Heuristic Analysis for Planning Search Project

December 5th, 2017

Jesús Martínez

The focus of this project was to implement a domain-independent planning search agent capable of solving a set of given problems related to an Air Cargo transport system. We used a planning graph and several domain-independent heuristics in tandem with search algorithms such as A*, Breadth-First Search, Depth-First Search and Uniform Search.

Planning Problems

These are the three planning problems in the Air Cargo domain. They use the same 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 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))
```

These goals can be achieved using several different plans, but the optimal lengths for Problem 1, Problem 2 and Problem 3 are 6, 9 and 12, respectively. Here are examples of optimal plans for each problem:

Problem 1 sample plan:

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

Problem 2 sample plan:

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

Problem 3 sample plan:

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

Uninformed Search Strategies Analysis

Uninformed or blind search algorithms are those that have no additional information besides the one coming from the problem definition. They have the ability to test if a state is a goal state or not, and also can generate successor states based on what actions can be performed in a given state.

Our goal in this section is to compare seven uninformed search algorithms in terms of:

- Speed, measured in seconds. Problems time out after 20 minutes.
- Optimality, where a *Yes* means that a path of optimal length was found and *No* means that no optimal path was found.
- Memory usage in terms of node expansions.

Problem 1 results

Algorithm	Optimal	Path length	Speed (seconds)	Node Expansions
Breadth-First Search	Yes	6	0.032	43
Breadth-First Search Tree	Yes	6	0.991	1458
Depth-First Graph Search	No	20	0.016	21
Depth-Limited Search	No	50	0.103	101
Uniform Cost Search	Yes	6	0.039	55
Recursive Best First Search	Yes	6	2.882	4229
Greedy Best First Graph Search	Yes	6	0.005	7

Problem 2 results

Algorithm	Optimal	Path length	Speed (seconds)	Node Expansions
Breadth-First Search	Yes	9	15.429	3343
Breadth-First Search Tree	-	-	TIME OUT	-
Depth-First Graph Search	No	619	3.963	624
Depth-Limited Search	-	-	TIME OUT	-
Uniform Cost Search	Yes	9	16.184	4853
Recursive Best First Search	-	-	TIME OUT	-
Greedy Best First Graph Search	No	21	2.691	998

Problem 3 results

Algorithm	Optimal	Path length	Speed (seconds)	Node Expansions
Breadth-First Search	Yes	12	122.031	14663
Breadth-First Search Tree	-	-	TIME OUT	-
Depth-First Graph Search	No	392	2.108	408
Depth-Limited Search	-	-	TIME OUT	-
Uniform Cost Search	Yes	12	73.934	18235
Recursive Best First Search	-	-	TIME OUT	-
Greedy Best First Graph Search	No	22	21.865	5614

Analysis

As we can see in the results tables above, only two algorithms achieve optimality in all problems without timing out: Breadth-First Search and Uniform Cost Search. The other five algorithms either fail to find an optimal path (Depth-First Graph Search and Greedy Best First Graph Search) or time out (Breadth-First Search Tree, Depth-Limited Search and Recursive Best First Search).

However, depending on the most important factor to us (optimality, time or memory usage), we can favor one algorithm over another.

In case we need optimal paths, Breadth-First Search is the best choice because it saves memory and it is faster than Uniform Cost Search.

On the other hand, if we don't need optimal paths, then Depth-First Graph Search should be our pick because as the problem grows in complexity, it tends to save more memory and provide solutions faster than Greedy Best First Graph Search.

Heuristic Search Strategies Analysis

Unlike blind or uninformed search, heuristic search strategies consider more information specific to the problem than the initial definition of the problem itself. In this section we compare the performance of A* with three different heuristics: H1, Ignore Preconditions and Level Sum. As in the last section we'll compare each A* instance in terms of:

- Speed, measured in seconds. Problems time out after 20 minutes.
- Optimality, where a *Yes* means that a path of optimal length was found and *No* means that no optimal path was found.
- Memory usage in terms of node expansions.

Problem 1 results

Algorithm	**Optimal**	**Path length**	**Speed (seconds)**	**Node Expansions**
A* Search with H1 heuristic	Yes	6	0.040	55
A* Search with Ignore Preconditions Heuristic	Yes	6	0.0395	41
A* Search with Level Sum Heuristic	Yes	6	1.079	11

Problem 2 results

Algorithm	**Optimal**	**Path length**	**Speed (seconds)**	**Node Expansions**
A* Search with H1 heuristic	Yes	9	13.611	4853
A* Search with Ignore Preconditions Heuristic	Yes	9	4.596	1450
A* Search with Level Sum Heuristic	Yes	9	252.938	86

Problem 3 results

Algorithm	Optimal	Path length	Speed (seconds)	Node Expansions
A* Search with H1 heuristic	Yes	12	64.630	18235
A* Search with Ignore Preconditions Heuristic	Yes	12	23.233	5040
A* Search with Level Sum Heuristic	Yes	12	1183.462	318

Analysis

We see that no matter the heuristic, A* always delivers an optimal solution. Regarding speed, A* with H1 and Ignore Preconditions Heuristic are considerably faster than A* with Level Sum Heuristic. In terms of memory usage, the latter outperforms the other two. The more balanced solution in terms of both speed and space is A* with Ignore Preconditions Heuristic because is faster than A* with H1 (and much faster than A* with Level Sum) and consumes less memory than A* with H1.

Comparison between Uninformed and Heuristic Based Search Algorithms

A* with Ignore Preconditions Heuristic is clearly better than Breadth-First Search because even when optimality is guaranteed by both algorithms, the memory footprint of A* with Ignore Preconditions is lesser than that of Breath-First Search, and it’s faster as well.