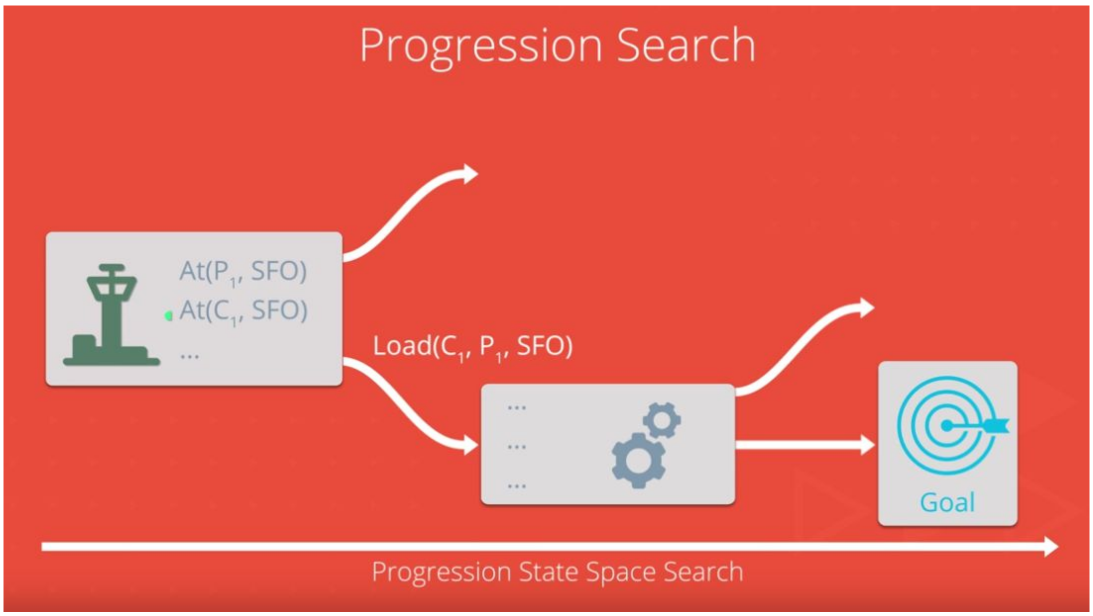
Artificial Intelligence Nanodegree Program

Manceñido Agustín

Build a Forward-Planning Agent



Introduction

Planning is an important topic in AI because intelligent agents are expected to automatically plan their own actions in uncertain domains. This project is split between implementation and analysis. First I will combine symbolic logic and classical search to implement an agent that performs progression search to solve planning problems. Then I will experiment with different search algorithms and heuristics, and use the results to answer questions about designing planning systems.

Implementation

Planning Graph & Heuristic Implementation

For this project, a progression search agent was developed to solve planning problems. The project required the implementation of support functions and a progression search experiment to evaluate the performance for different algorithms. Code passes all Project Assistant test cases for:

- `ActionLayer` mutual exclusion rules:
  - `_inconsistent_effects()`
  - `_interference()`
  - `_competing_needs()`
- `LiteralLayer` mutual exclusion rules:
  - `_inconsistent_support()`
  - `_negation()`
- `PlanningGraph` class heuristics:
  - `h_levelsum()`
  - `h_maxlevel()`
  - `h_setlevel()`

```
# python -m unittest
     Ran 35 tests in 17.571s
     OK
```

Experimental Results & Report

The four available problems were experimented with the 11 search algorithms to understand the tradeoffs as problem size increases. The results can be summarized in the next Table:

Problem 1 (20 Actions)						Problem 2 (72 Actions)					
Search Function	Plan Lth	Time (s)	Nodes	Goal Tests	Optimal?	Search Function	Plan Lth	Time (s)	Nodes	Goal Tests	Optimal?
BFS	6	0.0072	178	56	Yes	BFS	9	2.0747	30503	4609	Yes
DFS	20	0.0072	84	22	No	DFS	619	3.14	5602	625	No
UCS	6	0.0098	240	62	Yes	UCS	9	3.5679	46618	5156	Yes
GB BFS unmet	6	0.0017	29	9	Yes	GB BFS unmet	9	0.0216	170	19	Yes
GB BFS levelsum	6	0.1948	28	8	Yes	GB BFS levelsum	9	3.9847	86	11	Yes
GB BFS maxlevel	6	0.1384	24	8	Yes	GB BFS maxlevel	9	6.3775	249	29	Yes
GB BFS setlevel	6	0.6722	28	8	Yes	GB BFS setlevel	9	16.5709	84	11	Yes
A* unmet	6	0.0099	206	52	Yes	A* unmet	9	2.2515	22522	2469	Yes
A* levelsum	6	0.4676	122	30	Yes	A* levelsum	9	105.7746	3426	359	Yes
A* maxlevel	6	0.5073	180	45	Yes	A* maxlevel	9	619.8675	26594	2889	Yes
A* setlevel	6	1.5122	138	35	Yes	A* setlevel	9	1489.1898	9605	1039	Yes
Problem 3 (88 Actions)						**Problem 4 (104 Actions)**					
Search Function	**Plan Lth**	**Time (s)**	**Nodes**	**Goal Tests**	**Optimal?**	**Search Function**	**Plan Lth**	**Time (s)**	**Nodes**	**Goal Tests**	**Optimal?**
BFS	12	10.9831	129625	18098	Yes	BFS	14	95.7858	944130	114953	Yes
DFS	392	5.239	3364	409	No	DFS	24132	16441.602	228849	25175	No
UCS	12	14.9696	161936	18512	Yes	UCS	14	731.2309	1066413	113341	Yes
GB BFS unmet	15	0.0367	230	27	No	GB BFS unmet	18	0.2017	280	31	No
GB BFS levelsum	14	9.3965	126	16	No	GB BFS levelsum	17	129.6636	165	19	No
GB BFS maxlevel	13	9.2386	195	23	No	GB BFS maxlevel	17	269.4391	580	58	No
GB BFS setlevel	17	92.4436	345	37	No	GB BFS setlevel	23	2586.1117	1164	109	No
A* unmet_goals	12	8.5154	65711	7390	Yes	A* unmet	14	522.5451	328509	34332	Yes
A* levelsum	12	195.6628	3403	371	Yes	A* levelsum	15	5897.6493	12210	1210	No
A* maxlevel	12	3573.1945	86312	9582	Yes						

Search complexity analysis

From the data collected above it is possible to analyze the search complexity as a function of domain size, search algorithm, and heuristic. The memory required by the search can be analyzed by number of nodes expanded by the algorithm Figure 1 and Figure 2 represents the number of nodes expanded for each problem and search algorithm (note that the scale is logarithmic).

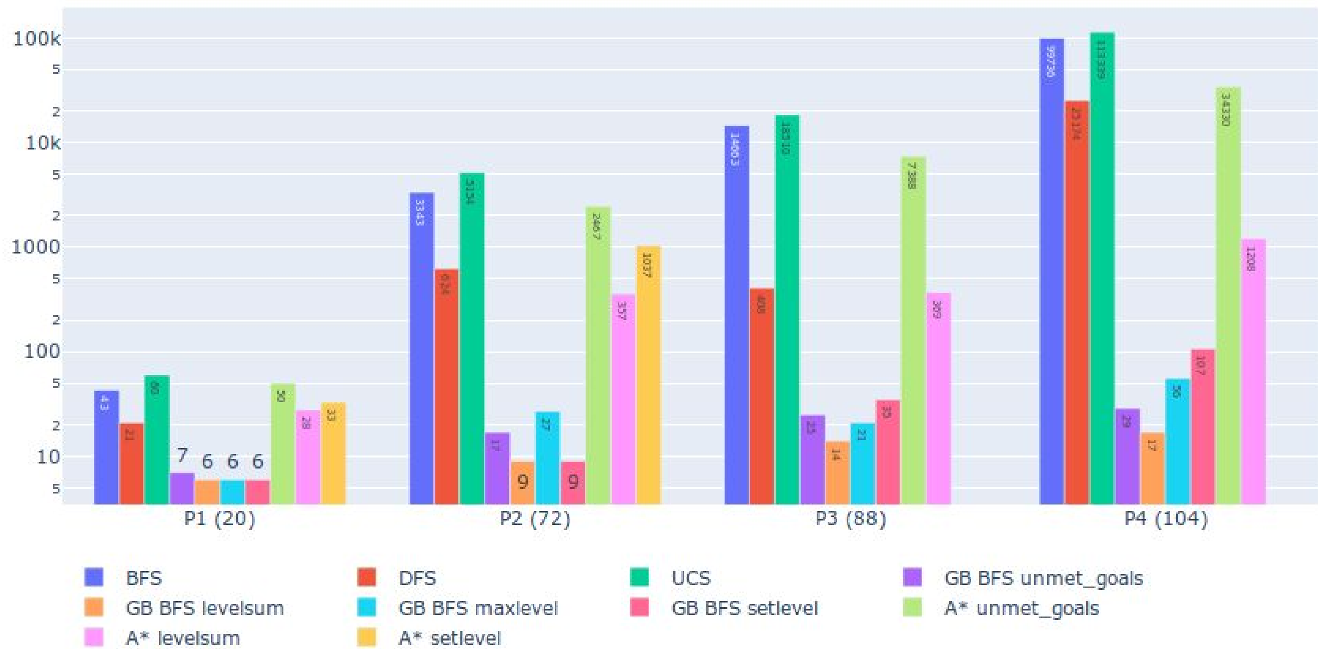


Figure 1 - Number of nodes expanded for each problem [log]

We can notice form Figure 1 that the shape of each group of bars is similar for each problem, given that the number of nodes expanded is directly related with the number of actions.

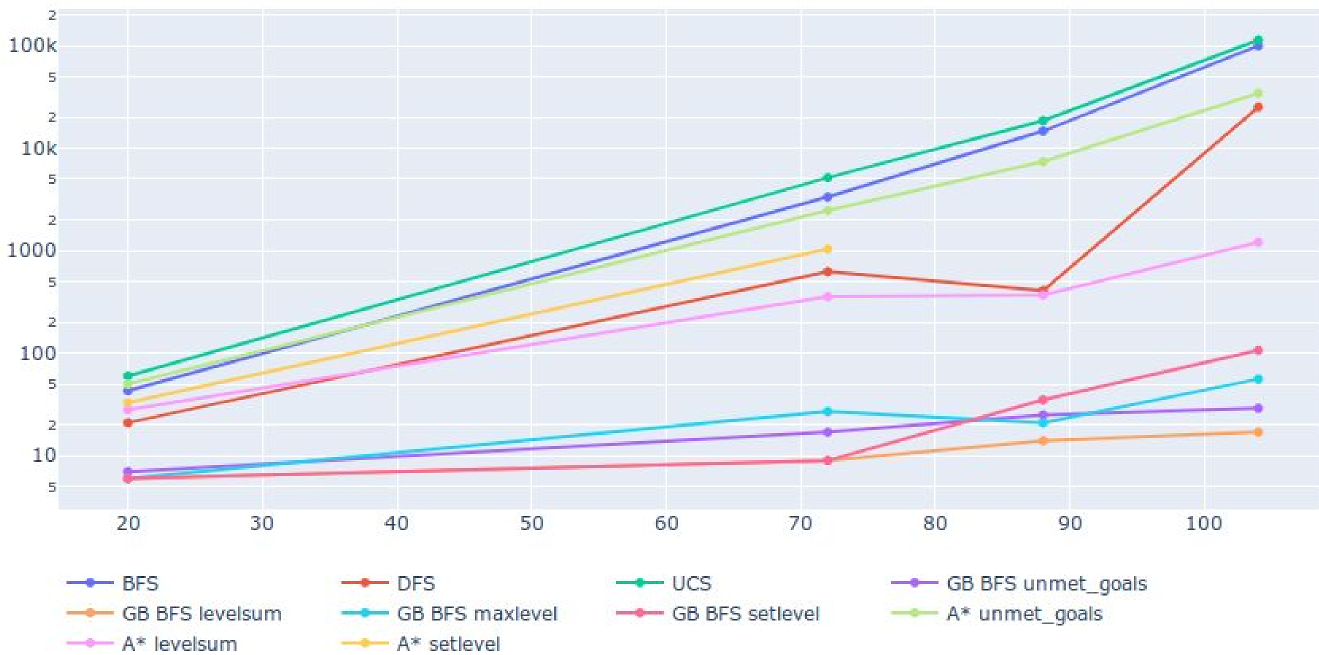


Figure 2 - Number of nodes expanded against number of actions [log]

From the analysis of the figures we can see that the 4 `greedy_best_first_graph_search` algorithms (GB BFS) results with the less number of nodes expanded followed by the DFS, BFS and UCS search algorithms.

Search time analysis

The execution time analysis for each of the problems and type of search has been represented in Figure 3 and Figure 4.

Figure 3 - Time elapsed (s) for each problem [log]

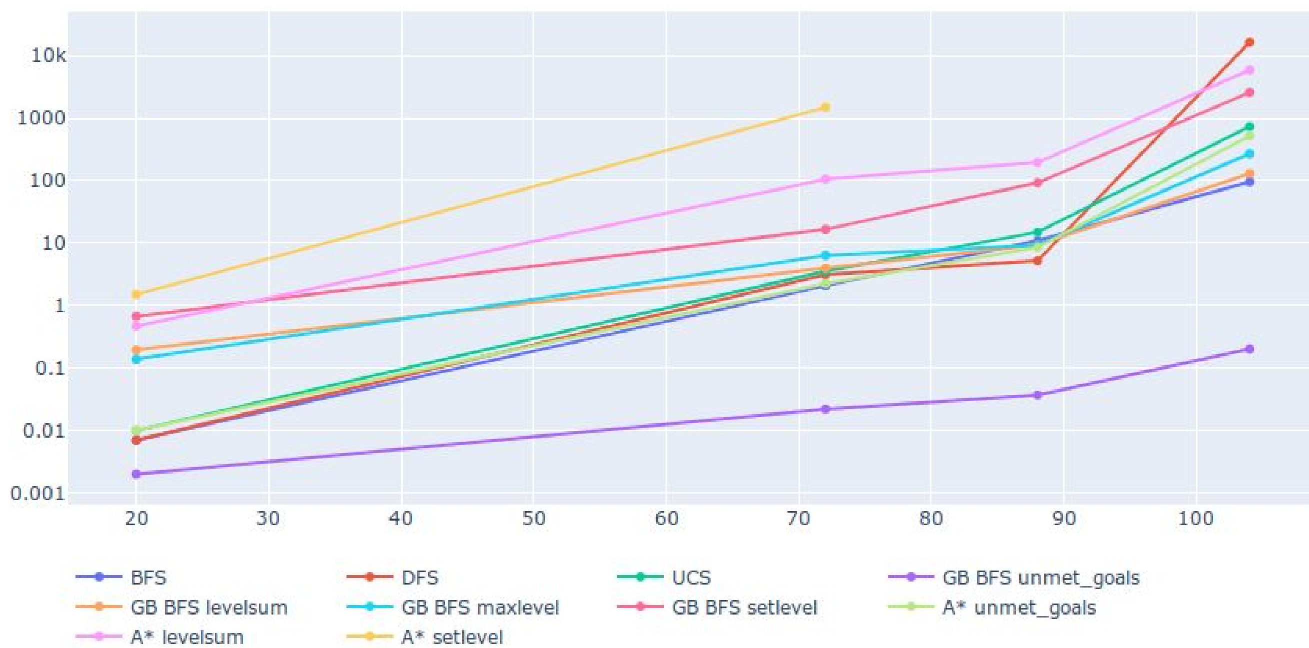


Figure 4 - Time elapsed (s) against number of actions [log]

We can see from the figures that there is a correlation between the number of nodes expanded and the time execution.

It can be seen from the analysis that time taken to reach goal state by `greedy_best_first_graph_search unmet_goals` (GB BFS unmet_goals) is considerably low compared with other algorithms (note that the scale is logarithmic) even for different number of actions.

Optimality of solution

BFS, UCS and A* unmet_goals provides optimal solution for each of the problems with path length of 6, 9, 12 and 14 for P1, P2, P3 and P4 respectively. The path length of DFS is quite high as compared to the optimal solution but the other search algorithms were close the the optimal value. Figure 5 shows the number of nodes expanded and the execution times for the optima solutions:

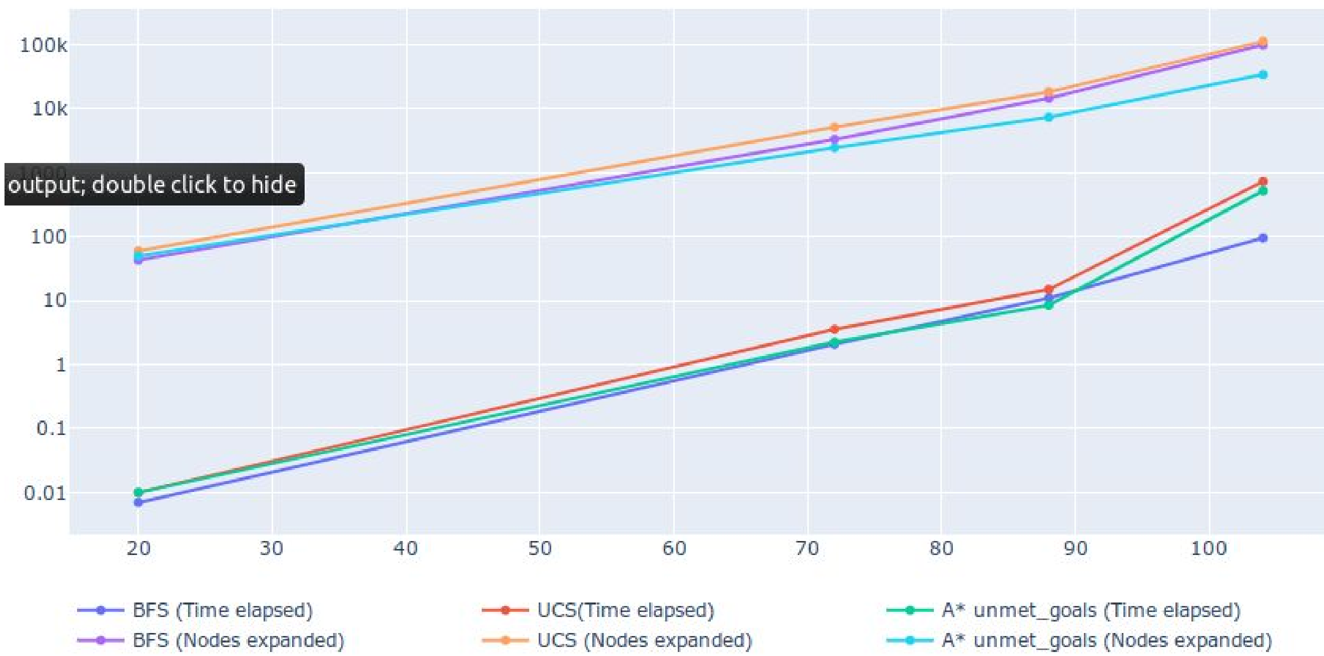


Figure 5 - Time elapsed (s) and Nodes expanded for optimal solutions [log]

Questions

Which algorithm or algorithms would be most appropriate for planning in a very restricted domain (i.e., one that has only a few actions) and needs to operate in real time?

For this case we will choose the algorithm with lower response times to operate in real time. From the search algorithms analyzed we would choose `greedy_best_first_graph_search unmet_goals` (GB BFS unmet_goals) but BFS and UFC are good choices too.

Which algorithm or algorithms would be most appropriate for planning in very large domains (e.g., planning delivery routes for all UPS drivers in the U.S. on a given day)

As observed in the previous analysis Greedy searches provide the best trade-off between time, nodes expanded and plan length with solutions close to the optimal. Breadth-first search and Uniform Cost search are also good alternatives with optimal solutions.

Which algorithm or algorithms would be most appropriate for planning problems where it is important to find only optimal plans?

Breadth-First Search and Uniform Cost Search are most appropriate for planning problems requiring an optimal plan as they are guaranteed to find an optimal solution.