AIND-Planning Heuristic Analysis

Goals

1. Implement Planning problems TODO in `my_air_cargo_problems.py`
2. Implement Domain independent heuristics TODO in `my_air_cargo_problems.py`
3. Implement Planning Graph and automatic heuristic in `my_planning_graph.py`
4. Written analysis of the search solutions (this)

Problem definition

The problem used for this project is in the Air Cargo domain. The planning language used is the PDDL (Planning Domain Definition Language). All the problems have the same action schema, but different initial states and goals.

Action Schema

```
Action(Load(c, p, a),
    PRECOND: At(c, a) ∧ At(p, a) ∧ Cargo(c)   ∧ Plane(p) ∧ Airport(a)
    EFFECT: ¬ At(c, a) ∧ In(c, p))
Action(Unload(c, p, a),
    PRECOND: In(c, p) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)
    EFFECT: At(c, a) ∧ ¬ In(c, p))
Action(Fly(p, from, to),
    PRECOND: At(p, from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)
    EFFECT: ¬ At(p, from) ∧ At(p, to))
```

Problem 1

```
Init(At(C1, SFO) ∧ At(C2, JFK)
    ∧ At(P1, SFO) ∧ At(P2, JFK)
    ∧ Cargo(C1) ∧ Cargo(C2)
    ∧ Plane(P1) ∧ Plane(P2)
    ∧ Airport(JFK) ∧ Airport(SFO))
Goal(At(C1, JFK) ∧ At(C2, SFO))
```

Problem 2

```
Init(At(C1, SFO) ∧ At(C2, JFK) ∧ At(C3, ATL)
    ∧ At(P1, SFO) ∧ At(P2, JFK) ∧ At(P3, ATL)
    ∧ Cargo(C1) ∧ Cargo(C2) ∧ Cargo(C3)
    ∧ Plane(P1) ∧ Plane(P2) ∧ Plane(P3)
    ∧ Airport(JFK) ∧ Airport(SFO) ∧ Airport(ATL))
Goal(At(C1, JFK) ∧ At(C2, SFO) ∧ At(C3, SFO))
```

Problem 3

```
Init(At(C1, SFO) ∧ At(C2, JFK) ∧ At(C3, ATL) ∧ At(C4, ORD)
    ∧ At(P1, SFO) ∧ At(P2, JFK)
    ∧ Cargo(C1) ∧ Cargo(C2) ∧ Cargo(C3) ∧ Cargo(C4)
    ∧ Plane(P1) ∧ Plane(P2)
    ∧ Airport(JFK) ∧ Airport(SFO) ∧ Airport(ATL) ∧ Airport(ORD))
Goal(At(C1, JFK) ∧ At(C3, JFK) ∧ At(C2, SFO) ∧ At(C4, SFO))
```

Search methods

These are the different search methods used to find a solution to the problems. The optimal solution should be recorded and searches should be analysed against various metrics. All searches are timed out after 600s.

num	search	heuristics	notes
1	Breadth First	-	Graph search with FIFO-Q as frontier and goal test applied when node is generated and before adding to frontier
2	Breadth First Tree	-	Tree search with FIFO-Q as frontier and goal test applied when node is pop'd out of frontier and before expansion
3	Depth First Graph	-	Graph search with Stack as frontier and goal test applied when node is pop'd out of frontier and before expansion
4	Depth Limited	-	Tree search with Stack as frontier and goal test applied when node is generated
5	Uniform Cost	-	Best First Search with f() = g() (= node.pathcost)
	Recursive		Similar to DFS. Depth is bounded by f_limit generated per parent based on f().f() = g() + h().

6	BestFirst	h_1	g() is path cost. In this case the heuristic, h() is constant heuristic
7	Greedy BestFirstGraph	h_1	Best first Search with f() = h(),Heuristic function. f() is used to order the PrioQ. In this case heuristic function is a contant. Similar to Graph Search except: uses PrioQ(g()), if child node already in frontier then least cost node is selected among them to remain in frontier
8	A*	h_1	Best first Search with f() = g() + h(). In this case h() is a constant.
9	A*	h_ignore_preconditions	Best first Search with f() = g() + h(). In this case heuristic, h() estimates the minimum no of actions to satisfy all goal conditions
10	A*	h_pg_levelsum	Best first Search with f() = g() + h(). In this case heuristic, h() is the planning graph levelsum. It is the sum of levelcost of all the goals in the planning graph

Optimal Solutions

One of the optimal solutions for each of the problem statements were found by running the different search methods on the state space.

Problem 1

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

Problem 2

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

Problem 3

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

Solution Analysis

Analysis data

problem	search	expansions	tests	new nodes	plan length	time
1	1	43	56	180	6	0.024007190004340373
1	2	1458	1459	5960	6	0.6136128880025353
1	3	12	13	48	12	0.005626584010315128
1	4	101	271	414	50	0.07546712399926037
1	5	55	57	224	6	0.026934384994092397
1	6	4229	4230	17029	6	1.8555523059912957

1	7	7	9	28	6	0.004607081005815417
1	8	55	57	224	6	0.027083968991064467
1	9	41	43	170	6	0.03590689900738653
1	10	11	13	50	6	1.5813310770026874
2	1	3343	4609	30509	9	9.868373634002637
2	2	0	0	0	0	0.0
2	3	1669	1670	14863	1444	11.395577344999765
2	4	0	0	0	0	0.0
2	5	4852	4854	44030	9	37.89966792600171
2	6	0	0	0	0	0.0
2	7	990	992	8910	15	7.244699397997465
2	8	4852	4854	44030	9	39.85160706299939
2	9	1506	1508	13820	9	12.620577531008166
2	10	86	88	841	9	161.76354925899068
3	1	14663	18098	129631	12	81.48128371099301
3	2	0	0	0	0	0.0
3	3	592	593	4927	571	2.187321028992301
3	4	0	0	0	0	0.0
3	5	18235	18237	159716	12	355.8362283030001
3	6	0	0	0	0	0.0
3	7	5614	5616	49429	22	89.38294529699488
3	8	18235	18237	159716	12	352.1639006740006
3	9	5118	5120	45650	12	79.5783260629978
3	10	0	0	0	0	0.0

Analysis Graphs

Fig 1. Plan len (log scale)

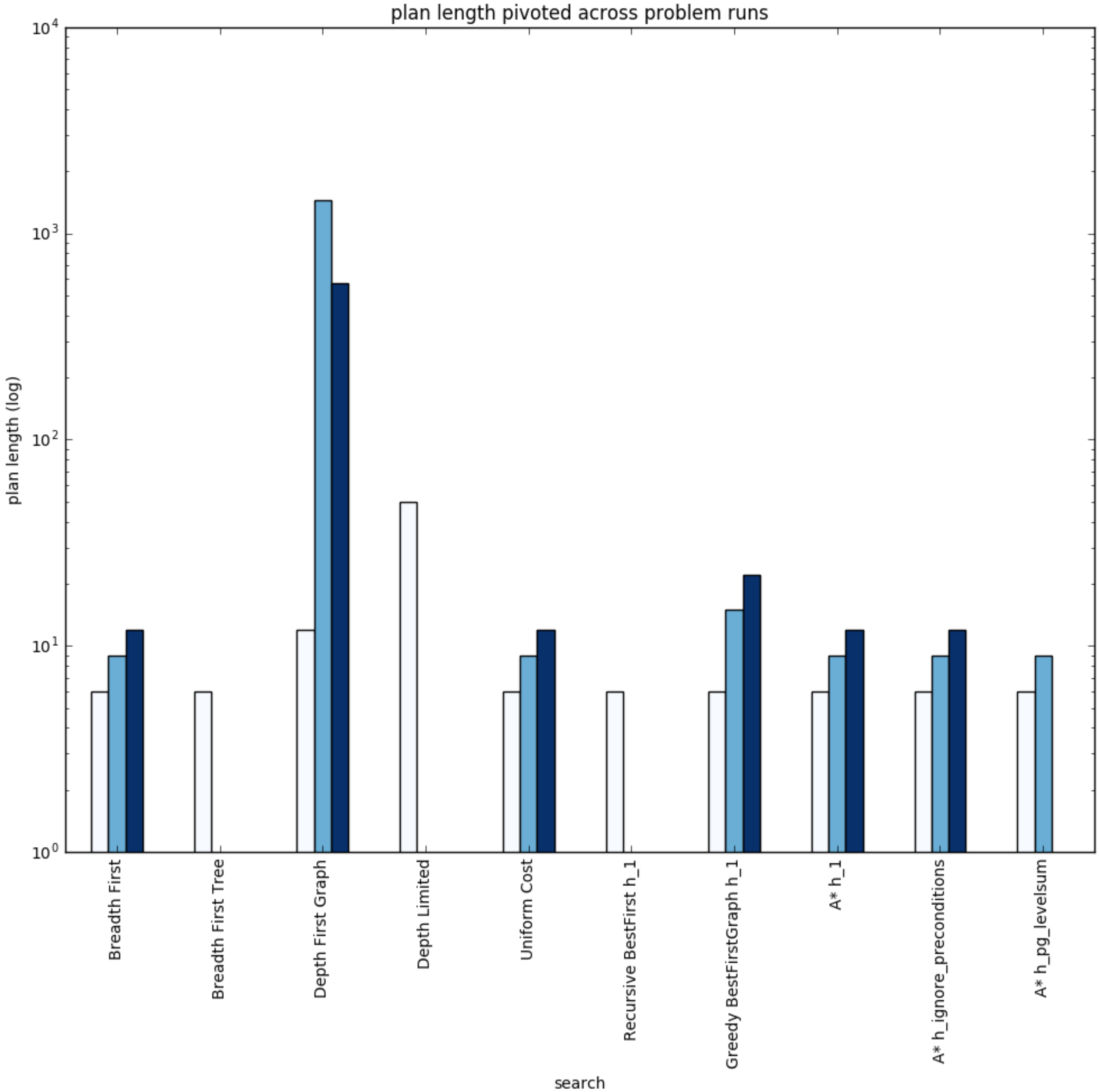


Fig 2. Time (log scale)

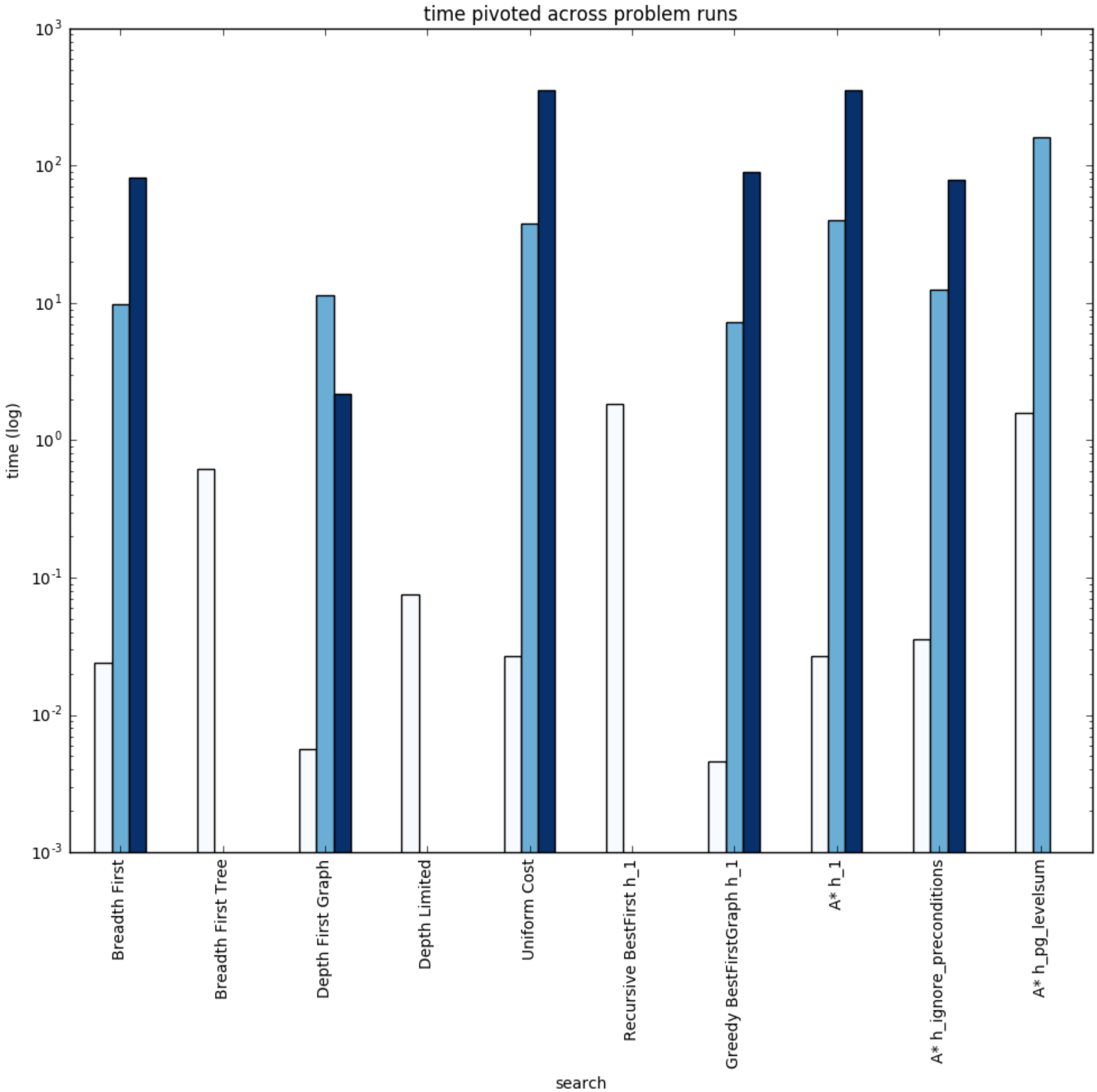
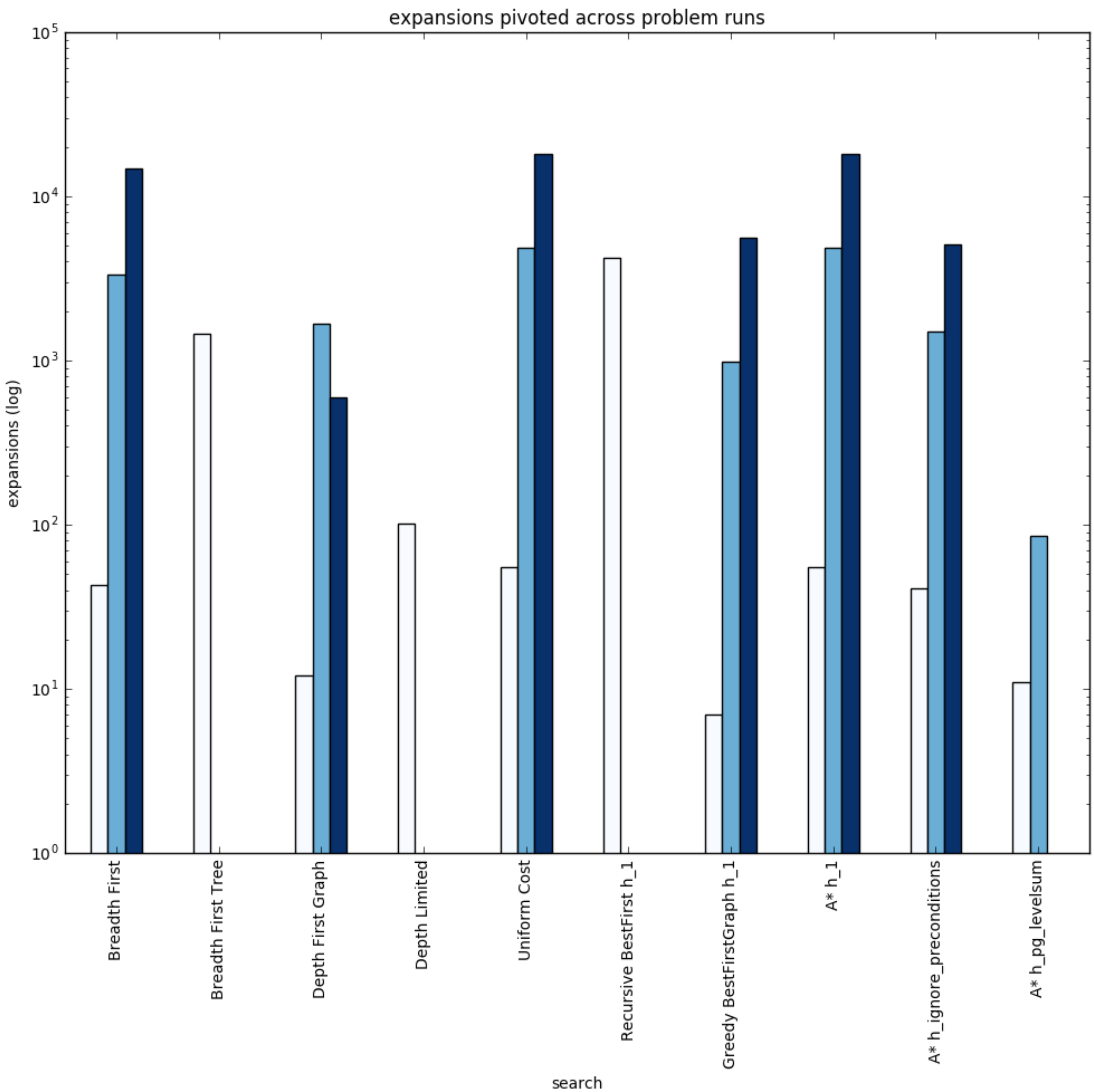


Fig 3. Expansions (log scale)



Non-Heuristic search analysis

Expansions is an indication of the space used by the search.

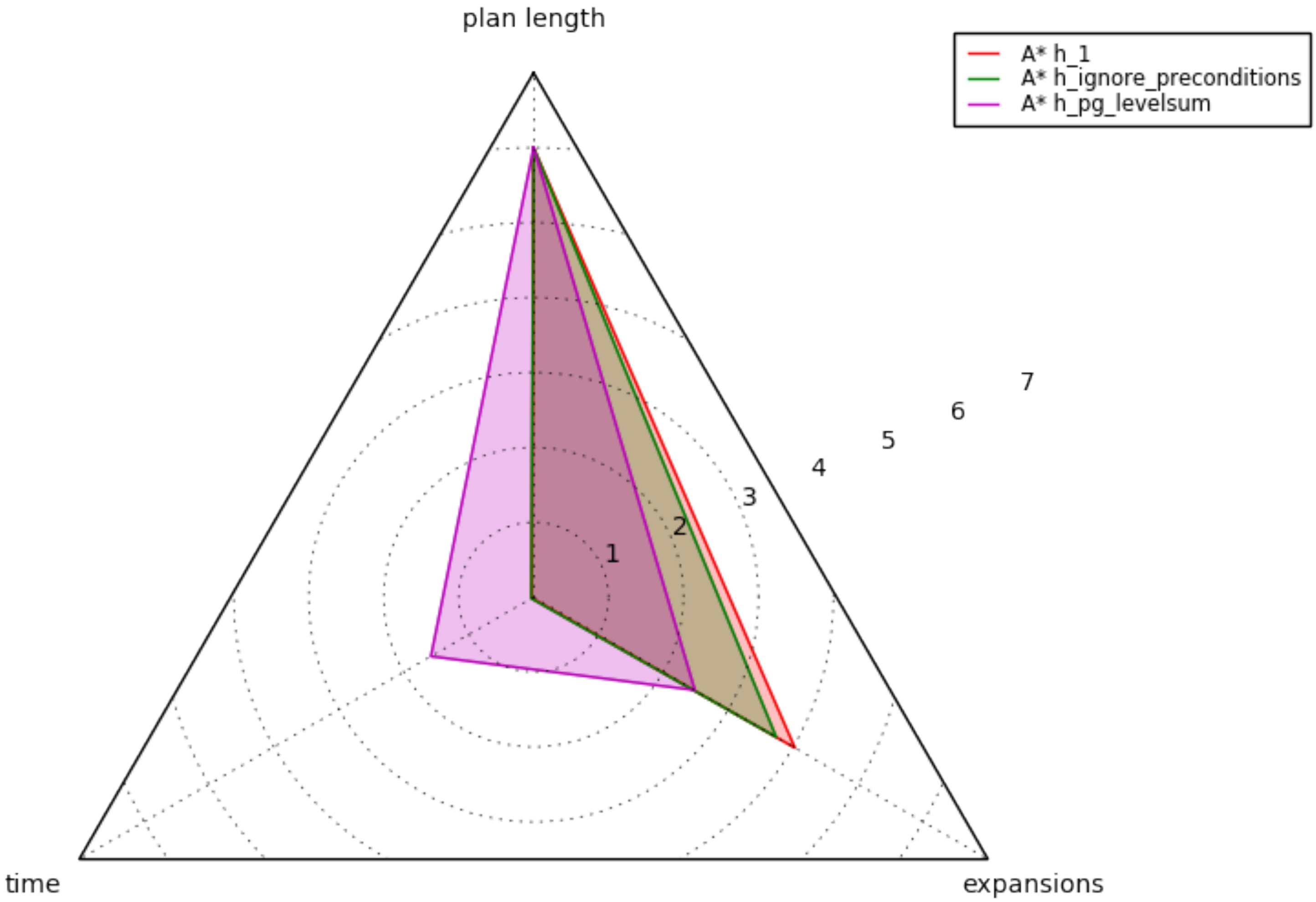
search	notes	pro	con
Breadth First (graph)	being optimal search a solution is always found. This search used more space than other searches. The time complexity is O(b^(d)) compared to O(b^(d+1)) for regular graph search.	optimal search	more space used
Breadth First Tree	Since a tree expansions was used, potential for loops exist. It took longer time and much larger space compared to Breadth First graph search.	Optimal search	Larger space and time. Timedout for 2 of the 3 problems.
Depth First	used fewer expansions and time but resulted in sub-optimal solutions. The solution in some cases had an order of magnitude greater number of steps compared to other searches.	less time and space	sub optimal results
Depth Limited	With depth_limit 50. This search returned the longest solution for the only problem it solved. Timed out for others.	None	Timed out for problem 2 and 3. More expansions than BFS graph
Uniform Cost	Since node costs were all equal(to 0), the performance was similar to BFS graph. Optimal solution was found. But used slightly more memory and time compared to BFS graph as the goal checks are applied only when nodes are pop'd from the fontier. The time complexity is O(b^(d+1))	Optimal search	More space used

A* heuristics analysis

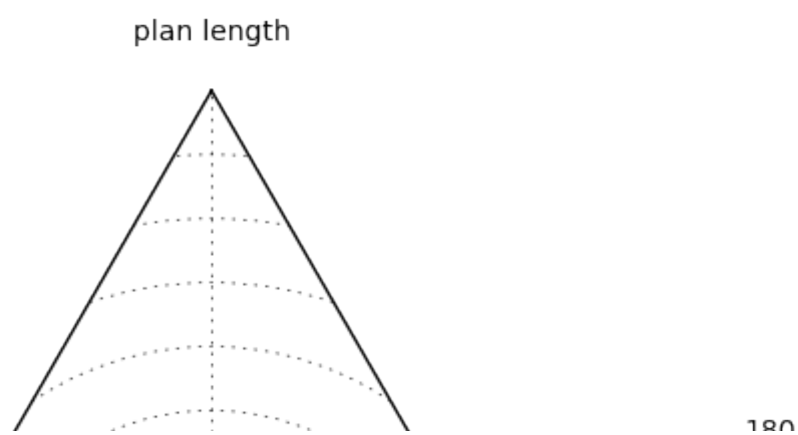
A with ignore_precondition* took less time but used more expansions to find optimal solution for all the problems. *A* with pg_levelsum* took longer time but expanded less nodes to find optimal solution for the first 2 problems. For the 3rd problem, it timed out (600s). Recommended for this problem is **ignore_precondition** heuristic since it finishes faster and provides an optimal solution in a comparable space usage.

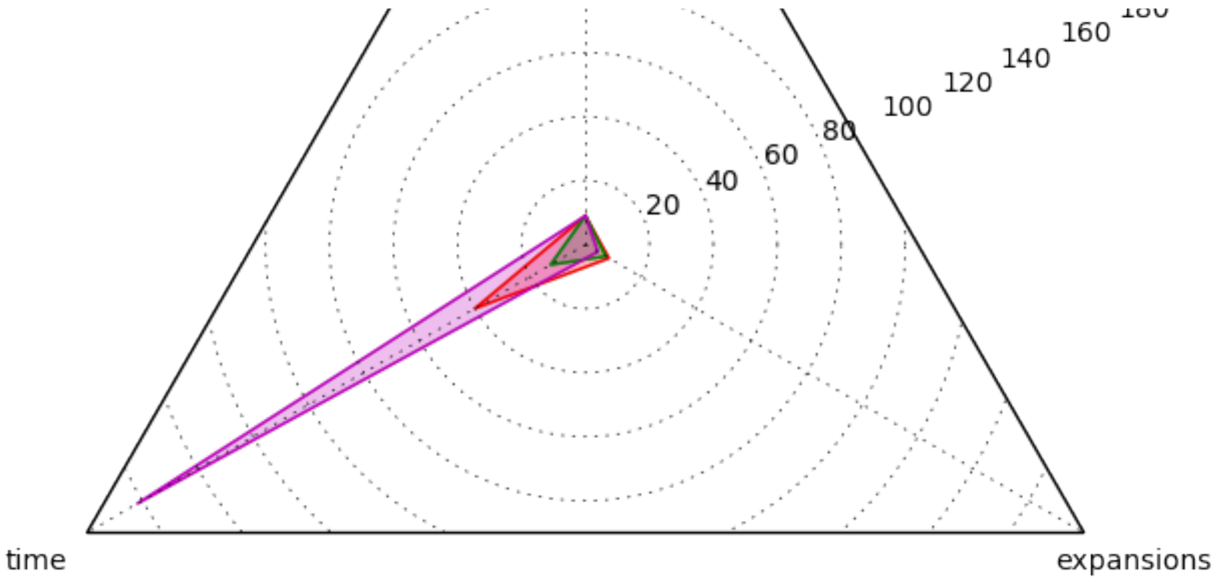
Analysis of A* Heuristics

problem 1

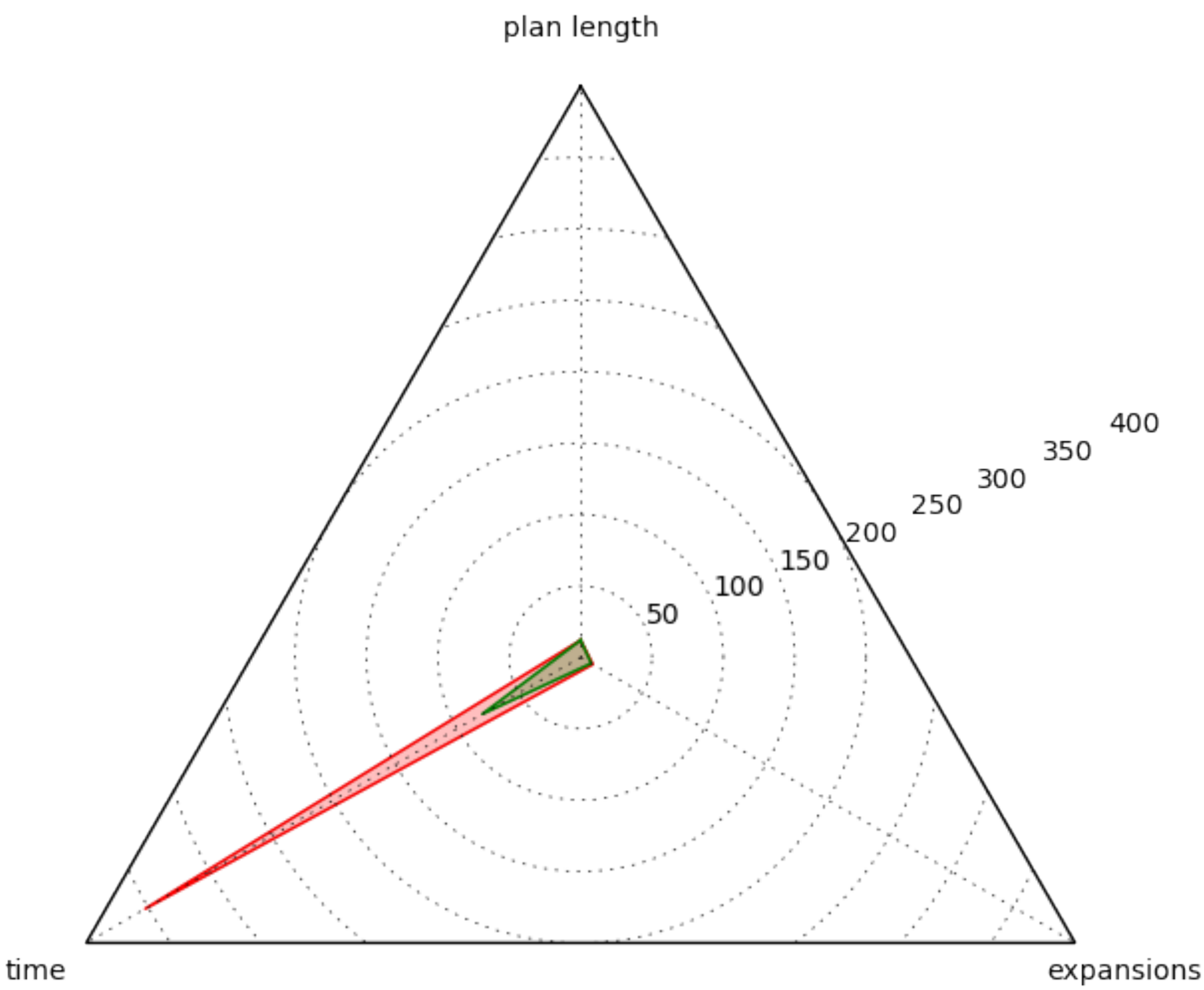


problem 2




20
40
60
80
100
120
140
160
time
expansions

problem 3


plan length
50
100
150
200
250
300
350
400
time
expansions

Heuristics vs non-heristics searches

With a good heuristic, A* search took less time comapred to Uniform Cost Search. It used almot the same time or worse compared to Breadth First Search. If the heuristic was optimized for saving space (pg_levelsum), the time used by A* increased significantly.

In terms of space A* performed better than Uniform Cost Search and Breadth First Search. With a proper heuristic, the space savings were significant.

Recommendations

what's important	search	notes
time to find solution	Breadth First search	or A* with ignore_preconditions
space used to find solution	A* with pg_levelsum	DFS if path len is not a concern
solution length	avoid DFS	other searches returned short solutions
optimal search	BFS, Uniform Cost, A*	avoid DFS