Heuristics Analysis

Lucas Sabbatini de Barros Fonseca

Artificial Intelligence Nanodegree

Domain-independent Planner Project

1. Overview

The course of development of this programming project is twofold. First, a regular search will be performed using action schemas with preconditions to be met so to sample possible actions at each state. This means that states are considered atomic, and actions take from one state to another, and the objective is to find a sequential set of actions that leads from initial state to goal state. The terminal tests checks if every condition (literal) for the goal state has been met at a given state.

In part two, a GRAPHPLAN algorithm will be used to create domain-independent heuristics for states in an informed search, so to compare this approach to other heuristics and uninformed searches.

The main objective is to compare the performance of regular search algorithms, with determined heuristics, along with one using the GRAPHPLAN to generate them, through data gathered with simulations.

2. GRAPHPLAN algorithm

Planning Domain Description Language (PDDL) is a representation language for logical statements that makes it possible to characterize a state of a problem in a factored manner while, which then allows for an originally exponential search problem to be approximated polynomially with the GRAPHPLAN algorithm. [1]

Using a factored description with literals for states and action schemas to represent actions in a propositional form (instantiating variables with actual values) a search through the literal levels space may be attained by using actions preconditions to determine which action may be taken at each level. Also, relaxation of action's preconditions may yield good heuristics in a GRAPHPLAN algorithm. [1]

The GRAPHPLAN algorithm starts out with an initial state level, which contains all positive literals from the starting state, along with relevant negative ones, and alternates between state and action levels.. The search does not expand states through actions at each state level, but opens up literas that could arise from taking every possible action from the previous action level.

This means that the number of literals being expanded is constant in the worst case, and the space and time complexities are functions of the number of literals in the problem and the maximum mutual exclusion constraints which would be $O(n(a + l)^2)$. [1]

3. Data

All data shown below was gathered using the run_search.py script from the Udacity's project repository. Runs that lasted more than ten minutes have "+10 min" in their Time Elapsed column.

ACP 1	Plan Length	Expansions	Goal Tests	New Nodes	Time Elapsed (sec)
Breadth-First	6	43	56	180	0.022
Breadth-First tree	6	1458	1459	5960	0.584
Depth-First graph	20	21	22	84	0.012
Depth Limited	50	101	271	414	0.062
A*+h1	6	55	57	224	0.024
A*+h_ignore	6	41	43	170	0.020
A*+h_level_sum	6	11	13	50	0.722

ACP 2	Plan Length	Expansions	Goal Tests	New Nodes	Time Elapsed (sec)
Breadth-First	9	3343	4609	30509	5.770
Breadth-First tree	-	-	-	-	+10 min
Depth-First graph	619	624	625	5602	2.913
Depth Limited	50	222719	2053741	2054119	680.6312
A*+h1	9	4853	4855	44041	7.823
A*+h_ignore	9	1450	1452	13303	3.036
A*+h_level_sum	9	86	88	841	65.355

ACP 3	Plan Length	Expansions	Goal Tests	New Nodes	Time Elapsed (sec)
Breadth-First	12	14663	18098	129631	28.1900
Breadth-First tree	-	-	-	-	+10 min
Depth-First graph	392	408	409	3364	1.3285
Depth Limited	-	-	-	-	+10 min
A*+h1	12	18223	18225	159618	36.6116
A*+h_ignore	12	5040	5042	44944	13.510
A*+h_level_sum	12	318	320	2934	340.219

4. Analysis

From the data it is observed that increasing size of the search problem strongly correlates to the search space and time complexities, as expected. As Peter Norvig demonstrates in the video “Quiz: State Spaces 2” from lesson 11: Search in the AI Nanodegree, the state space size is exponential on the number of variables to be described in a state.

The h1 heuristic, which return a constant value of 1 showed to perform worse than a simple breadth first search, and it was also expected since it returns no information about the goal. The h_ignore_preconditions heuristic yielded better performance than most uninformed searches, except depth-first-search, which was always the quickest one between all of them, though solutions were the longest ones (not optimal).

h_level_sum using the GRAPHPLAN algorithm decreased the size of the search space by a large amount for every problem, but the computation requirements for creating the graph, along with counting the number of steps for each individual goal made the heuristic worse in time than every other, though it did find optimal solution.

Optimality was true for every A* with heuristic and breadth-first search. h_ignore_preconditions had the best average performance in time, and given problems of these sizes, it is the recommended one, even though it is way more expensive in space than the h_level_sum.

5. Optimal sequence of actions for problems

For each of the three problems an optimal plan has been found, as presented below.

Problem 1: optimal path length = 6

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

Problem 2: optimal path length = 9

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

Problem 3: optimal path length = 12

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

6. References

[1] Norvig, P. and Russell, S. (2009). Artificial Intelligence: a Modern Approach. pp. 366 - 388.