Heuristics Analysis - Air Cargo Transport Problem

August 13, 2017

1 Introduction

A **Planning Search Agent** to solve deterministic logistics planning problems for an Air Cargo Transport System was implemented for this project based on **Planing Graph** and **Automatic Domain-independent Heuristics with A* Search** given three problems in the Air Cargo domain that use the same action schema. An analysis of the results obtained is provided in this document based on the performace in terms of the number of node expansions required, number of goal tests, time elapsed, and optimality of solution. Foward(progression) state-space search is compared using uniformed and informed algorithms. We will analyize the performance of several uninformed search algorithms—algorithms that are given no information about the problem other than its definition, although some of these algorithms can solve any solvable problem, none of them can do so efficiently. Finally we will also analyize the performance of Informed search algorithms, on the other hand, can do quite well given some guidance on where to look for solutions. For the following section PDDL (Planning Domain Definition Language) is used to describe all possible actions in a single action schema. PDDL describes the four things we need to define a search problem: the *initial state*, the *actions* that are available in a state, the *result* of applying an action, and the *goal test*. We will describe this concepts in specific for each problem.

All problems covered are fully observable, deterministic, static environments with single agents.

2 The Planning problems

A Air cargo transport problem involves loading and unloading cargo and flying it from place to place. The problem can be defined with three actions: *Load, Unload,* and *Fly*. The actions affect two predicates: *In(c, p)* means that cargo *c* is inside plane *p*, and *At(x, a)* means that object *x* (either plane or cargo) is at airport *a*. When a plane flies from one airport to another, all the cargo inside the plane goes with it. In first-order logic it would be easy to quantify over all objects that are inside the plane. But basic PDDL does not have a universal quantifier, so we need a different solution. The approach we use is to say that a piece of cargo ceases to be At anywhere when it is In a plane; the cargo only becomes At the new airport when it is unloaded. So At really means "available for use at a given location."

All problems are in the Air Cargo domain. They have the same action schema defined, but different initial states and goals. The *Air Cargo Action Schema* is the following:

```
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))
```

Fig 1. Action Schema Definition

2.1 What Is Planning?

“Planning is a task of finding a sequence of actions that will transfer the initial world into one in which the goal description is true. The planning can be seen as a sequence of actions generator which are restricted by constraints describing the limitations on the world under view.”

2.2 Uninformed Search Review

"Uninformed search (also called **blind search** has no additional information about states beyond that provided in the problem defintion. ALL they can do is generate successors and distinguish a goal state from a non-goal state" **Section** 8 (*Uninformed Search Strategies, p. 81*). We compare this type of algorithms in the following sections.

2.3 Informed Search Review

"INFORMED SEARCH search strategy—one that uses problem-specific knowledge beyond the definition of the problem itself—can find solutions more efficiently than can an uninformed strategy". **Section** 8 (*Informed(Heuristic) Search Strategies, p. 92*)

2.4 Performance

We have seen AI from the rational point, so in order to determine how good is an intelligent agent we based that in the performace of the agent, so for the following problems we are going to measure the performance of each algorithm in terms of:

- **speed** : algorithm execution time in seconds
- **memory usage** : measured in search node expansions
- **optimality**: "an optimal solution has the lowest path cost among all solutions" **Section** 8 (*Problem-Solving Agents, p. 68*)

3 Problem 1

The first problem has 2 pieces of cargo (C1, C2), 2 airports(SFO,JFK) and 2 planes (P1,P2). Each airport has a single piece of cargo and a plane, which means that Airport *SFO* has cargo C1 and plane P1 and Airport *JFK* has cargo C2 and a plane P2. The goal is to move C1 from SFO airport to JFK and C2 from JFK to SFO.

3.1 Initial States

- SFO has C1 and P1
- JFK has C2 and P2

3.2 Goal

- JFK has C1
- SFO has C2

3.3 Algorithm Comparison

For this problem we can see how **uninformed search algorithms**—algorithms that are given no information about the problem other than its definition, can solve the given problem, nonetheless the solution is not the optimal for all the cases. But it is important to see how some of them like BFS, BFTS and UCS give a optimal solution.

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: [a] complete if b is finite; [b] complete if step costs $\geq \epsilon$ for positive ϵ; [c] optimal if step costs are all identical; [d] if both directions use breadth-first search.

Fig 2. Uninformed search algorithms performance

On the other hand, the performance of Informed search algorithms can do quite well given some guidance on where to look for solutions. Greedy Best First Graph Search (GBFGS + h1) has the best performance of all the algorithms, *achieving the fastest speed with the lowest number of node expansions*. The reasoning is that "Greedy best-first search tries to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly. Thus, it evaluates nodes by using just the heuristic function; that is, f(n) = h(n)." "The worst-case time and space complexity for the tree version is O(bm), where *m* is the maximum depth of the search space. With a good heuristic function, however, the complexity can be reduced substantially. The amount of the reduction depends on the particular problem and on the quality of the heuristic.**Section 8**. We already saw how the performance increase with such hueristic function.

Problem	Algorithm	Optimal	Path Length	Execution Time(s)	Nodes Expanded
1	BFS	YES	6	0.036	43
1	BFTS	YES	6	0.904	1458
1	DFGS	NO	12	0.012	12
1	DLS	NO	50	0.101	101
1	UCS	YES	6	0.042	55
1	RBFS + h1	YES	6	2.617	4229
1	**GBFGS + h1**	**YES**	**6**	**0.005**	**7**
1	A* + h1	YES	6	0.043	55
1	A* + hIgnore	YES	6	0.030	41
1	A* + hLevelsum	YES	6	2.200	11

Table 1. Problem 1 algorithm comparison

3.4 Solution

For this small problem GBF finds the solution easily with minimal node expansion and with the following 6 steps:

1. Load(C1, P1, SFO)
2. Load(C2, P2, JFK)
3. Fly(P1, SFO, JFK)
4. Fly(P2, JFK, SFO)
5. Unload(C1, P1, JFK)
6. Unload(C2, P2, SFO)

4 Problem 2

The second problem has 3 pieces of cargo, 3 airports and 3 planes. It is, Airport *SFO* has cargo C1 and plane P1, Airport *JFK* has cargo C2 and a plane P2 and Airport *ATL* has cargo C3 and a plane P3. Each airport has a single piece of cargo and a plane, and the goal is to get to a state where Airport *SFO* has cargo C2 and C3 and Airport *JFK* has cargo C1.

4.1 Start

- SFO has C1 and P1
- JFK has C2 and P2
- ATL has C3 and P3

4.2 Goal

- JFK has C1
- SFO has C2 and C3

4.3 Algorithm Comparison

As we saw in Problem 1, **uninformed search algorithms**can solve any given problem, but none of them can do so efficiently. For this Problem 2 the number of variables has increased, giving so a larger state space than the one in Problem 1. This is a good way to verify the performance of an algorithm in a wider state space range. For **uninformed search algorithms**, BFS and UCS were the only ones that gave an optimal solution, nonetheless the **space complexity** or **memory usage** is big, for BFS 3401 and for UCS 4853 nodes expanded. Therefore, as it is shown in Fig 2. both algorithms has a exponential time and space complexity.

For all the A* with heuristics the problem was solved optimally. A* with the ignore heuristic is the one that solves the problem optimally and with the less time and space complexity, in 8.045 seconds and with 1450 nodes expanded.

BFTS, DLS, RBFS and A* + hLevelsum took longer than 10 minutes. BFTS and RBFS + h1 search were cancelled because their execution time exceeded 10 minutes; BFTS was running over 40 mins while RBFS + h1 over 1:40 hour, while DLS was running over 35 mins until it solved the problem.

Problem	Algorithm	Optimal	Path Length	Execution Time(s)	Nodes Expanded
2	BFS	YES	9	17.098	3401
2	BFTS	-	-	Timeout	-
2	DFGS	NO	346	1.844	350
2	DLS	NO	50	2110.270	254020
2	UCS	YES	9	15.586	4853
2	RBFS + h1	NO	-	Timeout	-
2	GBFGS + h1	NO	15	4.177	998
2	A* + h1	YES	9	24.971	4853
2	**A* + hIgnore**	YES	**9**	**8.045**	**1450**
2	A* + hLevelsum	YES	9	1539.086	86

Table 2. Problem 2 algorithm comparison

4.4 Solution

The most optimal solution to the problem is solved using **A* + hIgnore precondigtions** in 9 steps:

1. Load(C3, P3, ATL)
2. Fly(P3, ATL, SFO)
3. Unload(C3, P3, SFO)
4. Load(C2, P2, JFK)
5. Fly(P2, JFK, SFO)
6. Unload(C2, P2, SFO)
7. Load(C1, P1, SFO)
8. Fly(P1, SFO, JFK)
9. Unload(C1, P1, JFK)

5 Problem 3

The third problem has 4 pieces of cargo, 4 airports and 2 planes. It is Airport *SFO* has cargo C1 and plane P1, Airport *JFK* has cargo C2 and a plane P2, Airport *ATL* has cargo C3 and Airport *ORD* has cargo C4. Each airport has a single piece of cargo and two have planes, and the goal state is that all cargo is that airport JFK has C1 and C3 and airport SFO has C2 and C4.

5.1 Start

- SFO has C1 and P1
- JFK has C2 and P2
- ATL has C3
- ORD has C4

5.2 Goal

- JFK has C1 and C3
- SFO has C2 and C4

5.3 Algorithm Comparison for non-heuristic planning solution searches

Here we can see how **Uniformed search** has the drawback of enumerating all possible options (big time and space complexity), resulting in an inefficient search. This is obvious now that the state space has increased significantly. For this reason "from 1961 to 1998 forward search was considered too inefficient to be practical"**Section 8** (*p. 6*)

The optimal solution provides a very quick estimate of the how close any given state is to the goal state, is given by A* with the ignore preconditions heuristic.

BFTS, DLS, RBFS and A* + hLevelsum did not complete the problem in less than 10 minutes. BFTS was running over 40 min, DLS over 1 hr and none of them couldnt find the answer so the process was stopped.

Problem	Algorithm	Optimal	Path Length	Execution Time(s)	Nodes Expanded
3	BFS	YES	12	270.757	14629
3	BFTS	-	-	Timeout	-
3	DFGS	NO	2220	64.332	2269
3	DLS	-	Timeout	-	-
3	UCS	YES	12	64.510	18222
3	RBFS + h1	-	Timeout	-	-
3	GBFGS + h1	NO	22	19.403	5569
3	A* + h1	YES	12	68.68	18222
3	**A* + hIgnore**	YES	**12**	**23.713**	**5040**
3	A* + hLevelsum	-	-	Timeout-	-

Table 3. Problem 3 algorithm comparison

5.4 Solution

The most optimal solution to the problem is solved using **A* + hIgnore precondigtions** in 12 steps:

1. Load(C2, P2, JFK)
2. Fly(P2, JFK, ORD)
3. Load(C4, P2, ORD)
4. Fly(P2, ORD, SFO)
5. Unload(C4, P2, SFO)
6. Load(C1, P1, SFO)
7. Fly(P1, SFO, ATL)
8. Load(C3, P1, ATL)
9. Fly(P1, ATL, JFK)
10. Unload(C3, P1, JFK)
11. Unload(C2, P2, SFO)
12. Unload(C1, P1, JFK)

6 Conclusions

We have seen how **planning** allow us to scale up to problems that could not be handled by earlier aproaches (BFS, BFTS, DFGS, DLS, UCS, etc). Forward Search algorithms can take advantage of the representation PDDL uses throuhg accurate heuristics that can be derived automatically from the structure representation. A data structure called the **Planning Graph** was presented which can make the search for a plan more efficient, due to the its polynomial-size approximation of the exponential tree that represents all possible paths rather than just exponential time.

Finally we conclude the section by comparing various approaches were the results presented in Table 1,2,3 clearly illustrate the benefits of using **Informed Search Strategies**, by **doing the stupid thing first, and only add intelligence when necessary Section** 8, over uniformed search techniques when searching for an optimal plan.

Time complexity and space complexity is a huge problem for Uninformed search algorithms while with Informed Strategies we can obtained more reasonable solutions in term of path length, speed and memory usage.

7 Glosary

1. BFS : Breadth First Search
2. BFTS: Breadth First Tree Search
3. DFGS: Depth First Graph Search
4. DLS : Depth Limited Search
5. UCS : Uniform Cost Search
6. RBFS + h1: Recursive Best First Search h1
7. GBFGS + h1: Greedy Best First Graph Search h1
8. A* + h1: A_star Search h1
9. A* + hIgnore: A_star Search hIgnore preconditions
10. A* + hLevelsum: A_star Search hPGLevelSum
11. SFO: San Francisco International Airport
12. JFO: John F. Kennedy International Airport

8 References

[1] S.J. Russell and P Norvig, Artificial Intelligence: A Modern Approach (Prentice Hall, Englewood Cliffs, NJ, 2010) Third Edition.

[2] M. Hebert and A. Procaccia, Graduate AI Lecture 16: Planning 2. Link: http://www.cs.cmu.edu/~arielpro/15780/lec/780-16.pdf

[3] Thad Starner