Homework 3 16782: Planning and Decision Making in Robotics

Harsh Dhruva
AndrewID: hdhruva
Symbolic Planning

November 14th, 2021

1. Implementation

Compile the code with either option:

- g++ planner.cpp -o planner.out
- g++ planner.cpp -o planner.out -Ofast

To run the code

- ./planner.out BlocksTriangle.txt

For other filenames, the latter argument will change accordingly. The compiling and running instructions are the same as shown in the assignment. For compiling with faster optimization, the -Ofast argument can be used. I will show results for both compilations.

I have provided comments in the code about the implementation and the data structures used. I have precalculated the permutations of the symbols based on the maximum number of symbols required for a particular action or the maximum number of arguments. The permutations are stored in a vector<list<list<string>>>, where each list<list<string>> corresponds the permutations for n number of variables. Then I calculate all the possible grounded actions that can be applied based on the permuted arguments to symbols mapping. All the grounded actions are then stored into a list, with their corresponding grounded preconditions and effects added to the GroundedActions data structure.

In the A-star search, I check through all the actions, and only apply the action that is valid in preconditions. Once the action is applied a new State is generated only if it already does not exist. Initially I was not checking is a State with certain set of grounded conditions existed in the expanded list, and this was causing the Fire Extinguisher environment to run indefinitely. Once the goal state is achieved, I backtrack using the parent State and the Grounded Action that resulted in that State.

2. Results

The list of actions for each environment are as follows:

Blocks
MoveToTable(A,B)
Move(C,Table,A)
Move(B,Table,C)

Blocks and Triangles
MoveToTable(T0,B0)
MoveToTable(T1,B3)
MoveToTable(B0,B1)
Move(B1,B4,B3)
Move(B0,Table,B1)
Move(T1,Table,B0)

Fire Extinguisher
MoveToLoc(A,B)
LandOnRob(B)
MoveTogether(B,W)
FillWater(Q)
MoveTogether(W,F)
TakeOffFromRob(F)
PourOnce(F)
LandOnRob(F)
MoveTogether(F,W)
FillWater(Q)
MoveTogether(W,F)
Charge(Q)
TakeOffFromRob(F)
PourTwice(F)
LandOnRob(F)
MoveTogether(F,W)
FillWater(Q)
MoveTogether(W,F)
Charge(Q)
TakeOffFromRob(F)
PourThrice(F)

Table 1: Results for Symbolic Environments

		Time Taken (s)	Time Taken with -Ofast (s)	No. of States Expanded
Blocks	Heuristic	0.002966	0.001102	6
	No Heuristic	0.007802	0.001905	13
Blocks and Triangles	Heuristic	0.528218	0.101104	68
	No Heuristic	50.8441	9.73872	740
Fire Extinguisher	Heuristic	4.39457	0.701031	339
	No Heuristic	5.06294	0.789837	365

3. Discussion

Heuristic Used:
The number of conditions in the goal state that do not exist in the and the current state. Here the edge costs are set to 1. Using this a regular A-star search is performed.

From Table 1, we see that the use of a heuristic clearly guides the search better than no use of a heuristic, even though there are several other heuristics that could be used. We see that there are a greater number of expanded states when no heuristic is used, which contributes to the increase in execution time. In the Blocks and Triangles, there is a significant increase in the number of states expanded, which also shows a significant increase in the execution time. Therefore, the use of a heuristic for the Blocks and Triangles environments shows a huge difference in performance.

For the Fire Extinguisher environment, the number of states expanded without the heuristic are more than with, however, not as significant as the Blocks and Triangles. And therefore, the difference in execution time is not as significant either. This is because the heuristic value for the Fire Extinguisher environment will always be 1, since there is one condition in the goal conditions. Therefore, the search with a simple heuristic is almost as informative as no heuristic.

The use of a heuristic improves performance in all the environments. Since this heuristic can be an overestimate or sometimes an underestimate of how the current state is different from the goal state, a heuristic calculated using a forward Dijkstra search would be more guided and expand far less states. However, the execution time could depend on the size of the environment, since a Dijkstra search is used for each expanded state to calculate the heuristic value. For environments that are very large, it would be useful to have a far more guided search, since a large number of state expansions will affect the execution time more than it takes for calculating the Dijkstra heuristic. For smaller environments, it makes sense to use a simple heuristic, since the number of states expanded will not affect the execution time as much as running multiple Dijkstra searches.