Project 3 – Implement a Planning Search

Heuristics Analysis

Introduction to Part 3 (Written Analysis)

The purpose of this project is to compare various planning search algorithms using three different problems from the Air Cargo domain presented in the lectures and text (Russell & Norvig, 2010).

1 Provide an optimal plan for Problems 1, 2, and 3.

2 Compare and contrast non-heuristic search result metrics (optimality, time elapsed, number of node expansions) for Problems 1,2, and 3. Include breadth-first, depth-first, and at least one other uninformed non-heuristic search in your comparison; Your third choice of non-heuristic search may be skipped for Problem 3 if it takes longer than 10 minutes to run, but a note in this case should be included.

3 Compare and contrast heuristic search result metrics using A* with the "ignore preconditions" and "level-sum" heuristics for Problems 1, 2, and 3.

4 What was the best heuristic used in these problems? Was it better than non-heuristic search planning methods for all problems? Why or why not?

5 Provide tables or other visual aids as needed for clarity in your discussion

1 Optimal Plans

Problem 1 – 6 Steps

```
Load(C1, P1, SFO)
Load(C2, P2, JFK)
Fly(P2, JFK, SFO)
Unload(C2, P2, SFO)
Fly(P1, SFO, JFK)
Unload(C1, P1, JFK)
```

Problem 2 – 9 Steps

```
Load(C1, P1, SFO)
Load(C2, P2, JFK)
Load(C3, P3, ATL)
Fly(P2, JFK, SFO)
Unload(C2, P2, SFO)
Fly(P1, SFO, JFK)
Unload(C1, P1, JFK)
Fly(P3, ATL, SFO)
Unload(C3, P3, SFO)
```

Problem 3 – 12 Steps

```
Load(C1, P1, SFO)
Load(C2, P2, JFK)
Fly(P2, JFK, ORD)
Load(C4, P2, ORD)
Fly(P1, SFO, ATL)
Load(C3, P1, ATL)
Fly(P1, ATL, JFK)
Unload(C1, P1, JFK)
Unload(C3, P1, JFK)
Fly(P2, ORD, SFO)
Unload(C2, P2, SFO)
Unload(C4, P2, SFO)
```

Thomas Dang, 2017

2 Uninformed Search

Problem	Search (Heuristic)	Plan length	Expansions	Time (sec)	Time (min)
1	**breadth_first_search**	**6**	**43**	**0,0355**	**0,0006**
1	**depth_first_graph_search**	**20**	**21**	**0,0173**	**0,0003**
1	**uniform_cost_search**	**6**	**55**	**0,0425**	**0,0007**
1	**greedy_best_first_graph_search**	**6**	**7**	**0,0066**	**0,0001**
1	**A* (h=1 (null heuristic))**	**6**	**55**	**0,0417**	**0,0007**
1	**A* (ignore preconditions)**	**6**	**41**	**0,0312**	**0,0005**
1	**A* (h_pg_levelsum)**	**6**	**11**	**0,8007**	**0,0133**
1	breadth_first_tree_search	6	1458	1,0275	0,0171
1	depth_limited_search	50	101	0,0994	0,0017
1	recursive_best_first_search h_1	6	4229	2,9658	0,0494
2	**breadth_first_search**	**9**	**3343**	**14,7553**	**0,2459**
2	**depth_first_graph_search**	**619**	**624**	**3,8502**	**0,0642**
2	**uniform_cost_search**	**9**	**4853**	**13,4739**	**0,2246**
2	**greedy_best_first_graph_search**	**21**	**998**	**2,7511**	**0,0459**
2	**A* (h=1 (null heuristic))**	**9**	**4853**	**13,8061**	**0,2301**
2	**A* (ignore preconditions)**	**9**	**1450**	**4,3801**	**0,0730**
2	**A* (h_pg_levelsum)**	**9**	**86**	**74,8395**	**1,2473**
2	breadth_first_tree_search	-	-	-	-
2	depth_limited_search	50	222719	1074,5463	17,9091
2	recursive_best_first_search h_1	-	-	-	-
3	**breadth_first_search**	**12**	**14663**	**125,7773**	**2,0963**
3	**depth_first_graph_search**	**392**	**408**	**2,0446**	**0,0341**
3	**uniform_cost_search**	**12**	**18235**	**61,5990**	**1,0267**
3	**greedy_best_first_graph_search**	**22**	**5614**	**20,0959**	**0,3349**
3	**A* (h=1 (null heuristic))**	**12**	**18235**	**67,3210**	**1,1220**
3	**A* (ignore preconditions)**	**12**	**5040**	**19,8719**	**0,3312**
3	**A* (h_pg_levelsum)**	**12**	**315**	**404,1656**	**6,7361**
3	breadth_first_tree_search	-	-	-	-
3	depth_limited_search	-	-	-	-
3	recursive_best_first_search h_1	-	-	-	-

The table above shows all the results for every problem stated in the project. Searches were performed using the run_search.py utility tool. The number of actions required to find the goal as determined are represented by the plan length. The number of frontier nodes, which were expanded in order to find the solution, is represented by the extensions. Higher numbers show higher complexity of the searches. Heuristics coded for the project are only included in A* searches.

Uninformed searches are those which are not marked as A* and have no additional heuristics to support them. *Breadth-first, depth-first, uniform-cost* and *greedy-best-first* were run for every problem,

whereas *breadth-first-tree, recursive-best-first* were only run for problem 1 and *depth-limited* was only run for problem 1 and 2.

Optimality vs. efficiency

Breadth-first-tree, recursive-best-first and *depth-limited* were not accounted for further investigation because of their bad performance regarding plan length, time and expansions during the first few calculations.

Looking at problem 1 results only, the best algorithm seems to be *greedy-best-first*. But as soon as we include the results of problem 2 and 3 obviously the *greedy-best-first* algorithm is not reliable enough anymore to solve the more complex problems. Instead it will result in more actions necessary to find the goal, represented by the plan length.

Depth-first performed time-efficient searches, but didn't result in optimal plans.

Even though *breadth-first* and *uniform-cost* both don't perform very efficiently regarding time and expanded nodes necessary, they are still considered the best choices within the range of uninformed searches. At least they were able to identify an optimal plan for the problems given. The lack of time-efficiency may be considered as less important than finding an optimal plan.

3 Automatic Heuristics

Two automatic heuristics were implemented in the code, *ignore-preconditions* and *level-sum*.

Ignore-preconditions contains the number of goal requirements for the problem still not met at any given node in the planning search. This heuristic is based on the assumption of sub-goal independence.

Level-sum contains the sum of the level costs found for each of the goal requirements. The level cost for any given goal requirement is the level number where it first appears, where S levels are literal levels and A levels are action levels. This heuristic is also based on the assumption of sub-goal independence.

Regarding A* search calculating the heuristics directly affects the efficiency of the search. If calculating the heuristic itself takes a lot of resources regarding time and memory space the search loses its efficiency. However, a simple heuristic can perform quite well in helping A* search to lower the number of extensions. Therefore it will also improve the efficiency within the planning search.

Among the given heuristics the *ignore-preconditions* heuristic is the simplest. It doesn't take into account that specific actions or states are required to reach a goal. For Example, a plane must be loaded and flown from A to B, and then unloaded at the desired location in order to meet the requirement of moving cargo from one airport to another. Using *level-sum* will lead to calculations with a closer look into the number of steps needed to the goal, thus it will result in a longer calculation time. However, it can still be a useful heuristic.

4 Informed Search

Among informed search heuristics the *ignore-preconditions* heuristic performed the fastest A* searches for all of the given problems. However, the *level-sum* heuristic showed the fewest node expansions.

The Winner

Overall, the best heuristic found among the performed searches was the *ignore-preconditions* heuristic. It may not result in the fewest expansions, but its speed makes it the best choice in average. Regarding speed it also outperformed both uninformed planning searches with an optimal plan length, the *breadth-first* and the *uniform-cost*. The *ignore-preconditions* heuristic balances usability for the A* search and speed of calculation for the heuristic, which results in reduced expansions compared to uninformed searches.