Udacity Artificial Intelligence Nanodegree

Planning Search

Author: Ke Zhang

Submission Date: 2017-07-04 (Revision 1)

Analysis of the Air Cargo Transport Heuristics

Synopsis

This paper briefly explains how does our planning search agent solve the deterministic logistics planning problems for an Air Cargo transport system and compares and contrasts the performance of both uninformed and heuristic-based search strategies.

Planning Search Agent and Planning Problems

In the planning search agent implementation, a planning graph is used to provide automatic domain-independent admissible heuristics (e.g. *A* search*). Their performance are then compared against several uninformed non-heuristic search methods (a.k.a. blind search e.g. *breadth-first search*) in the selected planning problems.

Three planning problems from the Air Cargo domain are to be analyze. The problems are defined in classical PDDL (Planning Domain Definition Language). They share the common action schema '*Load*', '*Unload*' and '*Fly*'. The initial states and goals of the problems and the suggested sample solutions by the agent with the optimal plan lengths are listed in the table below. (In each case, the suggested plan is chosen from the best overall search method selected from the next performance comparison chapter.)

Problem 1

Initial States and Goals

```
    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))
```

Optimal Plan

```
    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

Initial States and Goals

```
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))
```

Optimal Plan

```
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)
```

Problem 3

Problem 3

Initial States and Goals	Optimal Plan

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))

```
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)
```

Performance Comparison

Below are the performance metrics of the different search methods for each planning problem grouped by uninformed and heuristic-based search. The performance is compared in terms of execution time, memory usage (expansions and nodes) and path lengths. The optimal search method should have the optimal path length and the least execution time with acceptable memory usage. We value time over memory because the computational power is usually the bottleneck when calculating the search plans in our test environment.

Problem 1

Result Matrix

# [1]	Search Method [2]	Expansions	Goal Tests	New Nodes	Time	Length [3]
1	*Breadth First Search* (U)	43	56	180	0.025s	**6**
2	*Breadth First Tree Search* (U)	1458	1459	5960	0.753s	**6**
3	*Depth First Graph Search<* (U)	12	13	48	0.007s	12
4	*Depth Limited Search* (U)	101	271	414	0.080s	50
5	*Uniform Cost Search* (U)	55	57	224	0.031s	**6**
6	*Recursive Best First Search (h1)*	4229	4230	17029	2.296s	**6**
7	*Greedy Best First Graph Search (h1)*	7	9	28	0.004s	**6**
8	*A* Search (h1)*	55	57	224	0.032s	**6**
9	*A* Search* ($h_{\text{ignore preconditions}}$)	41	43	170	0.034s	**6**
10	*A* Search* ($h_{\text{level sum}}$)	55	57	224	1.163s	**6**

- 1: The overall best search method is underlined and marked bold.
- 2: Type of the search method [U]ninformed or [h]eurisitc-Based abbreviated in brackets.
- 3: The optimal path length is underlined and marked bold.

Performance Analysis

> - Aside from *Breadth First Graph Search* and *Depth Limited Search*, all search algorithms ended with the optimal path length of *6*.
> - Among all search methods with the optimal path length, ***Greedy Best First Graph Search with heuristic 1*** performed best with the fastest execution time and the lowest memory consumption.
> -

Problem 2

Result Matrix

# [1]	Search Method [2]	Expansions	Goal Tests	New Nodes	Time [3]	Length [4]
1	*Breadth First Search* (U)	3343	4609	30509	14.0s	**<u>9</u>**
2	*Breadth First Tree Search* (U)	-	-	-	*timeout*	-
3	*Depth First Graph Search<* (U)	582	583	5211	4.75s	575
4	*Depth Limited Search* (U)	-	-	-	*timeout*	-
5	*Uniform Cost Search* (U)	4853	4855	44041	15.0s	**<u>9</u>**
6	*Recursive Best First Search (h1)*	-	-	-	*timeout*	-
7	*Greedy Best First Graph Search (h1)*	998	1000	8982	2.44s	15
8	*A* Search (h1)*	4853	4855	44041	13.389s	**<u>9</u>**
<u>9</u>	*A* Search ($h_{ignore\ preconditions}$)*	1450	1452	13303	4.12s	**<u>9</u>**
10	*A* Search ($h_{level\ sum}$)*	-	-	-	*timeout*	-

- 1: The overall best search method is underlined and marked bold.
- 2: Type of the search method [U]ninformed or [h]eurisitc-Based abbreviated in brackets.
- 3: Elapsed execution time over *2* minutes.
- 4: The optimal path length is underlined and marked bold.

Performance Analysis

> - The second problem is more complex and caused four of the search algorithms to timeout failures (> 2 minutes execution time).
> - Compared to the previous example, the best search method here *A* Search with ignore preconditions heuristic* required a time 1000-times of previous one, while the optimal path length grown by a factor of 1.5.
> - *Depth First Graph Search* and *Greedy Best First Graph Search (h1)* used both less memory, but didn't return the optimal result.

Problem 3

Result Matrix

# [1]	Search Method [2]	Expansions	Goal Tests	New Nodes	Time [3]	Length [4]
1	*Breadth First Search* (U)	14663	18098	129631	199s	**<u>12</u>**
2	*Breadth First Tree Search* (U)	-	-	-	*timeout*	-
3	*Depth First Graph Search<* (U)	627	628	6176	4.961s	596
4	*Depth Limited Search* (U)	-	-	-	*timeout*	-
5	*Uniform Cost Search* (U)	18223	18225	159618	81.8s	**<u>12</u>**

6	*Recursive Best First Search (h1)*	-	-	-	*timeout*	-
7	*Greedy Best First Graph Search (h1)*	5578	5580	49150	28.0s	22
8	*A* Search (h1)*	18223	18225	159618	78.4s	**12**
9	*A* Search ($h_{ignore\ preconditions}$)*	5040	5042		28.0s	**12**
10	*A* Search ($h_{level\ sum}$)*	-	-	-	*timeout*	-

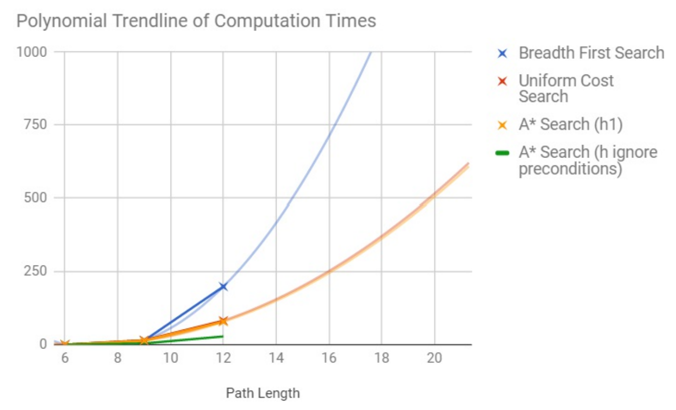
- 1: The overall best search method is underlined and marked bold.
- 2: Type of the search method [U]ninformed or [H]eurisitc-Based abbreviated in brackets.
- 3: Elapsed execution time over *5* minutes.
- 4: The optimal path length is underlined and marked bold.

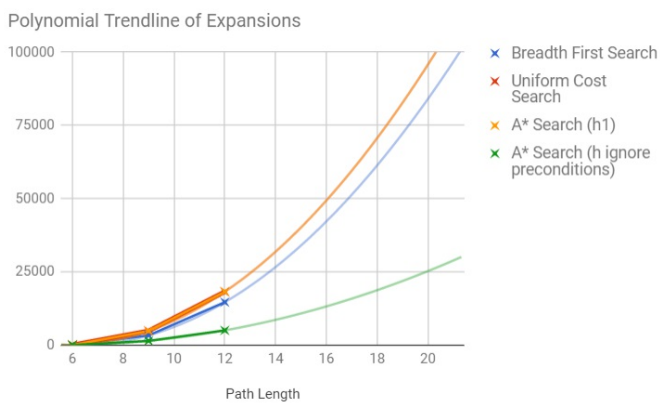
Performance Analysis

> - The last problem is the most complex and caused again four of the search algorithms to timeout failures (this time: > 5 minutes execution time).
> - Compared to the previous example, the best search method here is again the *A* Search with ignore preconditions heuristic*. It dominated the other methods with the correct path length in every domains from number of expansions to time elapsed.
> - Only *Depth First Graph Search* used less expansions to come to a result, but that was unfortunately not optimal.

Summary and Recommendation

The following graph analyzes the algorithms passed all three tests and summarizes the relationship between path length, time and expansions and proposes a polynomial trendline based on the three problem results:





The trendlines clearly illustrates the fact that informed search strategies with custom heuristics gain more and more benefits over uninformed search techniques when the search space gets larger. The benefits are significant both in terms of speed and memory usage.

A complete and optimal uninformed search strategy like *Breadth First Search* struggles a lot with execution speed and memory usage, when it comes to complex planning problems. Since it doesn't have any additional information about the problem beyond the definition and states, it is impossible to bypass this limitation. On the other hand, a complete and optimal heuristic-based strategy like *A* Search* uses problem-specific knowledge and can find solutions more efficiently than the uninformed ones.

Finally, we can conclude, that ***A* Search with ignore preconditions heuristic*** is the fastest search method using the least memory and it would be the perfect choice for the Air Cargo problem.