Planning Search - Heuristic Analysis

The deterministic logistics planning problems for an Air Cargo transport system have been solved using a planning search agent. The problems defined in classical PDDL (Planning Domain Definition Language) have also been solved by domain-independent heuristics. Section 2 contains the detailed metrics on the performance of uninformed planning algorithms(BFS, DFS, UCS) and section 3 conatins the metrics for three automatic heuristics (h_1, h_ignore_preconditions, h_level_sum). Appendix-A contians the optimal paths for all three problems.

1. Problem Definitions

- Problem 1 initial state and goal:

```
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 initial state and goal:

```
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 initial state and goal:

```
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))
```

2. Performance of uninformed planning algorithms

Problem	Algorithm	Node Expanded	Goal Tests	Plan Length	Time Elapsed(sec)	Is optimal?
1	BFS	59	109	6	0.121	Yes
	DFS	23	24	22	0.042	No
	UCS	103	105	6	0.175	Yes
2	BFS	9588	21328	9	64.23	Yes
	DFS	206	207	206	2.124	No
	UCS	24317	24319	9	164.511	Yes
3	BFS	84830	171080	12	704.074	Yes
	DFS	299	300	299	3.801	No
	UCS	181426	181428	12	1518.093	Yes

Table 2.1 Metrics of the uninformed searches (BFS, DFS and UCS).

NOTE: The initial implementation of BFS used a list for it's FIFO queue, this was inefficient and took at least 2X longer than UCS to finish. So, the list implementation has been changed to use deque(from collections module) which has faster end operations.

Time Comparison

Fig 2.1 shows the time comparison for all the three problems using BFS, DFS and UCS. We can clearly observe that $t_{DFS} < t_{BFS} < t_{UCS}$, though this is the case in this particular situation it may not be the case in all situations. t_{DFS} may have a significantly greater time or even tend to infinity for some problems. However $t_{BFS} < t_{UCS}$ will typically be true because the time complexities are $O(b^d)$ and $O(b^{d+1})$ respectively. (Note: the time complexity of UCS is approximate. See https://www.ics.uci.edu/~rickl/courses/cs-171/cs171-lecture-slides/cs-171-03-UninformedSearch.pdf for more accurate explaination.)

Nodes Expanded

Fig 2.2 shows the nodes expanded(NE) in all the three problems by each of the uninformed searches. We can see that $NE_{DFS} < NE_{BFS} < NE_{UCS}$. Though the nodes expanded by DFS is least in each of these problems, this might not be the case always. But we can expect $NE_{BFS} \leq NE_{UCS}$ in most of the situations, since NE is directly related to the space complexity of search and the space complexitites are $O(b^d)$ and $O(b^{d+1})$ respectively. (Note: the time complexity of UCS is approximate. See https://www.ics.uci.edu/~rickl/courses/cs-171/cs171-lecture-slides/cs-171-03-UninformedSearch.pdf for more accurate explaination.).

Optimality

Since the step-cost is constant in this problem (all steps can be given a cost of 1, as we are interested in the least no.of steps to solve the problem) BFS is guaranteed to find the optimal path and this is the case as shown in table 2.1 . Typically DFS does not produce an optimal path and this is also seen in the table. This is because in every iteration DFS searches a node that is a level down than the current node, this might make DFS miss the goal on an optimal level and find the goal on a non-optimal level. (See Lesson 8: Search Comparison 3 for a graphical and intutive representation. https://www.youtube.com/watch?v=Fh5b8xVJhR8)

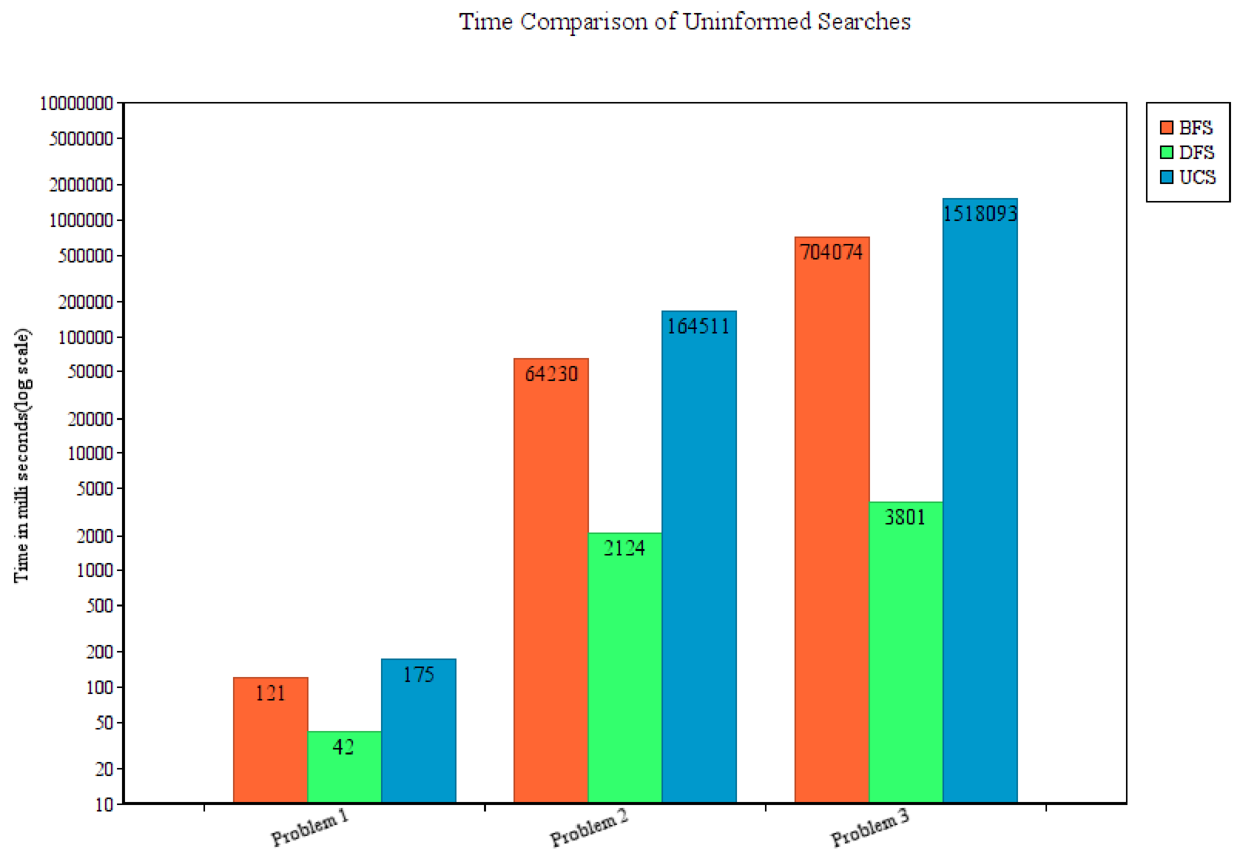

Fig 2.1 Time Comparison of Uninformed Searches

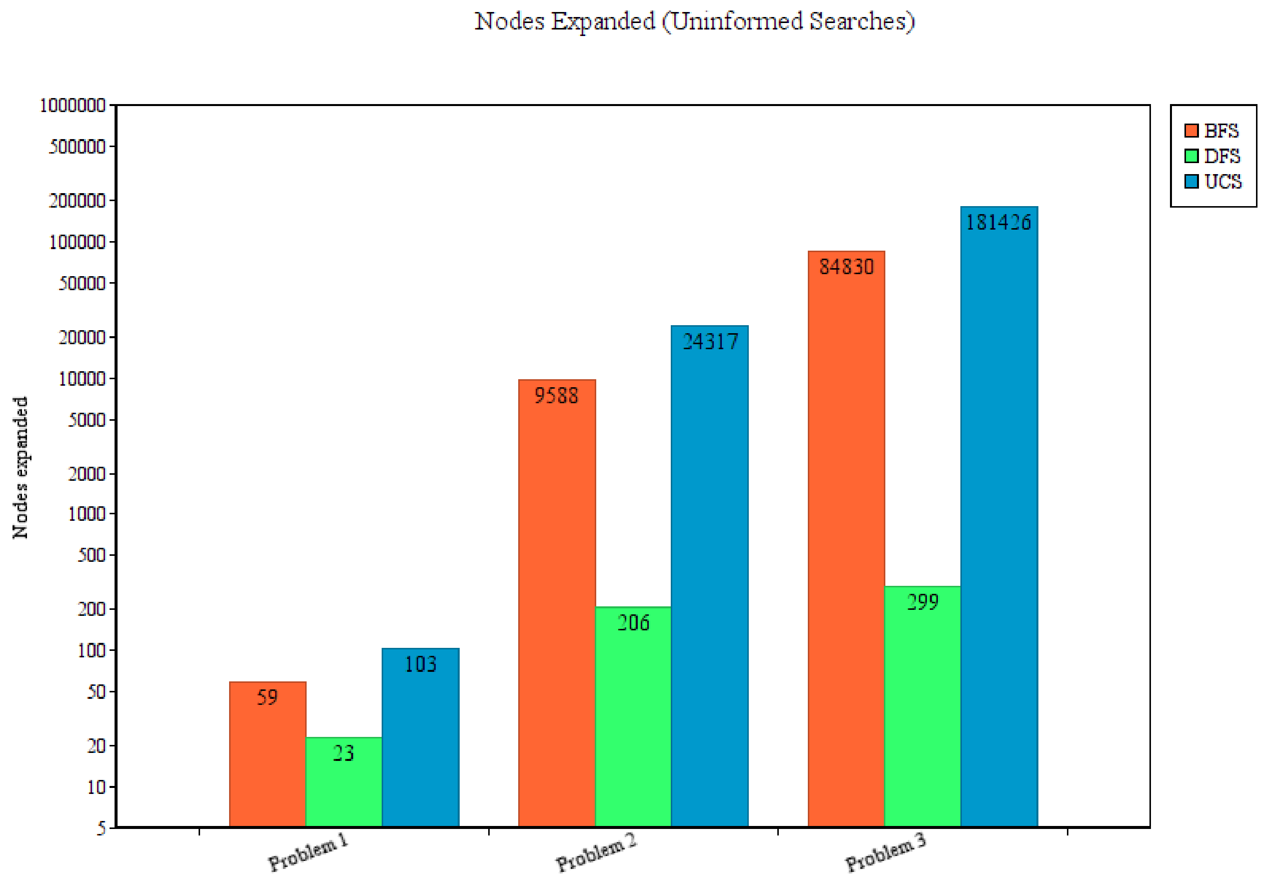

Fig 2.2 Nodes expanded by uninformed searches

3. Performance of automatic heuristics

The following table summarizes the metrics for the three heuristics h_1, ignore_preconditions and level_sum.

Problem	Heuristic	Node Expanded	Goal Tests	Plan Length	Time Elapsed(sec)	Is optimal?
1	h_1	103	105	6	0.18	Yes
	h_ignore_pre_cond	51	53	6	0.127	Yes
	h_pg_level_sum	19	21	6	3.736	Yes
2	h_1	24317	24319	9	163.75	Yes
	h_ignore_pre_cond	2953	2955	9	22.454	Yes
	h_pg_level_sum	127	129	9	505.799	Yes
3	h_1	181462	181482	12	1462.085	Yes
	h_ignore_pre_cond	17315	17317	12	155.684	Yes
	h_pg_level_sum	474	476	12	2740.302	Yes

Table 3.1 Metrics of heuristic searches

NOTE: The h_1 heuristic which always returns a constant (1) is nothing but UCS in these problems, as the step cost is constant.

Time Comparison and Nodes Expanded

Fig 3.1 shows the time comparison for all the three problems using constant, ignore pre conditions and level sum heuristics and Fig 3.2 shows their node expansions. We can clearly observe that $t_{ign\_pre\_cond} < t_{constant} < t_{level\_sum}$. Though the constant heuristic is computationally most effective, it suffers from a huge node expansion overload(because of it’s less accuracy as a heuristic) which increases it’s overall time.

The level sum heuristic, following the subgoal independence assumption, returns the sum of the level costs of the goals; this can be inadmissible but works well in practice for problems that are largely decomposable. It is much more accurate than the ignore pre conditions heuristic. (See Section 10.3 in Artificial Intelligence: A Modern Approach (3rd Edition) (11 December 2009) by Stuart Russell, Peter Norvig).

Though the level sum heuristic is more accurate than the ignore pre conditions heuristic and expands the least number of nodes it’s computationally more costly than ignore pre conditions. So we see a least time on ignore pre conditions.

Optimality

Though the ignore pre conditions heuristic and level sum heuristic aren’t strictly admissible in all domains, we see that they work in this set of problems and produce optimal paths. (See Lesson 8: Optimistic Heuristic for a more detailed and intutive explaination on how heuristics find optimal path. https://www.youtube.com/watch?v=Q5YJtZc37uc)

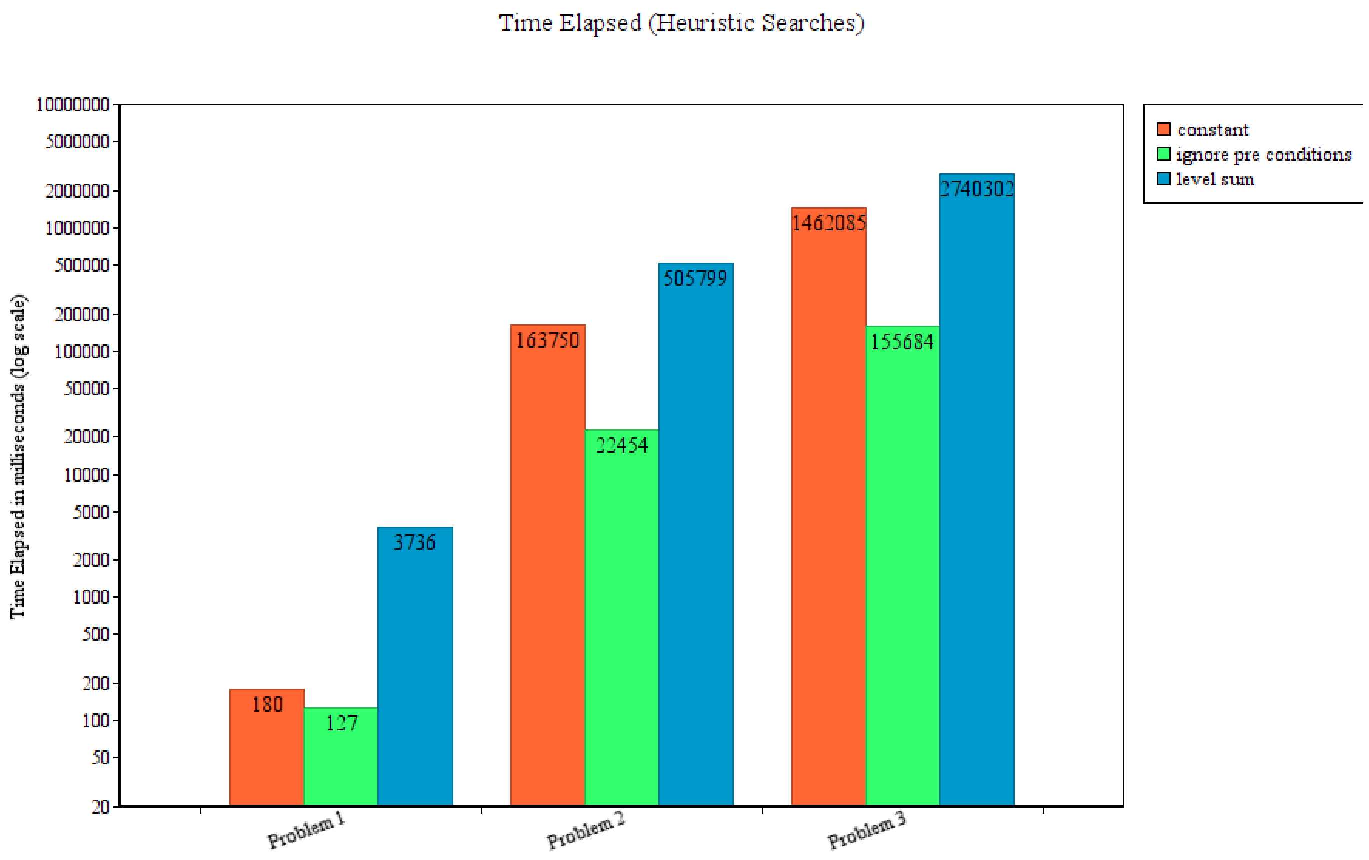

Fig 3.1 Time comparison of different heuristics

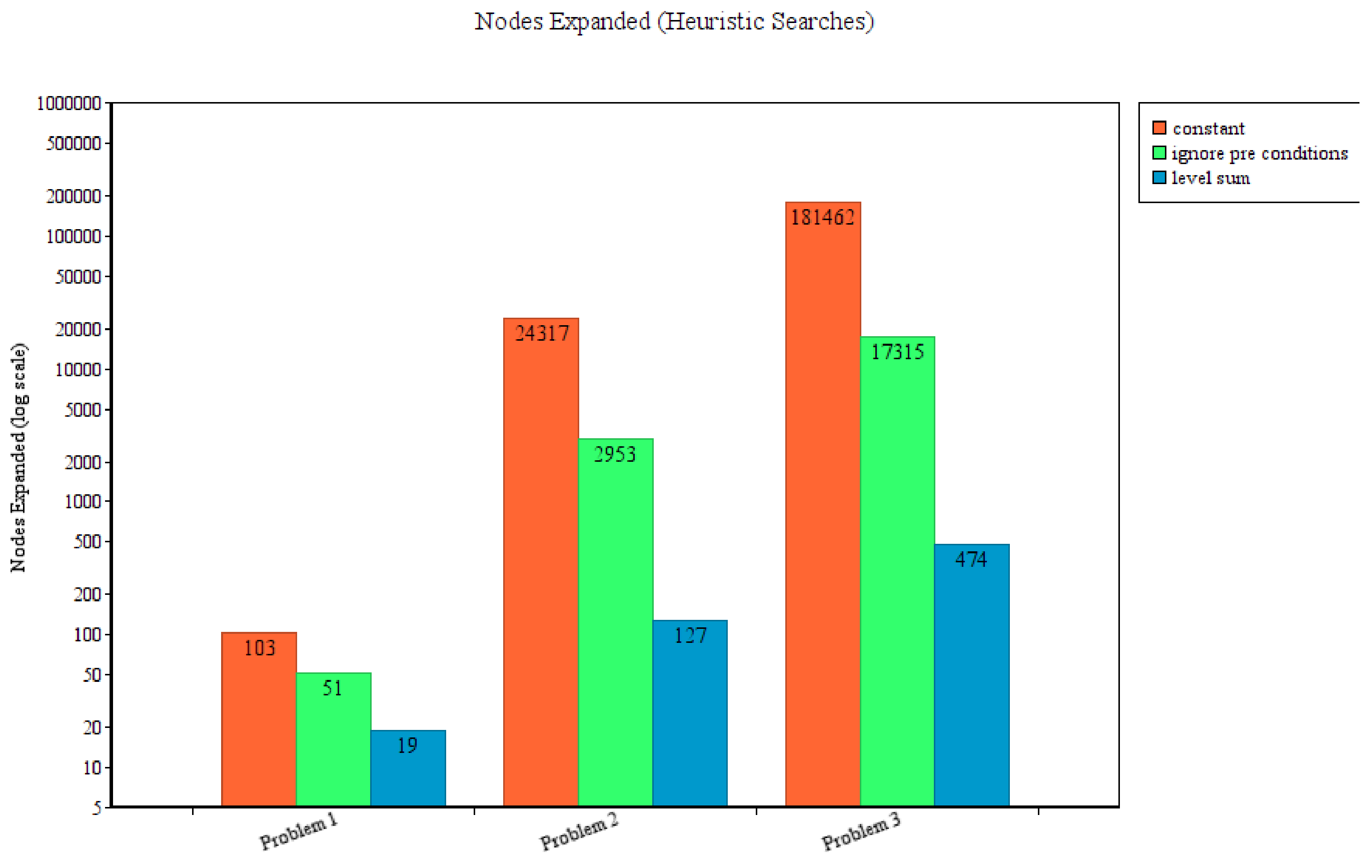

Fig 3.2 Nodes expanded in different heuristics

References

1.) Artificial Intelligence: A Modern Approach (3rd Edition) (11 December 2009) by Stuart Russell, Peter Norvig
2.) Lesson 8: Search
  i) https://www.youtube.com/watch?v=Q5YJtZc37uc
  ii) https://www.youtube.com/watch?v=Fh5b8xVJhR8
3.) Uninformed Search Slides: https://www.ics.uci.edu/~rickl/courses/cs-171/cs171-lecture-slides/cs-171-03-UninformedSearch.pdf

Appendix – A

All the optimal paths below are computed using the ignore pre conditions heuristic

Optimal path of problem 1

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

Optimal path of problem 2

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

Optimal path of problem 3

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(C2, P2, SFO)
Unload(C1, P1, JFK)