Artificial Intelligence Nanodegree

Heuristic analysis

Project 3 – Implement a Planning Search

Antonio Ferraioli

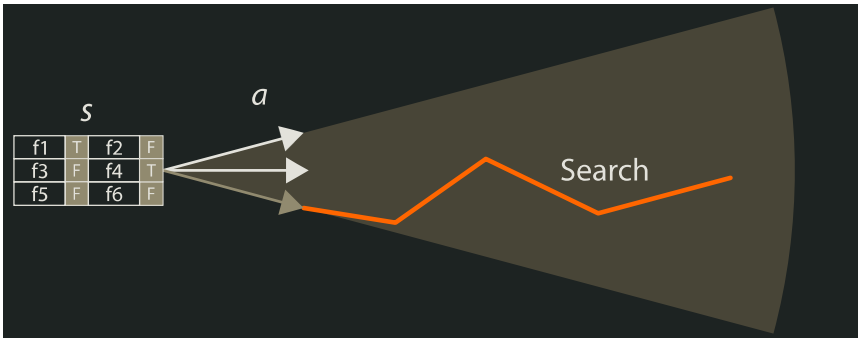
These pages provide a brief discussion about the performance results and heuristics related to the group of problems (three problems) in classical PDDL (Planning Domain Definition Language) for the *air cargo domain* discussed in the AIND lectures.

Air cargo action schema

All problems have the same set of action schemas but different initial states and goals. There are three action schemas (*Load*, *Unload* and *Fly*); actually each action schema formally represents a whole *set* (depending on some variables) of "concrete" actions: each variable assignment (`cargo`, `plane`, `airport`) determines a ground action of the set. Preconditions and effects are conjunctions of positive and negative literals (atomic sentences). The *air cargo action schemas* are 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))
```

Progression planning problems can be solved with graph searches such as breadth-first, depth-first, and A*, where the nodes of the graph are states and edges are actions. A state is the logical conjunction of all Boolean ground fluents, or state variables, that are possible for the problem.



The search algorithm tested were the following:

1	Breadth-first search (BFS)	2	Breadth-first tree search (BFTS)
3	Depth-first graph search (DFGS)	4	Depth-limited search (DLS)
5	Uniform-cost search (UCS)	6	Recursive best-first search with H1 heuristic (RBFS)
7	Greedy best-first graph search with H1 heuristic (GBFGS)	8	A* search with H1 heuristic (AS)
9	A* search with ignore preconditions heuristic (ASIP)	10	A* search with planning graph level sum heuristic (ASPGL)

Problem 1

The first problem concrete interpretation is the task to transport two cargo units (C1 and C2) using two planes (P1 and P2) between two airports (JFK and SFO). The initial state and goal of problem 1 are

```
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)).
```

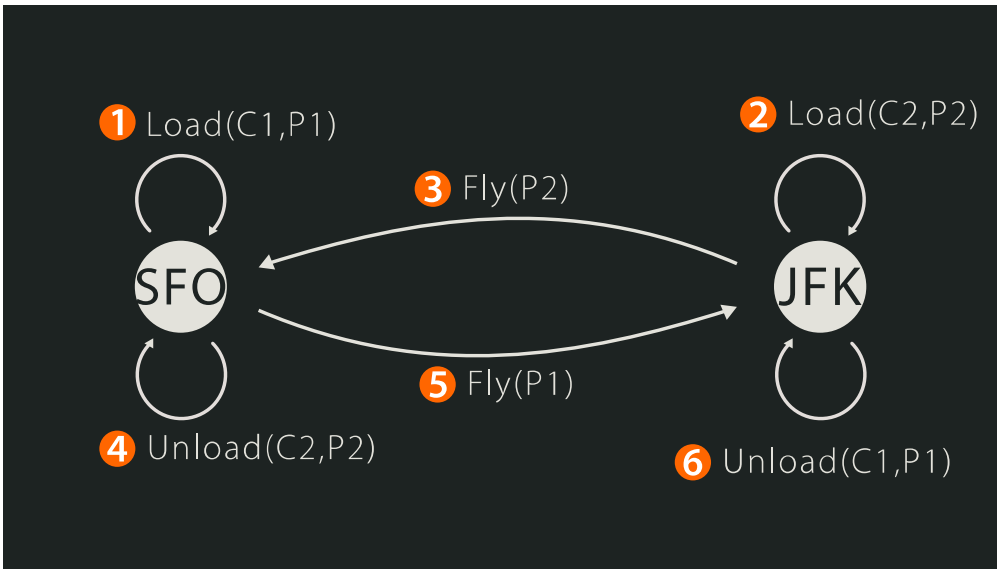
The implementation of the first problem involves states determined by 12 fluents, so the state space size is 2^{12}. A performance comparison test of non-heuristic and heuristic search methods was obtained collecting data after running `run_search.py`. The following table summarizes the test metrics (time elapsed is expressed in seconds, values are truncated to 3 decimal places).

Algorithm	Expansions	Goal Tests	New Nodes	Plan length	Time elapsed	Heuristic
BFS	43	56	180	6	0.484	
BFTS	1458	1459	5960	6	1.252	
DFGS	21	22	84	20	0.019	
DLS	101	271	414	50	0.134	
UCS	55	57	224	6	0.049	
RBFS	4229	4230	17023	6	3.619	H1
GBFGS	7	9	28	6	0.006	H1
AS	55	57	224	6	0.049	H1
ASIP	41	43	170	6	0.054	IP
ASPGL	11	13	50	6	1.396	PGL

There are several optimal solutions for problem 1. From a syntactic point of view these optimal sequences are pretty equivalent but if some additional conditions were provided (for example, if typical action timings were implemented), then some sequences would have been preferable. An example of optimal solution for problem 1 is the one given by breadth-first search (see *non-heuristic search algorithms comparison* paragraph for details on breadth-first search optimality). It is a 6 actions plan:

```
Plan length: 6
Load(C1, P1, SFO)
Load(C2, P2, JFK)
Fly(P2, JFK, SFO)
Unload(C2, P2, SFO)
Fly(P1, SFO, JFK)
Unload(C1, P1, JFK).
```

The figure below shows a graph representing the optimal solution.



Problem 2

The second problem has the following 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)).
```

Each state in this problem is determined by 27 fluents, so the state space size is 2^{27} (here complexity is higher). The test results are the following (algorithms tests with execution time > 30 minutes were canceled).

Algorithm	Expansions	Goal Tests	New Nodes	Plan length	Time elapsed	Heuristic
BFS	3343	4609	30509	9	21.078	
BFTS	-	-	-	-	-	
DFGS	624	625	5602	619	5.348	
DLS	222719	2053741	2054119	50	1677.328	
UCS	4852	4854	44030	9	21.732	
RBFS	-	-	-	-	-	H1
GBFGS	990	992	8910	21	4.365	H1
AS	4852	4854	44030	9	21.560	H1
ASIP	1450	1452	13303	9	7.849	IP
ASPGL	86	88	841	9	143.428	PGL

An optimal solution is

```
Plan length: 9
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

The initial state and goal of problem 3 are

```
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)).
```

Each state in this problem is determined by 32 fluents, so the state space size is 2^{32}, the most cumbersome of the three problems. The test results are the following (algorithms tests with execution time > 30 minutes were canceled).

Algorithm	Expansions	Goal Tests	New Nodes	Plan length	Time elapsed	Heuristic
BFS	14663	18098	129631	12	256.726	
BFTS	-	-	-	-	-	
DFGS	408	409	3364	392	4.941	
DLS	-	-	-	-	-	
UCS	18234	18236	159707	12	134.501	
RBFS	-	-	-	-	-	H1
GBFGS	5605	5607	49360	22	38.450	H1
AS	18234	18236	159707	12	126.382	H1
ASIP	5040	5042	44944	12	41.708	IP
ASPGL	325	327	3002	12	762.349	PGL

An optimal solution is

```
Plan length: 12
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).
```

Non-heuristic search algorithms comparison

Three blind search algorithms are examined: breadth-first search (BFS), depth-first graph search (DFGS), uniform-cost search (UCS).

BFS

- *Optimality*: if an optimal plan exists, BFS will find it. This is because BFS is an optimal search. When all step costs are equal, "optimal solution" means "shortest path solution": BFS is always expanding the shortest path first and so wherever the goal is hiding, it's going to find it by examining no longer paths (see *Search Comparison* lessons or pages 81–83 of the *AIMA* textbook, 3rd edition).

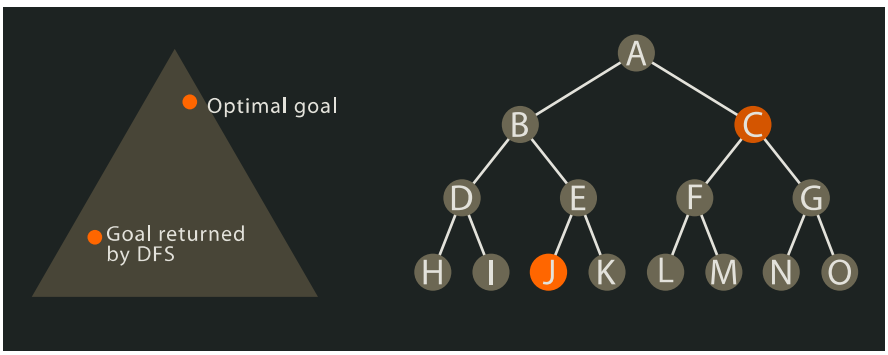
- *Time elapsed*: BFS is certainly slower than depth-first graph search and it is slower than UCS on problem 3.
- *Node expansions required*: BFS, on average, involves a large number of node expansions. The expansions required are more than those required by DFGS and slightly less than those required by UCS.
- *Reasons for the observed results*: the reason for the large number of node expansions is that the algorithm intrinsically has high complexity. Suppose that each state has *b* successors and the solution is at depth *d*; then the total number of nodes generated is

$$b + b^2 + b^3 + \cdots + b^d = O(b^d).$$

  In other words, if solution depth and branching are large, then a huge number of nodes must be considered. The memory requirements are even a bigger problem for breadth-first search than is the execution time. For breadth-first graph search in particular, every node generated remains in memory. There will be $O(b^{d-1})$ nodes in the explored set and $O(b^d)$ nodes in the frontier (*AIMA* 3rd edition, page 83). BFS is slowest than UCS in problem 3; the reason lies in implementation differences: BFS uses a FIFO queue, UCS uses a priority queue.

DFGS

- *Optimality*: DFGS is non-optimal. Generally, the properties of depth-first search depend strongly on whether the graph-search (like DFGS) or tree-search version is used. Both versions, however, are *non-optimal* because depth-first search will explore the entire left subtree even if a node C, placed on the right at a shallow level (see figure below), is a goal node. If a node J were also a goal node lying on the left, then depth-first search would return it as a solution instead of C, which would be a better solution; hence, depth-first search is not optimal (*AIMA* textbook, 3rd edition, page 86). So there is a huge disadvantage in using DFGS because solutions sequences are longer and involved (for example, it is hard to make a sense of a 392 step solution for problem 3).



- *Time elapsed*: the test results show that DFGS is the fastest algorithm.

- *Node expansions required*: even in terms of expanded nodes and goal tests, it is the cheapest algorithm.
- *Reasons for the observed results*: from the test results, DFGS is the fastest and cheapest algorithm. In particular, DFGS is faster and cheaper than BFS. Lesson videos (S*earch Comparison*) and *AIMA* textbook (3rd edition, page 86) apparently seem to contradict this claim because (talking about differences with BFS) "for a *graph search* there is *no advantage*, but a depth-first tree search needs to store only a single path from the root to a leaf node, along with the remaining unexpanded sibling nodes for each node on the path. Once a node has been expanded, it can be removed from memory as soon as all its descendants have been fully explored". However, there is a difference of implementation. In fact, BFS uses a FIFO data structure, DFGS uses a LIFO data structure (*AIMA* textbook, 3rd edition, page 85). Moreover, there are some performance discrepancies that depend on the frequency of solutions and their locations. For example, if solutions are frequent and located deep in the tree, DFGS could be faster than BFS.

UCS

- *Optimality*: UCS is optimal provided that cost function $g(n)$ is non negative. Optimality of UCS follows from the fact that whenever uniform-cost search selects a node n for expansion, the optimal path to that node has been found. Then, because step costs are nonnegative, paths never get shorter as nodes are added. These two facts together imply that uniform-cost search expands nodes in order of their optimal path cost. Hence, the first goal node selected for expansion must be the optimal solution (*AIMA* textbook, 3rd edition, page 85).
- *Time elapsed*: UCS is slower than DFGS, and is faster than BFS on problem 3 (on problems 1 and 2 the performance is quite similar).
- *Node expansions required*: the number of expansions is slightly larger than the BFS expansions.
- *Reasons for the observed results*: the different number of expansions with respect to BFS algorithm (UCS expands more nodes than BFS) is explained considering two significant differences from breadth-first search (*AIMA* textbook, 3rd edition, pages 83–84):
  - the goal test is applied to a node when it is *selected for expansion* rather than when it is *first generated*. The reason is that the first goal node that is generated may be on a suboptimal path;
  - a test is added in case a better path is found to a node currently on the frontier.

  In other words: after finding a path to a goal state, UCS keeps searching expanding new nodes and adding other paths; then it checks if these new paths are cheaper than the old one.

Heuristic search algorithms comparison

Two heuristic search algorithms are examined: A* search with ignore preconditions heuristic (ASIP) and A* search with planning graph level sum heuristic (ASPGL).

ASIP

- *Optimality*: ignore precondition heuristic is – in general – not admissible. However, since in this setting (1) is never true that some action may achieve multiple goals and (2) it is ignored that some actions may undo the effects of others, the heuristic implemented is accurate and in tests ASIP found optimal plans for the examined problems (*AIMA* textbook, 3rd edition, page 376).
- *Time elapsed*: overall ASIP performs quite good, it is faster than ASPGL and even than non-heuristic algorithms examined before (BFS, DFGS and UCS).
- *Node expansions required*: ASIP operates less expansions than most of the algorithms examined (only ASPGL is cheaper).
- *Reasons for the observed results*: think of a search problem as a graph where the nodes are states and the edges are actions. Then ignore precondition heuristic adds edges to the graph making the problem easier because it becomes easier to find a path (*AIMA* textbook, 3rd edition, page 376).

ASPGL

- *Optimality*: level sum heuristic is not admissible (optimality is not guaranteed) but works well in practice (*AIMA* textbook, 3rd edition, page 382). In fact, ASPGL found optimal solutions for all three problems.
- *Time elapsed*: ASPGL is the second slowest algorithm tested for all three problems. It is slowest than ASIP, BFS, UCS, DFGS.
- *Node expansions required*: for this feature ASPGL is the best algorithm tested, ASPGL expands a small amount of nodes compared to other algorithms.
- *Reasons for the observed results*: ASPGL needs to construct a *planning graph*, this causes a longer execution time. A planning graph is polynomial in the size of the planning problem. For a planning problem with *l* literals and *a* actions, an entire graph with *n* levels has a size of $O(n(a + l)^2)$. The time to build the graph has the same complexity (AIMA textbook, 3rd edition, page 381). ASPGL is memory-efficient; the reason is in the high accuracy of level sum heuristic in estimating the cost of a conjunction of goals.

Conclusion

Overall, the best algorithm tested is A* search with ignore preconditions heuristic (ASPI). This is a quite fast and memory-efficient algorithm that returned optimal plans for all the problems tested. The best algorithm (an informed one again) in terms of memory/space is ASPGL, it produces optimal plans but is really slow.