied on the relaxed sub-problem; the length of the solution plan sub-graph is then treated as a heuristic to guide the state space search algorithm for a plan to the original problem. This takes into account the positive interactions between various facts in the problem.

FF uses the enforced hill climbing algorithm to search for valid solutions. Using the length of the relaxed GraphPlan solution as a heuristic, the enforced hill climbing algorithm evaluates the direct successors of a search state S. Until a state ($S\prime$) is found with a better heuristic evaluation than S, the search goes one step further.

Thus in summary both ***FF*** and ***Blackbox*** algorithms use the GraphPlan algorithm, however *Blackbox* uses it in a more direct way, to set up the SAT problem. In *FF*, GraphPlan is used simply as a heuristic measure for the enforced hill climbing algorithm.

2. Which planner was fastest?

Both the algorithms were run on the same Linux box, and using the *time* command the time taken for them to produce plans were measure. Clearly the **FF** algorithm was the fastest among the two, with the following time measures

- FF - 0.004s
- Blackbox - 0.011s

This shows that **FF** is 2.75 times faster than **Blackbox**. This is pretty much expected from theory since the time complexity for each of the algorithms are,

$$\begin{gathered} \mathrm{O}(\textit{Blackbox}) = \mathrm{O}(\textsc{GraphPlan'}) + \mathrm{O}(\textit{SAT Solver}) \\ \mathrm{O}(\textit{FF}) = \mathrm{O}(\textsc{GraphPlan}) + \mathrm{O}(\textit{Enforced Hill Climbing}) \\ \mathrm{O}(\textit{Enforced Hill Climbing}) = \mathrm{O}(\text{iterations x successors}), \\ \mathrm{O}(\textit{SAT Solver}) = \mathrm{O}(2^n) \end{gathered}$$

Clearly the SAT solver's exponential time complexity, as well as the time complexity of the complete GraphPlan algorithm for *Blackbox* vs. that of the relaxed GraphPlan as well as that of *Enforced Hill Climbing* for the *FF* make it obviously slower than FF; this has been shown by using the towers of hanoi state and domain PDDL description multiple times (to get an average measure) with Blackbox and FF.

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

Its pretty obvious that the length of a plan in case of the Towers of Hanoi problem are a similar order of magnitude as that of the total number of states ($2^n - 1$ steps and 3^n states). In case of the **FF** planner, GraphPlan is used only as a heuristic and that too for a relaxed subset. *FF*'s major component is the local-search algorithm, which means it does not have to necessarily traverse all the states in order to obtain a solution.

Blackbox on the other needs to explicitly construct the mutex graph for every level until a solution is obtained. In case the state space is large (like 10 disks in the Towers of Hanoi problem), the construction of a complete plan graph is memory prohibitive and will not be effective on larger instances of this problem (it actually does not give plans for even 6 disks on my machine after 1 minute of execution). Another major issue with *Blackbox* is that several levels of the GraphPlan algorithm may lead to the same set of SAT problems, which will remain unsatisfiable. *Blackbox* is bound to be less effective in similar large state problems, since it seems to perform the same computation multiple times.

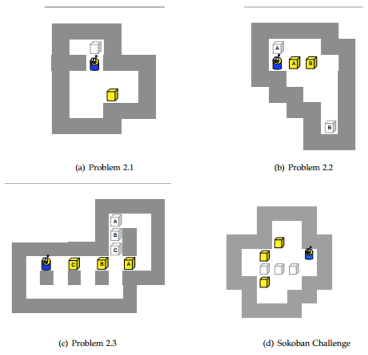

Figure 2: Sokoban Problems

2 Project Part I: Sokoban PDDL

2.1 Questions

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

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

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

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 semantic (classical) planning. Explain why a semantic representation would be desirable

3 Project Part II: Sokoban Planner

3.1 Questions

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.

4. 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

4 Post-Project: Towers of Hanoi Revisited

Constructing a PDDL representation for the N-disk towers of hanoi is pretty simple by utilizing the following simple structures, for initializing the problem state

1. Each disk labeled D_i is smaller than a disk labeled D_{i+1}. That is `(smaller` d_i d_{i+1}`)`.
2. Every disk is smaller than each of the three poles (by definition).
3. The smallest disk (i.e., D1), and the two other poles (P1 and P2) are clear.

4. Every disk D_i is *on* the disk D_{i+1}. The largest disk D10, is *on* the pole P3.

5. Every element D_i is a disk

The goal state is simply a conjunction (AND-ing) of all the states mentioned in Step 4 above, except that the largest disk D10, is *on* the pole P1.

The corresponding PDDL representations for the 6 disk and 10 disk towers of hanoi problem are present in the files "hanoi-6.pddl" and "hanoi-10.pddl".

Both **FF** and **Blackbox** planners were applied onto the PDDL representations, however only *FF* was able to produce valid plans for the 6 disk and 10 disk problem. *Blackbox* was unable to produce any results, due to relatively large nature of the state-space (I halted execution after 1 minute since the blackbox executable started running).

4.1 Questions

1. Give successful plans from at least one planner with 6 and 10 disks

The plan for the towers of hanoi problem with 6 disk, generated by *FF* is 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
        6: MOVE-DISK D1 D4 D2
        7: MOVE-DISK D4 D5 P1
        8: MOVE-DISK D1 D2 D4
        9: MOVE-DISK D2 D3 D5
       10: MOVE-DISK D1 D4 D2
       11: MOVE-DISK D3 P2 D4
       12: MOVE-DISK D1 D2 P2
       13: MOVE-DISK D2 D5 D3
       14: MOVE-DISK D1 P2 D2
       15: MOVE-DISK D5 D6 P2
       16: MOVE-DISK D1 D2 D6
       17: MOVE-DISK D2 D3 D5
       18: MOVE-DISK D1 D6 D2
       19: MOVE-DISK D3 D4 D6
       20: MOVE-DISK D1 D2 D4
       21: MOVE-DISK D2 D5 D3
       22: MOVE-DISK D1 D4 D2
       23: MOVE-DISK D4 P1 D5
       24: MOVE-DISK D1 D2 D4
       25: MOVE-DISK D2 D3 P1
       26: MOVE-DISK D1 D4 D2
       27: MOVE-DISK D3 D6 D4
       28: MOVE-DISK D1 D2 D6
       29: MOVE-DISK D2 P1 D3
       30: MOVE-DISK D1 D6 D2
       31: MOVE-DISK D6 P3 P1
       32: MOVE-DISK D1 D2 D6
       33: MOVE-DISK D2 D3 P3
       34: MOVE-DISK D1 D6 D2
       35: MOVE-DISK D3 D4 D6
```

```
36: MOVE-DISK D1 D2 D4
37: MOVE-DISK D2 P3 D3
38: MOVE-DISK D1 D4 D2
39: MOVE-DISK D4 D5 P3
40: MOVE-DISK D1 D2 D4
41: MOVE-DISK D2 D3 D5
42: MOVE-DISK D1 D4 D2
43: MOVE-DISK D3 D6 D4
44: MOVE-DISK D1 D2 D6
45: MOVE-DISK D2 D5 D3
46: MOVE-DISK D1 D6 D2
47: MOVE-DISK D5 P2 D6
48: MOVE-DISK D1 D2 P2
49: MOVE-DISK D2 D3 D5
50: MOVE-DISK D1 P2 D2
51: MOVE-DISK D3 D4 P2
52: MOVE-DISK D1 D2 D4
53: MOVE-DISK D2 D5 D3
54: MOVE-DISK D1 D4 D2
55: MOVE-DISK D4 P3 D5
56: MOVE-DISK D1 D2 D4
57: MOVE-DISK D2 D3 P3
58: MOVE-DISK D1 D4 D2
59: MOVE-DISK D3 P2 D4
60: MOVE-DISK D1 D2 P2
61: MOVE-DISK D2 P3 D3
62: MOVE-DISK D1 P2 D2
```

Clearly it has $2^6 - 1 = 63$ steps in the plan. Now the problem with 10 disks will have a plan of $2^10 - 1$ $= 1023$ steps. The *FF* planner produces a valid plan with 1023 steps, but for the lack of space it is not produced here. The plan is present under the resources directory as "hanoi-10-solutions".

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

One observation from the three plans generated (3, 6 and 10 disks) is that whenever there are odd number of initial disks, then the top most disk is moved onto the "destination" pole, and when there are even number of initial disks, then the top most disk is moved onto the "middle" pole.

Another noticeable aspect of the problem is that the towers of hanoi can be viewed as a simple recursive problem of moving the smaller $n-1$ disks from $P3$ to $P2$ then moving the largest disk from $P3$ to $P1$ followed by moving the $n-1$ disks from $P2$ to $P1$. This can be empirically verified - at every $(2^N - 1)^{th}$ step, if the initial number of disks n are odd, then there will be $N-1$ disks on P2 and largest disk will be on P1. If the initial number of disks n are even, then there will be $N-1$ disks on $P3$ and the largest will be on $P2$. Of course here, $0 < N <= n$.

Thus the entire plan can be represented recursively (and efficiently) as:

Algorithm 1 Towers of Hanoi recursive definition

```
1: function HANOI-SOLVER($N, P3, P2, P1$)    ▷ Move N disks from P3 to P1 (using P2 as intermediate)
2:     HANOI-SOLVER($N-1, P3, P1, P2$)
3:     MOVE($1, P3, P1$)                    ▷ Move the largest disk from P3 to P1
4:     HANOI-SOLVER($N-1, P2, P3, P1$)
```

Thus this is a bottom-up approach in constructing a plan. All one needs to define are the "macro" propositions HANOI-SOLVER and MOVE (which is applied when there is just the largest disk remaining on

$P3$). The planner can then use this base condition (a.k.a "macro" proposition) and basically "un-wind" the call stack to generate sub-plans. The correct plan is then obtained by simply reversing the "popped" elements of the call stack.

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?

A general recursive algorithm takes advantage of whenever there is a possibility of breaking down a problem into multiple smaller, but similarly structured problems, each if solved (and possibly in parallel if there is sufficient independence) can be "combined" to get the solution to the original problem. The generic recursive algorithm is:

Algorithm 2 Recursive Planner

```
1: function RECURSIVE-PLANNER($Prob_N$)                ▷ Solve a sub-problem
2:     if Some Terminal Condition then
3:         TERMINAL-OPERATION($Prob_1$)                ▷ Some constant terminal operation
4:     RECURSIVE-PLANNER($Subset(Prob_N)$)
```

There are a whole class of problems to which a recursive planner can be applied. For instance, this algorithm can be used in large scale map navigation problems, where loading an entire terrain in one go may be memory prohibitive.

There are two conditions for a planner to be complete,

1. **Produces a plan if there is one**: The recursive planner relies on all the sub-goals to complete their execution (or the recursion stack to be empty). So in case all the sub-goals reach their respective terminating condition, then all of them will reach completion (this is from an execution standpoint - different from the notion of completeness). If all the individual problems finish, then the top-level problem, which is basically a composition of these sub-problems will also produce a valid plan.

   However, it is possible that even though the entire problem has a solution, one of the sub-problems may never reach completion – this would cause the original problem to not produce a plan. Hence it cannot always be guaranteed that the planner produces a plan if there is a valid one.

2. **Reports that there is no plan if there is none**: The same sub-problem non-termination issue exists here as well. In case one of the sub-problems gets "stuck", there is no way for the original problem to terminate reporting that there is no plan.

However, if some sort of a terminating "*parameter*" can be used to force the sub-problems to halt/report no solution. In such a case the "stack overflow" issue will be resolved and both the conditions of completeness will be met. Basically, we need to ensure that the sub-problems will be forcibly terminated which will ensure that the main problem will always terminate as well!

References

[1] Henry Kautz and Bart Selman. Blackbox: A new approach to the application of theorem proving to problem solving. In *AIPS98 Workshop on Planning as Combinatorial Search*, volume 58260, pages 58–60, 1998.

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

[3] Henry Kautz, David McAllester, and Bart Selman. Encoding plans in propositional logic. *KR*, 96:374–384, 1996.